PLVC

Technical Documentation

Release: 2.0.6-0-g5b0081f (2019-08-12)

HAWE HYDRAULIK

Copyright

Printdate: August 12, 2019

Contents

List of Figures

List of Tables

Part I.

Introduction

1 Safety

1.1 General

The programmable valve control type PLVC comes generally with an operating system. For the actual application a user-specific software has to be generated and transfered to the control system. The responsibility for ensuring proper and error-free operation of the end application is with the user of PLVC.

Attention:

When replacing a PLVC, you have to order hardware components, the current software version and the parameters set by the manufacturer of the machine! To ensure safe operation of the user-generated application programs there is - if necessary - a final acceptance test of the machine in accordance of national standards to be done by a monitoring organization.

1.2 Responsibility

This description is an integral part of the device. It contains information regarding the correct handling of the PLVC and must be read prior to installation or use. Follow the instructions in the description. Non-compliance with the notes or any use outside the intended usage outlined in the following, wrong installation or faulty handling can seriously impair and endanger the safety of people and machinery and will result in the exclusion of any liability and warranty claims. This manual is addressed to individuals who can be deemed to be “knowledgeable” in the understanding of the EMC- and the low-voltage guideline. The wiring of the valves must be performed by an electrician and must be activated by trained programmers and/or service technicians.

2 Basic information on the PLVC

Valve controls like PLVC are complex PLC-enabled micro-controller controls with integrated proportional amplifiers for mobile and stationary applications in hydraulics. The operating range for these controls is broad, including:

- Cranes
- Construction machines
- Complex hoists
- Forestry machines
- Clamping hydraulics for machine tools
- Molding press

The various control tasks can be implemented by:

- A module construction with various extensions and add-ons
  - Basic module
  - Expansion modules (additional inputs and outputs)
  - Extensibility via CAN bus
- A flexible programming to IEC 61131-3 (PLC programming using structured text ST)
- Free configuration of all inputs and outputs as well as full diagnostic capability and short circuit resistance
- Remote diagnosis via external modem (or mobile phone with integrated modem)
- Combination of several valve controllers via CAN bus in a system for control of complex systems

It will meet all relevant standards regarding personal safety, EMC, vibration and shaking.

3 Important Hints for Installation and Operation

NOTE

To ensure safe operation of the controller the instructions below must be guaranteed:

3.1 Installation

- When installing avoid high heat (e.g. exhaust).
- The distance to radio devices must be sufficiently large.
- Signal lines should not be run near power cables.
- The cable break detection and short-circuit detection for signal lines should be used.
- When installing a control system, the ground wire of power supply should be distributed as near as possible to the control (See figures 3.1 and 3.2).

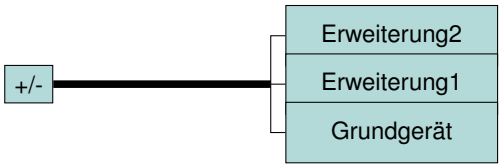


Figure 3.1.: Correct Wiring

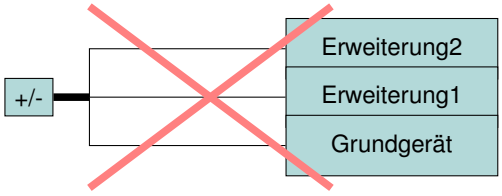


Figure 3.2.: Incorrect Wiring

- The supply voltage of each group should be protected separately.
- A shielding of the signal lines is recommended.
- The signal and the ground of the sensors should be put on the same terminal block of the device.

3.2 In Operation

- The operation of the control can only be guaranteed in a temperature range of $-40°C$ to $+80°C$.
- The device may become hot during operation and cause danger of burning in case of contact!
- Before electric welding on the machine, all connectors from the PLVC should be disconnected (power, signal lines) and/or electrical isolation should be ensured.

4 Electronic Installation

- Connect metal housing with ground, select the shortest connection between housing and machine (independently from the negative pole of power supply)
- Wiring in accordance with safe extra low voltage and isolated from other circuits
- Incorrect wiring can cause unexpected signals at the outputs of the controller

WARNING

A parallel interconnection of external power sources (e.g. emergency operation by buttons) and outputs of the PLVC is not allowed!

- Note the application-related documents (wiring diagrams, software descriptions, etc.)
- Provide the wiring recommendations.
  Power Supply X30, relay ports base device and POW: $>= 1mm^2$
  Other inputs and outputs: $>= 0.5mm^2$
- Do not place wires to electronics in the vicinity of other power lines in the machine
- Inductive consumers, which are not connected to the PLVC, must be wired with spark diodes close to the inductance.
- The outputs of the PLVC are partly equipped with internal spark diodes. (See chapter Outputs (5.2)).
- Use only HAWE SE approved accessories
- An emergency shutdown of the power supply is provided. The emergency-stop-switch must be installed on the machine and should be easily accessible for the operator. The achievement of a safe state by pressing the emergency-stop-switch must be guaranteed by the manufacturer of the machine.

5 Components of the Control System

5.1 Serial Interface

The basic device of PLVC has a serial interface.

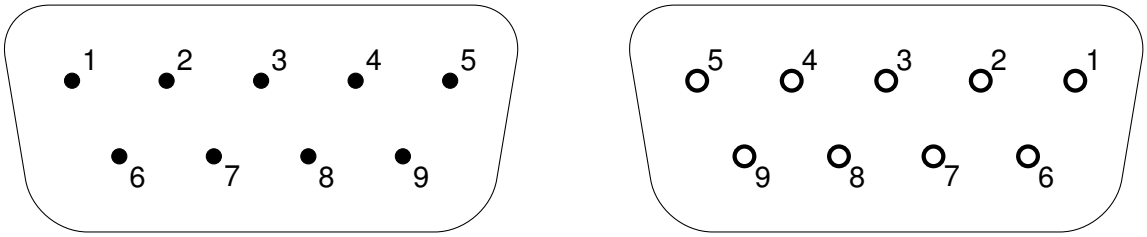


Figure 5.1.: Pin assignment of 9-pin D-Sub connector and mating connector

Via serial interface you can:

- Querry currently signals at the PLVC
- Make settings for proportional outputs and analog inputs
- Plot charts (oscilloscope function in Visual Tool)
- Make settings for CAN-Bus/Profibus

The PLVC control is connected to the PLVC via standard serial 9-pin cable and corresponding adapter.

The adapter can easily be made by yourself. You need a 9-pin D-sub connector. The wires for RX, TX and GND should be soldered to the pins 2, 3 and 5. The appropriate wires are then connected to the terminal block of the PLVC in accordance with the pin description lists.

Function	Pin at D-sub connector
RX	3
TX	2
GND	5

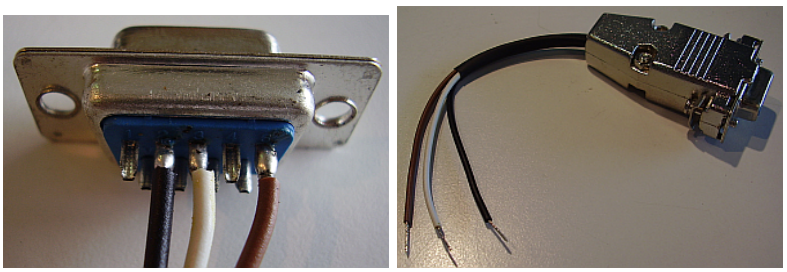


Figure 5.2.: Serial Interface Adapter

The transmission rate can be set between 9600kBd and 57000kBd.

5.2 Outputs

Current Controlled Proportional Outputs

Current controlled proportional Outputs serve to control solenoid valves. The current is changed by pulse width modulation (PWM) and controlled by back measurement.

The PWM frequency is 1kHz. The clock ratio can be adjusted from 10% to 94%. So there is a control range from ca. 100mA to 1.8A depending on operating voltage and coil resistance. Dither frequency (on and off switching frequency) and dither amplitude are also adjustable.

These outputs do have an internal free-wheeling diode.

Digital PWM Outputs

Digital PWM outputs can be used as black/white outputs or as kind of proportional. There is no back measurement of the current. But the voltage is monitored to detect short circuits. The dither frequency can be switched between 50Hz and 100Hz. The PWM duty cycle can be set from 5% to 100% in 5% increments.

These outputs do have an internal free-wheeling diode.

Relay Outputs

Relay outputs offer floating make or change-over contacts.

The maximum switching current is 5A.

WARNING

It is strongly recommended to protect the individual relays with fuses. To actuate inductive loads freewheeling diodes have to be used for not damaging electronics.

5V Output

The 5V output serves to supply sensors and joysticks with voltage.

The maximum load depends on the using module.

The 5V output is monitored internally. Voltage fluctuations are recorded and the sensor signals of the connected devices can automatically be adjusted to the fluctuating supply voltage. This means that a change from the 5V output voltage still guarantees a stable sensor signal.

Programmable 10V Output

The programmable 10V output offers an output signal to control measuring instruments with high resistance. This output has a maximum load of 10mA.

Constant 10V Output

The constant 10V output serves to supply sensores and joysticks with voltage. The maximum load of this output is 200mA.

Digital Auxiliary Outputs

The digital auxiliary outputs can be used to control e.g. small lamps or relais.

The maximum load depends on the using module.

Older modules do not have free-wheeling diodes. So the inductive consumers must be wired accordingly.

5.3 Inputs

Emergency-Stop-Input

The emergency-stop-input serves to power the proportional valve outputs. It must be supplied with 10-30V, otherwise the proportional outputs will be shut down.

Resetting of the emergency-stop function comes with reboot of the controller. It can also be reset by using the software via special parameters.

Analog Inputs

Analog inputs serve to process analog (fluctuating) signals (joystick, potentiometer, pressure transmitter, length sensor). All sensors, providing an output signal of 0-10V or 4-20mA, can be connected to the PLVC.

The particular configuration of the analog inputs on the PLVC is done by soldering jumpers or parameters depending on the using module.

With the power supply of analog sensors, attention must be paid to a suitable ground reference. Otherwise, the corresponding sensor signal is distorted.

The input impedances can be seen on the terminal diagrams of the modules.

Digital Inputs

The digital inputs detect switching signals of mechanic and electronic switches. The input impedances of the digital inputs can be seen in the terminal diagrams of each module.

The switching threshold is at high-min > 4V and low-max < 1V.

Frequency Inputs

The frequency inputs detect frequencies up to 5kHz. The input impedances of the digital inputs can be seen in the terminal diagrams of each module.

Part II.

Hardware

6 PLVC21

6.1 Technical Data

6.1.1 General Data

Enclosure	IP 20 to IEC 60529
Temperature range	$-40°C$ to $+80°C$
Supply voltage	10V DC to 30V DC
Max. total electricity	Basic module: 10A
Required external fuse	Basic module: 10A, slow-blow
Protection	Against reverse polarity Against load dump (DIN 40839) Vibration resistance (vibration: IEC 68-2-6, shock: IEC 68-2-27) EMV (EN 61000-6-4, EN 61000-6-1, EN 61000-6-2, EN 61000-6-3)
Monitoring	Short-circuit Undervoltage, overvoltage Cable break
Cables connectors	using spring-cage connection phoenix with cable section up to $1,5mm^2$
Microcontrollers	80C167, 16bit
Basic parameter memory	EEPROM 256 words
Memory	Flash: 256kByte RAM: 128 kByte
Fixing	Snap-on housing Phoenix for top-hat rail
Housing material	Plastic, cover anodised alminium
Ground (weight)	Ca. 0,3kg (basic module) Ca. 0,1kg (extension module)

6.1.2 Base Unit PLVC21

Features of the basic device PLVC21

- 4 analog inputs (for joystick, potentiometer, sensors, e.g. analog pressure sensors)
- 5 digital inputs (for limit switches, pressure switches, buttons, etc.)
- 3 frequency inputs (for encoder, tachometer, incremental encoder, etc.)
- Emergency-stop-input (opto-decoupled)
- Interface for RS232 and PROFIBUS
- 4 outputs for proportional or b/w-valves (current regulated, highside) 2A
- 8 digital outputs for ohmic or inductive consumers 1,2A
- Power supply 10-30V DC, max 5A

Performance of the Connections

Terminal block	Function	Description	Parameters
X 1	Power supply	Nominal voltage U_N max. total current	10-30V DC 5A
	prop. bzw. b/w-Outputs 0-3 (with high-side-Messung)	I_{min} I_{max} Ditherfrequency Dither amplitude (referring to PWM) cold resistance	100...1200mA 100...2200mA 25...200Hz 0...48% 2...35$k\Omega$
	frequency inputs 0-2	cutoff frequency	$f_{Grenz} = 5kHz$
	digital inputs 0-4	voltage range Debounce for rising/falling signal edge can be switched separately	$10...30VDC/5k\Omega$
	digital outputs 0 . . . 7	for b/w - valves and ohmic consumers	10 . . . 30 V DC / 1,2A
	Emergency-stop input	opto-decoupled	
	Interface RS232	Interface parameters	19,2kBd
	analog inputs 0-3 (for joysticks, potentiometer, sensors etc) area monitoring	10 bit DC $\hat{=}$ 1024 steps	4 . . . 20 mA 0...10V DC (default) 0 . . . 5 V DC
X 2	Interface PROFIBUS	DP-Slave	max. 6MBd

Table 6.2.: Performance of the Connections

6.1.3 Extension Module

General Properties

Supply voltage	10-30VDC
Max. total current	10A
Necessary external security	10A, slow-blow
Fixing	integrated in basic system

Table 6.3.: General Properties of the PLVC Extensions

Features of the Extension Module

- 8 analog /digital inputs
- 8 digital outputs for ohmic or inductive consmuers 1,2A
- CAN-Bus
- Power supply 10-30V DC, max. 5A

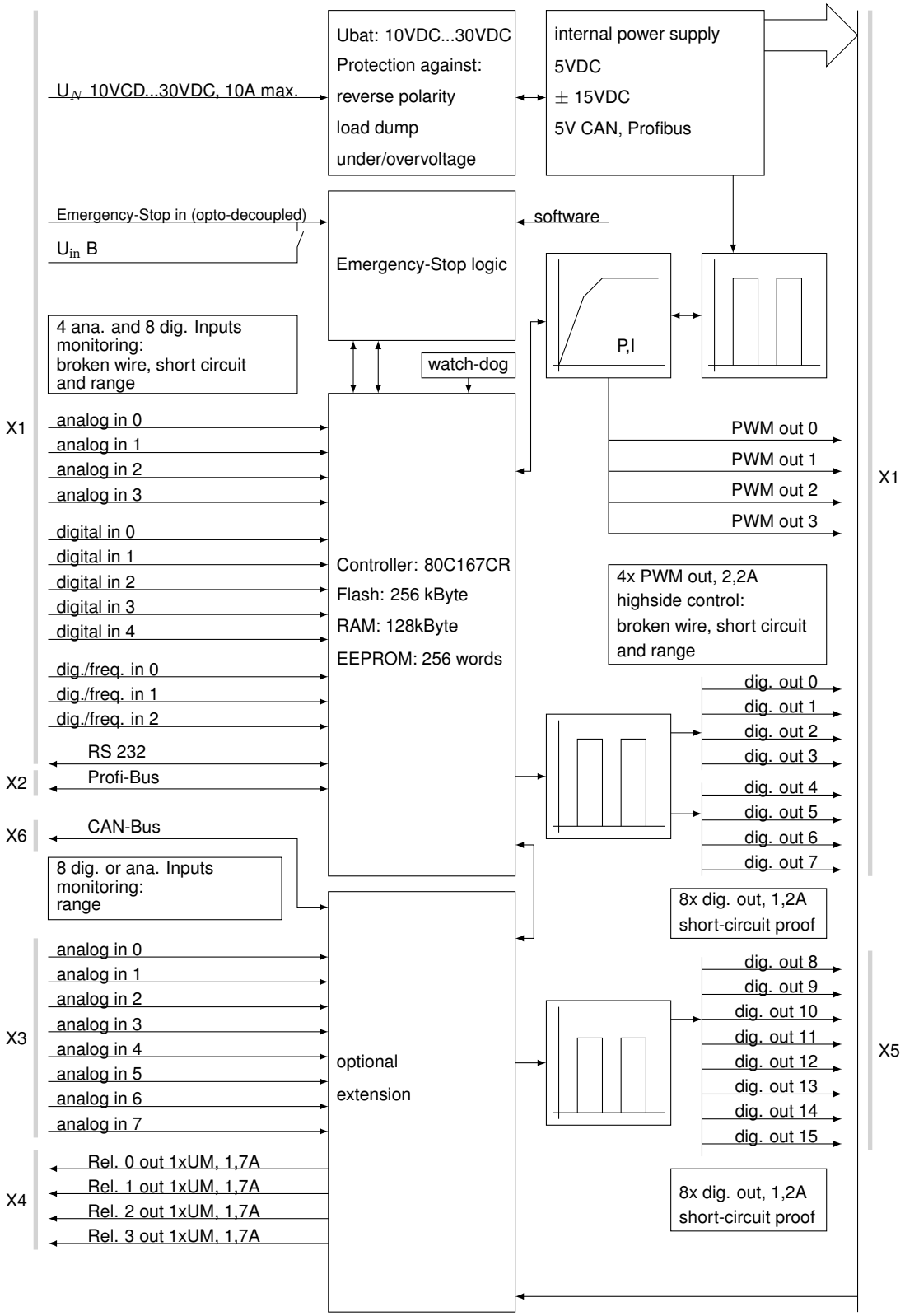
Attention: The Emergency-stop on the basic-module doesn't influence the digital outputs of the expansion modules.

Performance of the Connections of the Extension

Terminal-block	Function	Description	Parameters
X 1	power supply	nominal voltage U_N max. total current	10...30V DC 10A
X 3	8 analog inputs	10bit A DC $\hat{=}$ 1024 steps	4...20mA 0...10V DC (default) 0...5V DC
X 5	digital outputs 8-15	for b/w-valves and ohmic consumers	10...30V DC / 1,2A
X 6	CAN-Bus-connection		100, 125, 250, 500kBd

Table 6.4.: Performance of the connections

6.1.4 Block Diagram - Basic device with extension



6.2 Download an Operating System

The operating system can be updated using a Windows PC (or Windows laptop).

6.2.1 With an Intact Operating System

A new operating system is easy to install via an already existing operating system. All functionality for an update is already included in the existing operating system. Connect the PLVC control via serial interface to the PC and start the appropriate upload program of the operating system.

6.2.2 Damaged Operating System

If you cannot activate your current operating system (e.g. by an aborted operating system update), a new operating system can be installed though.

For this, the PLVC must be started in a special mode.

First, the control should be connected to a PC via serial interface.

The following steps are neccessary:

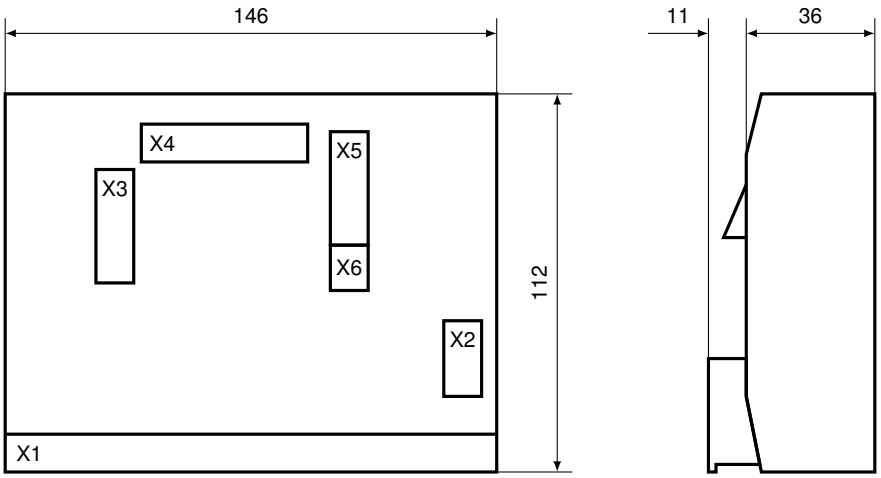
- Switch control off.
- Connect the two pins next to the RS232-Interface with a thin conductive object (e.g. a small screwdriver).
- Switch on control when the pins are connected. The LEDs on the front must not light.
- Now start the operating system upload.

6.3 Mechanical Installation

6.3.1 Attaching the Base Plate

Attaching with snap-on housing Phoenix for top-hat rail.

The exact dimensions can be seen in the figure below:



6.3.2 Clamp Contacts

Ferrules must not be used when connecting the individual wires to the spring terminals of the PLVC.

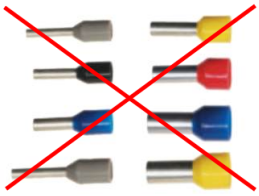


Figure 6.1.: No Ferrules for Connecting Individual Wires!

Due to the design the best tensile strength is achieved by pinching the stripped end into the terminal.

In contrast to a ferrule, the bare wire is bent over in the terminal.

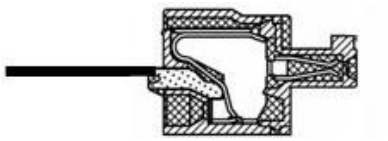


Figure 6.2.: Section of a Clamp

Pull on the wire to check the connection strength.

The following figure showes the individual steps of clamping.

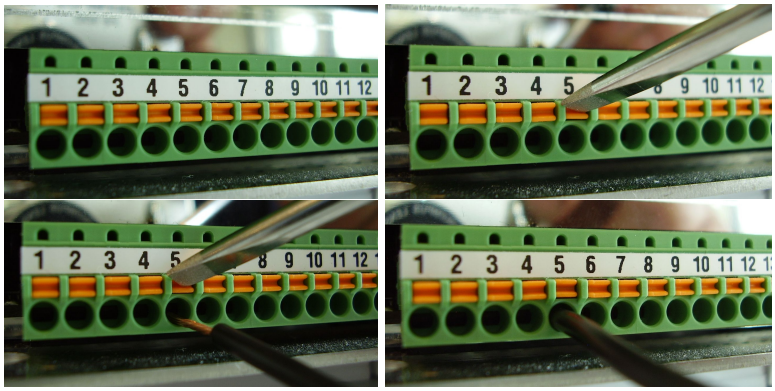


Figure 6.3.: Individual Steps of Clamping

6.4 PLVC21 LEDs Status Indication

- Two independent LEDs.
- Three speeds: periods 0,5 sec, 1 sec, 2 secs.
- With two ratios: short on, long off, and long on short off.

⟵2 seconds⟶	
LED1 (System)	
always off	
▬▬▬▬▬▬▬▬▬▬▬▬	Emergency stop
slow 2 second period:	
▔▔▔▔▔▔▔▔▬▬▬▬▬▬▬▬▬▬▬▬	Emergency radio
▬▬▬▬▬▬▬▬▬▬▬▬▔▔▔▔▔▔▔▔	PLC internal error
medium 1 second period:	
▔▔▔▬▬▬▬▬▬▔▔▔▬▬▬▬▬▬▬▬	Digital output
▔▔▔▔▔▔▔▔▔▔▔▬▬▬▔▔▔▔▔▔▔▔▔▔▬▬▬	Analog input
fast 0.5 second period:	
▔▬▬▬▬▬▔▬▬▬▬▬▔▬▬▬▬▬▔▬▬▬▬▬	Prop valve open
▔▔▔▔▔▔▬▔▔▔▔▔▔▬▔▔▔▔▔▔▬▔▔▔▔▔▔▬	Prop valve shortcut
always on	
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬	System ok
LED2	
always off	
▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔	Not used
slow 2 second period:	
▔▔▔▔▔▔▔▔▬▬▬▬▬▬▬▬▬▬▬▬	CAN bus off
▬▬▬▬▬▬▬▬▬▬▬▬▔▔▔▔▔▔▔▔	CAN warning
medium 1 second period:	
▔▔▔▬▬▬▬▬▬▔▔▔▬▬▬▬▬▬▬▬	Error EEprom
▔▔▔▔▔▔▔▔▔▔▔▬▬▬▔▔▔▔▔▔▔▔▔▔▬▬▬	Wrong supply voltage
fast 0.5 second period:	
▔▬▬▬▬▬▔▬▬▬▬▬▔▬▬▬▬▬▔▬▬▬▬▬	Error digital input
▔▔▔▔▔▔▬▔▔▔▔▔▔▬▔▔▔▔▔▔▬▔▔▔▔▔▔▬	No radio signal
always on	
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬	CAN ok (and no other errors for LED2)

Table 6.5.: PLVC21 LED Error Codes

All alarms are written in descending priority, i.e. if all is ok, both LEDs are on.

6.5 Pin Description Lists

6.5.1 PLVC21 Basic Device

Clamp	SPS	Connection data	Name	Note	Customer assign-ment
X1 1		GND	PGND		GND
2	0	Coil 12 / 24VDC, max. 2ADC	Coil A proportional valve 0	Also s/w- valve	
3	1	Coil 12 / 24VDC, max. 2ADC	Coil B proportional valve 1	Also s/w- valve	
4		PGND			
5	2	Coil 12 / 24VDC, max. 2ADC	Coil A proportional valve 2	Also s/w- valve	
6	3	Coil 12 / 24VDC, max. 2ADC	Coil B proportional valve 3	Also s/w- valve	
7		PGND			
8	%IW24.0	0 ... 5VDC, 0 ... 10VDC, 4 ... 20mA	Analog input 0		
9	%IW26.0	0 ... 5VDC, 0 ... 10VDC, 4 ... 20mA	Analog input 1		
10	%IW28.0	0 ... 5VDC, 0 ... 10VDC, 4 ... 20mA	Analog input 2		
11	%IW30.0	0 ... 5VDC, 0 ... 10VDC, 4 ... 20mA	Analog input 3		
12		PGND			GND
13	%QB0.0	Transistor output 1A, 10VDC ... 30VDC	Digital output 0		
14	%QB0.1	Transistor output 1A, 10VDC ... 30VDC	Digital output 1		
15	%QB0.2	Transistor output 1A, 10VDC ... 30VDC	Digital output 2		
16	%QB0.3	Transistor output 1A, 10VDC ... 30VDC	Digital output 3		
17	%QB0.4	Transistor output 1A, 10VDC ... 30VDC	Digital output 4		
18	%QB0.5	Transistor output 1A, 10VDC ... 30VDC	Digital output 5		
19	%QB0.6	Transistor output 1A, 10VDC ... 30VDC	Digital output 6		
20	%QB0.7	Transistor output 1A, 10VDC ... 30VDC	Digital output 7		
21		**PGND**			**GND**
22		**10VDC ... 30V DC max. 10A**	Power supply		**Ubat**
23	%IB3.7	10VDC ... 30VDC	Emergency stop input		Emergency stop
24	%IB0.4	10VDC ... 30VDC	Digital input IB0.4		
25	%IB0.6	10VDC ... 30VDC	Digital input IB0.6		
26	%IB0.5	10VDC ... 30VDC	Digital input IB0.5		
27	%IB0.3	10VDC ... 30VDC	Digital input IB0.3		
28		PGND			GND
29	%IB0.7	10VDC ... 30VDC	Digital input IB0.7		
30	%IB0.0	10VDC ... 30VDC, Frequency input 0 5kHz	Digital input IB0.0		
31	%IB0.1	10VDC ... 30VDC, Frequency input 1 5kHz	Digital input IB0.1		
32	%IB0.2	10VDC ... 30VDC, Frequency input 2 5kHz	Digital input IB0.2		
33		PGND			GND
34	RxD	RS232 Data Cable			
35	TxD				
36	PGND				

Table 6.6.: Pin Description List PLVC21 Basic Device

6.5.2 PLVC21 Extension

Clamp		SPS	Connection data	Name	Note	Customer assignment
X3	1	%IW40.0/%IB1.0	0 .. 10VDC, 4 .. 20mA, 0 .. 30VDC	Analog/Digital Input 8/IB1.0		
	2	%IW42.0/%IB1.1	0 .. 10VDC, 4 .. 20mA, 0 .. 30VDC	Analog/Digital Input 9/IB1.1		
	3	%IW44.0/%IB1.2	0 .. 10VDC, 4 .. 20mA, 0 .. 30VDC	Analog/Digital Input 10/IB1.2		
	4	%IW46.0/%IB1.3	0 .. 10VDC, 4 .. 20mA, 0 .. 30VDC	Analog/Digital Input 11/IB1.3		
	5		PGND			GND
	6	%IW48.0/%IB1.4	0 .. 10VDC, 4 .. 20mA, 0 .. 30VDC	Analog/Digital Input 12/IB1.4		
	7	%IW50.0/%IB1.5	0 .. 10VDC, 4 .. 20mA, 0 .. 30VDC	Analog/Digital Input 13/IB1.5		
	8	%IW52.0/%IB1.6	0 .. 10VDC, 4 .. 20mA, 0 .. 30VDC	Analog/Digital Input 14/IB1.6		
	9	%IW54.0/%IB1.7	0 .. 10VDC, 4 .. 20mA, 0 .. 30VDC	Analog/Digital Input 15/IB1.7		
	10		PGND			GND
X4	1			n. c. , deactivates X5.1		
	2	%QB1.0	Relay output 2A, 10VDC ... 30VDC	n. o. , deactivates X5.1	optional	
	3			com.		
	4			n. c. , deactivates X5.2		
	5	%QB1.1	Relay output 2A, 10VDC ... 30VDC	n. o. , deactivates X5.2	optional	
	6			com.		
	7			n. c. , deactivates X5.3		
	8	%QB1.2	Relay output 2A, 10VDC ... 30VDC	n. o. , deactivates X5.3	optional	
	9			com.		
	10			n. c. , deactivates X5.4		
	11	%QB1.3	Relay output 2A, 10VDC ... 30VDC	n. o. , deactivates X5.4	optional	
	12			com.		
X5	1	%QB1.0	Transistor output 1A, 10VDC ... 30VDC	Digital output 8		
	2	%QB1.1	Transistor output 1A, 10VDC ... 30VDC	Digital output 9		
	3	%QB1.2	Transistor output 1A, 10VDC ... 30VDC	Digital output 10		
	4	%QB1.3	Transistor output 1A, 10VDC ... 30VDC	Digital output 11		
	5	%QB1.4	Transistor output 1A, 10VDC ... 30VDC	Digital output 12		
	6	%QB1.5	Transistor output 1A, 10VDC ... 30VDC	Digital output 13		
	7	%QB1.6	Transistor output 1A, 10VDC ... 30VDC	Digital output 14		
	8	%QB1.7	Transistor output 1A, 10VDC ... 30VDC	Digital output 15		
	9	**Ubat**	**10VDC ... 30V DC max. 10A**	Positive power supply		Ubat
	10	**PGND**	**PGND**			PGND
X6	1		CAN High	CAN-Interface		
	2		CAN Low			
	3		Termination CAN-Bus		Connection to X6.2, if termination needed	

Figure 6.4.: Pin Description List PLVC21 Extension

7 PLVC41

7.1 PLVC41 Advantages and Differences

7.1.1 General Information

The PLVC41 is the successor of the PLVC4. It is compatible to the PLVC4 concerning OpenPCS code, parameters, and possible connections to extension modules, so you can use all programs and parameter files from your existing projects. The PLVC41 has got a faster processor (tripled speed) and more memory (tripled size).

7.1.2 Hardware

New plug

On the back of the PLVC41 you can connect to relay or proportional outputs (version PLVC41_4). The connector had been changed from blade terminal to a molex plug with crimped contacts. See the attached pin description for details (plug X31).

Additional digital input

At pin X31.1 you have the ability to connect an additional digital input signal.

Additional pins for voltage supply

Plug X31 also gives the possibility to connect supply-voltage instead of connecting to plug X301. This can save installation space. The plugs of X31 and X301 are internally connected.

EE_Save functionality

One special version of the PLVC4 basic device was the so called EE_Save version. Power can be supplied to either X31 or X301.

This functionality is now available as a standard. You have to wire pin X31.11 with a permanent power for this functionality. After PLVC41 detects removed power at normal power supply, parameters can be safed, and device switches off supply from pin X31.11 after 2 sec.

Second serial port type RS232

PLVC41 offers a second serial port, which can be used to connect to displays etc.

7.2 Technical Data

7.2.1 General Data

Enclosure	IP 20 to IEC 60529
Temperature range	$-40°C$ to $+80°C$
Supply voltage	10VDC to 30VDC
Max. total electricity	Basic module: 8A IPWM, PWM: 10A POW: 5A
Required external fuse	Basic module: 8A, slow-blow IPWM, PWM: 10A, slow-blow POW: 5A, slow-blow
Protection	Against reverse polarity Against load dump (DIN 40839) Vibration resistance (vibration: IEC 68-2-6, shock: IEC 68-2-27) EMV (EN 50081-1, EN 50081-2, EN 58082-1, EN 58082-2)
Lifetime relay	PLVC41 basic device - Resistive load 100000 operations at 5A/30VDC, 300000 operations at 2A/30VDC - Inductive load 100000 operations at 2A/30VDC, 300000 operations at 1A/30VDC Extension module POW 100000 operations at 20A/14VDC
Monitoring	Short-circuit Undervoltage, overvoltage Cable break
Cables connectors	- Inputs and outputs: using spring-cage connection phoenix FK-MCP, framework 3,5mm max. 8A, - Power supply: flat tabs 6,3mm

Continued on the next page. . .

. . . Continued from previous Page

	- Relay outputs: flat tabs 2,8mm
Microcontrollers	ST10F276, 16bit
Basic parameter memory	EEPROM 1000 words
Memory	Flash: 830kByte RAM: 188 kByte
Accessories	Software CAN bus power relays (see D 7845 Z) CAN bus nodes CAN HMI (see D 7845 HMI)
Fixing	6 x M3
Housing material	Stainless steel
Ground (weight)	Ca. 0,5kg (basic module) Ca. 0,25kg (extension module)

7.2.2 Base Unit PLVC41-G

Features of the Basic Devices PLVC41-G / PLVC41_4

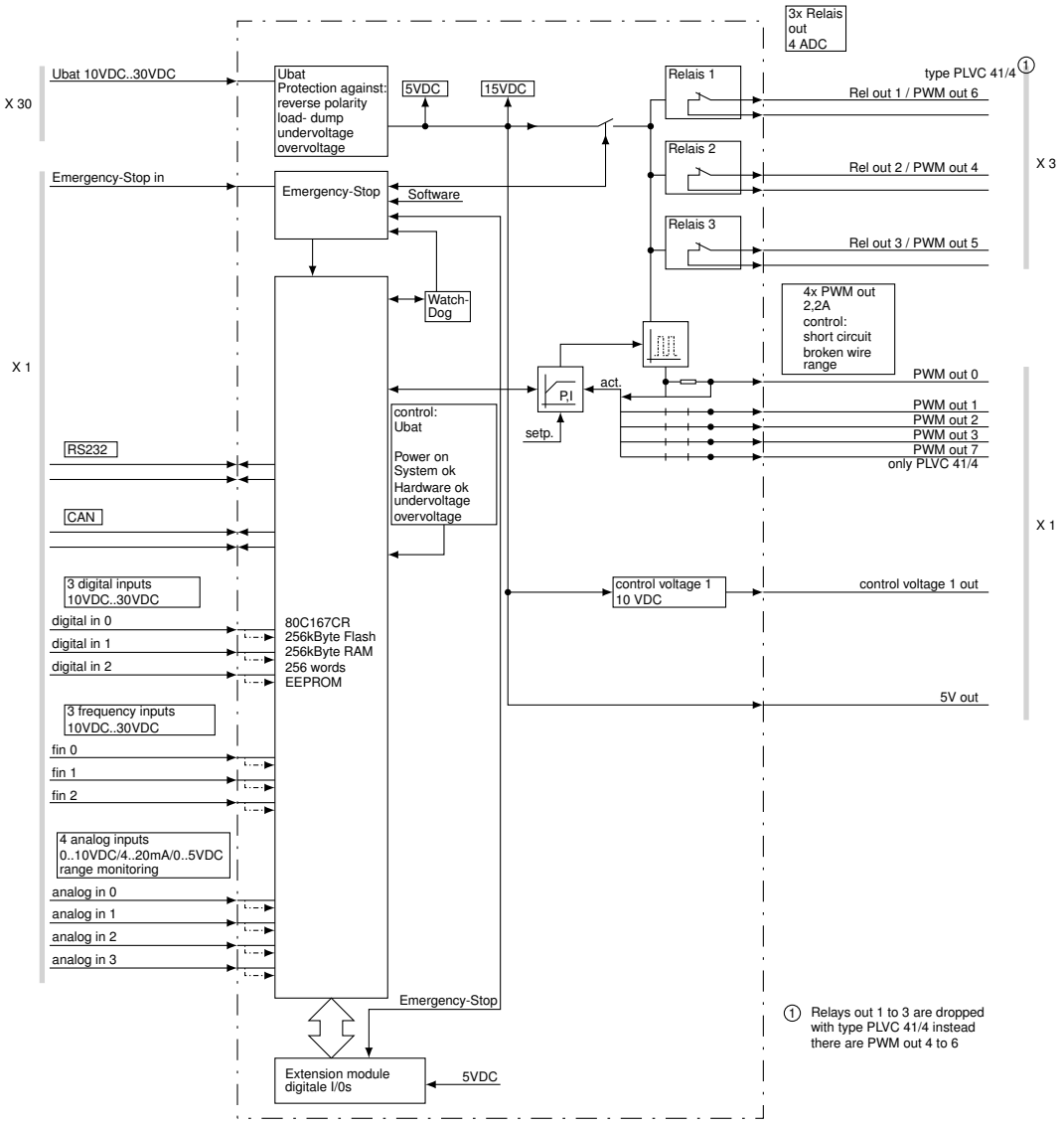
- 4 analog inputs (for joystick, potentiometer, sensors, e.g. analog pressure sensors)
- 6 digital inputs/frequency inputs (for limit switches, pressure switches, buttons, encoder, tachometer, incremental encoder, etc.)
- Emergency-stop-input (opto-decoupled)
- Interface for RS232 and CAN bus
- 4 outputs for proportional or b/w-valves (current regulated), 8 outputs at PLVC41_4
- 1 output 0-10VDC, max. 10mA
- 1 auxiliary voltage output 5VDC (voltage monitored), max. 200mA (for the supply of joysticks, potentiometers, etc.)
- 3 relay outputs (make contact) max. 5A, omitted for PLVC41_4
- Power supply 10-30VDC

Performance of the Connections

Terminalblock	Function	Description	Parameters
X301	Power supply	Nominal voltage U_N	10-30VDC
		max. total current	5A
X11	Digital inputs 0-2	Voltage range	10-30VDC/5kΩ
		Debounce for rising/falling signal edge can be switched separately	
	Analog inputs 0-3 (for joysticks, potentiometers, sensors etc.) Area monitoring	10bit ADC, 1024 steps	4-20mA 0-10VDC (default) 0-5VDC
	Frequency inputs 0-2	Cutoff frequency	$f_{Grenz} = 5kHz$
	Auxiliary voltage	For sensor, potentiometer	5VDC/200mA
	Voltage output	As a control signal	0-10VDC/10mA
	Emergency-stop-input	Opto-decoupled	
	PLVC41_E	I_{min}	100-1200mA
	Proportional or b/w-outputs 0-3	I_{max}	100-2200mA
	PLVC41_4	dither frequency	25-200Hz
	Proportional or b/w-outputs 0-7	Dither amplitude (based on PWM)	0-50%
	(each with low-side measurement)	cold resistance	2-35Ω
X31	Relay outputs 1, 2, 3 (omitted in PLVC41_4)	Voltage	1030VDC/4A
X11	Interface CAN bus		max. 500 kBd
	Interface RS232	Interface parameter	19,2kBd

Table 7.2.: Performance of the Connections

Block Diagram Basic Device PLVC41



A more detailed pin assignment is shown in graphic 7.8.

7.2.3 Extension Modules

General Properties

Supply voltage	10-30VDC
Max. total current	POW: 5A IPWM, PWM: 10A
Necessary external security	5A or 10A, slow-blow
All other data	See General Data (7.2.1)
Fixing	By four screws on the base module

Table 7.3.: General Properties of the PLVC Extensions

A total of three extensions can be fitted to the basic system, but a maximum of two pieces per each type of expansion module.

Exceptions:

- POW - can be installed only once
- The basic module PLVC41_4 can not be combined with two expansion modules IPWM.

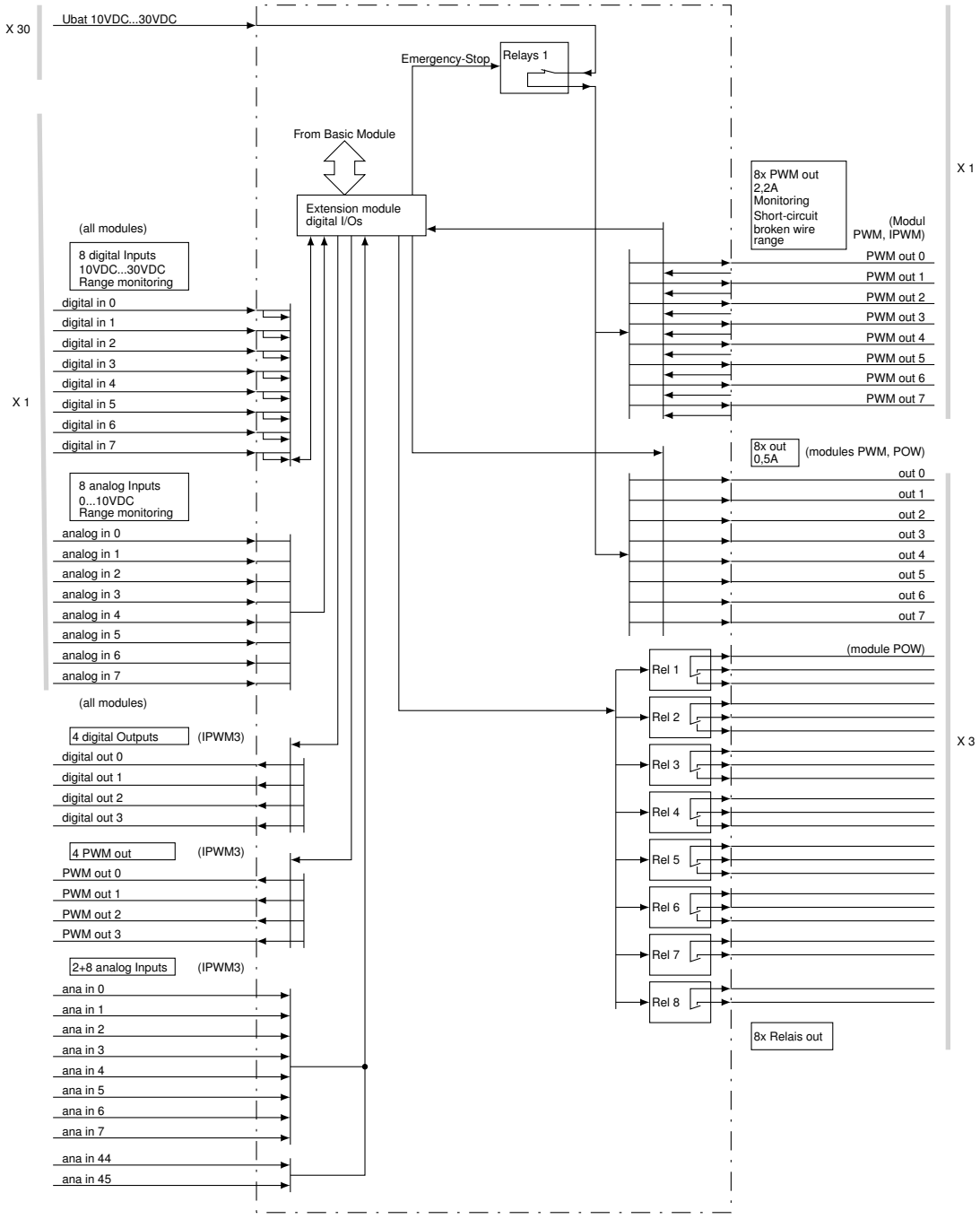
Features of the Expansion Modules IPWM, PWM and POW

- Extension module IPWM (IPWM1 and IPWM2)
  - 8 analog inputs (for joystick, potentiometers, sensors such as e.g. analog pressure sensors)
  - 8 digital inputs (for limit switches, pressure switches, buttons etc.)
  - 8 outputs for proportional or s/w-valves (current regulated)
  - Power supply 10-30VDC, max. 10A
- Extension module IPWM3
  - 18 analog inputs (for joystick, potentiometers, sensors such as e.g. analog pressure sensors)
  - 12 outputs for proportional or s/w-valves
  - 8 digital inputs (for limit switches, pressure switches, buttons etc.)
  - 4 digital outputs
  - Power supply 10-30VDC, max. 10A
- Extension module PWM

  - 8 analog inputs (for joystick, potentiometers, sensors such as e.g. analog pressure sensors)
  - 8 digital inputs (for limit switches, pressure switches, buttons etc.)
  - 8 PWM outputs for proportional or s/w-valves
  - 8 outputs for radiance or LED, max. 500mA (switch to ground)
  - Power supply 10-30VDC, max. 10A

- Extension module POW
  - 8 analog inputs (for joystick, potentiometers, sensors such as e.g. analog pressure sensors)
  - 8 digital inputs (for limit switches, pressure switches, buttons etc.)
  - 8 relay outputs (6x changeover, 2x normally open), max. 15A
  - 8 outputs for lights or LED, max. 500mA (switch to ground)
  - Power supply 10-30VDC, max. 5A

Block Diagram of Expansion Modules



More detailed pin assignments are shown in graphics, beginning with 7.9.

Performance of the Connections of the Extensions

Terminal block	Function	Description	Parameter	PWM	IPWM	POW
X301	Power supply	Rated voltage U_N	10-30VDC	•	•	•
		max. total current	5A			•
		max. total current	10A	•	•	
X1	Digital inputs 0-7	Voltage range	10-30VDC / 5kΩ	•	•	•
	Analog inputs 0-7 with area monitoring	10bit ADC, 1024 steps	4-20mA	•	•	•
			0-10VDC (default)			
			0-5VDC			
	proportional or s/w-outputs 0-7	I_{min}	100-1200mA	•	•	
	IPWM: with low-side measurement	I_{max}	100-2200mA			
	PWM: without low-side measurement (PWM out 0-7)	Ditherfrequenz	25-200Hz			
		Dither amplitude (related to PWM)	0-50%			
		Cold resistance	2-35Ω			
	Digital outputs 0-7 (out 0-7) (switch to ground)	Imax	100mA	•		•
X31	Relay outputs 1-8	I_{max}	15A			•

Table 7.4.: Performance of the Connections

7.3 Download an Operating System

The operating system can be updated using a Windows PC (or Windows laptop).

7.3.1 With an Intact Operating System

A new operating system is easy to install via an already existing operating system. All functionality for an update is already included in the existing operating system. Connect the PLVC control via serial interface to the PC and start the appropriate upload program of the operating system.

7.3.2 Damaged Operating System

If you cannot activate your current operating system (e.g. by an aborted operating system update), a new operating system can be installed though.

For this, the PLVC must be started in the BSL mode.

First, the control should be connected to a PC via serial interface.

The following steps are neccessary:

- Switch control off.
- Connect the two pins at the right hand side of the terminal block of the case (figure 7.1) with a thin conductive object (e.g. a small screwdriver).
- Switch on control when the pins are connected. The LEDs on the front must not light.
- Now start the operating system upload.

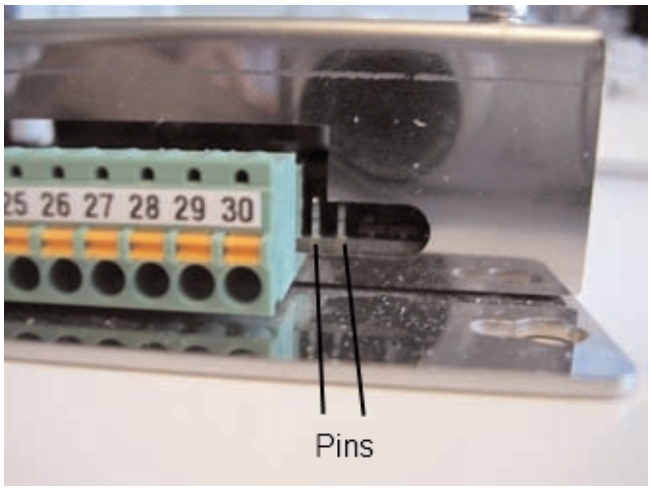


Figure 7.1.: Pins at the right hand side of the terminal block of PLVC41

7.4 Mechanical Installation

7.4.1 Attaching the Base Plate

The base plate can be attached in the cabinet with 6 M3 screws.

The exact dimensions can be seen in the figure below:

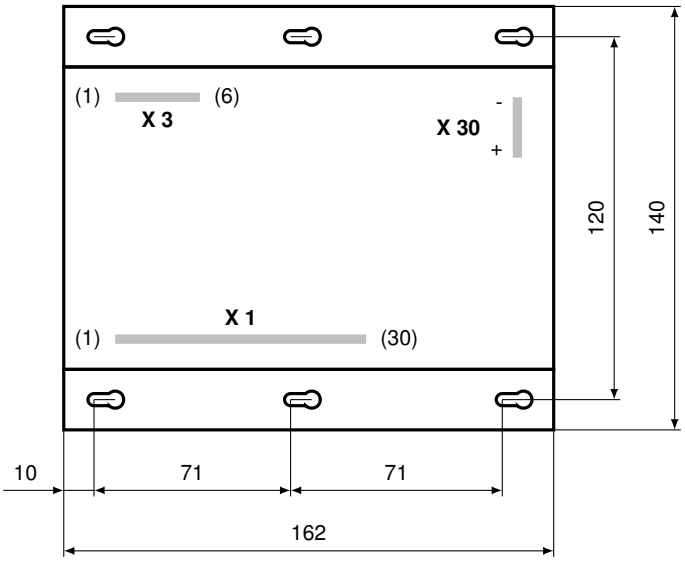


Figure 7.2.: Dimensions of the PLVC41 Housing Base Plate

Instructions for potential separation are given in chapter Electronic Installation (4).

7.4.2 Power Supply

In general, the voltage supply takes place via a Molex connector, which needs to be fixed to the terminal block X.31.10/12.

Alternatively, the device can be supplied with voltage via two flat connectors. For connection two isolated flat receptacles with 6,3 mm width will be needed. These need to bis fixed to the terminal block X301.10/12.

Figure 7.3.: Flat Receptacles for Power Supply

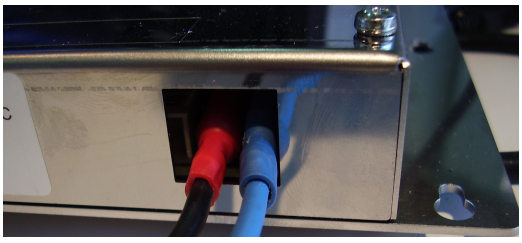


Figure 7.4.: Power Supply PLVC41 Base Device

Attention:

The internal connection between X31.10/12 and X301.1/2 must not be used for transmitting voltage.

7.4.3 Clamp Contacts

Ferrules must not be used when connecting the individual wires to the spring terminals of the PLVC.

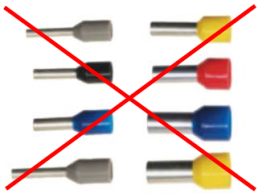


Figure 7.5.: No Ferrules for Connecting Individual Wires!

Due to the design the best tensile strength is achieved by pinching the stripped end into the terminal.

In contrast to a ferrule, the bare wire is bent over in the terminal.

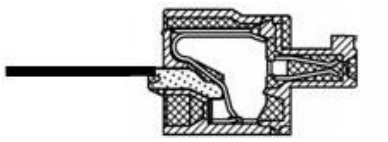


Figure 7.6.: Section of a Clamp

Pull on the wire to check the connection strength.

The following figures show the individual steps of clamping.

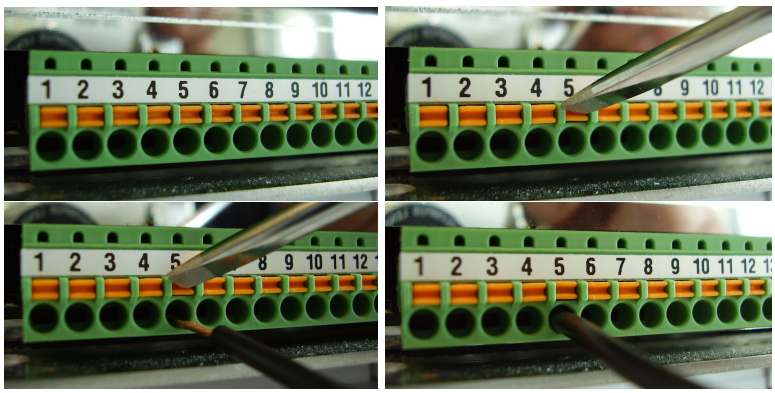


Figure 7.7.: Individual Steps of Clamping

7.5 Versions

7.5.1 Example Configurations

PLVC41-G/VVVV	Basic module PLVC41, four analog inputs set to 0-10V
-OS/EN	Standard operating system in English
PLVC41_4-G/VVAA	Basic module PLVC41_4, two analog inputs set to 0-10V, two analog inputs set to 4-20mA
-OS/EN	Standard operating system in English
PLVC41-X/VVVV	Basic module PLVC41, four analog inputs set to 0-10V
-IPWM/VVVVJJAA	Expansion module IPWM, four analog inputs set to 0-10V, two analog inputs set to 0-5V and two analog inputs set to 4-20mA
-OS/EN	Standard operating system in English

Table 7.5.: Example Configurations

Notes for the Basic Modules

PLVC4_4-G and PLVC41_4-G

At the PLVC41_4-G the three relays are omitted. As compensation for this one gets four additional proportional outputs. Nevertheless the amount of measuring inputs remain at four. Therefore it is recommended to use the eight proportional as twin coils. Otherwise the current measurement must be switched off via parameters.

Configuring Analog Inputs

As standard, all analog inputs are 0-10VDC. Other assignments (4-20mA = A or 0-5VDC = J) can be specified in the type code.

In principle, any input can be set separately. But it is recommended to make the configuration change “from the top down”, that means starting with the last extension module.

Expansion modules

A total of three extensions can be fitted to the basic system, where a maximum of two pieces of any type of expansion module can be used.

There are max. 16 current-controlled outputs (basic module and expansion module IPWM) available.

Exceptions:

- POW - can be installed only once

- The basic module PLVC41_4 can not be combined with two expansion modules IPWM.

7.5.2 Possible Options

Basic modules	PLVC41-G PLVC41_4-G, 4 additional proportional outputs (3 relays are omitted)
Extension modules	IPWM PWM POW
Operating system	EN, English

Table 7.6.: Possible Options

7.6 Dimensions

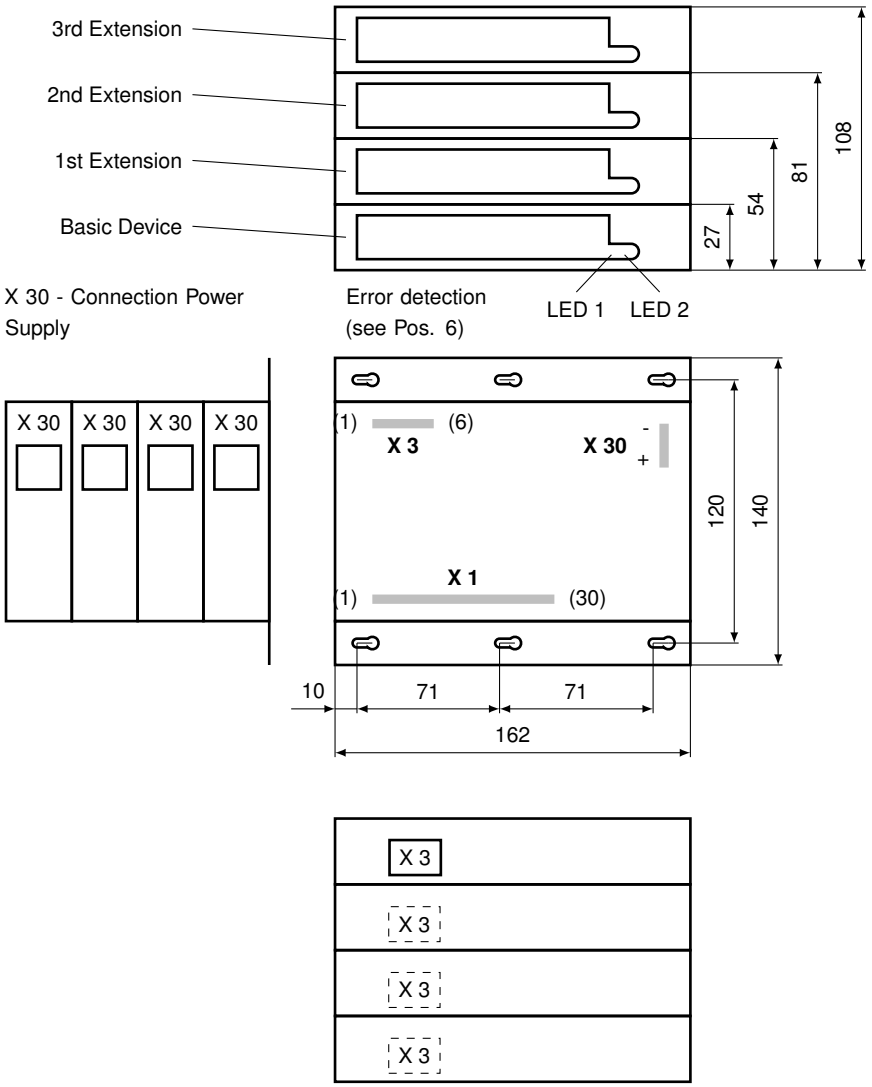


Figure 7.8.: Dimensions PLVC41

7.7 PLVC41 LEDs Status Indication

- Two independent LEDs.
- Three speeds: periods 0,5 sec, 1 sec, 2 secs.
- With two ratios: short on, long off, and long on short off.

⟵2 seconds⟶	
LED1 (System)	
always off	
▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁	Emergency stop
slow 2 second period:	
▔▔▔▔▔▔▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁	Emergency radio
▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▁▁▁▁▁▁▁	PLC internal error
medium 1 second period:	
▔▔▁▁▁▁▁▁▁▁▁▁▔▔▁▁▁▁▁▁▁▁▁▁▁▁	Digital output
▔▔▔▔▔▔▔▔▔▔▔▔▁▁▁▔▔▔▔▔▔▔▔▔▔▔▁▁▁	Analog input
fast 0.5 second period:	
▔▁▁▁▁▁▔▁▁▁▁▁▔▁▁▁▁▁▔▁▁▁▁▁	Prop valve open
▔▔▔▔▔▁▔▔▔▔▔▁▔▔▔▔▔▁▔▔▔▔▔▁	Prop valve shortcut
always on	
▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔	System ok
LED2	
always off	
▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔	Not used
slow 2 second period:	
▔▔▔▔▔▔▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁	CAN bus off
▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▁▁▁▁▁▁▁	CAN warning
medium 1 second period:	
▔▔▁▁▁▁▁▁▁▁▁▁▔▔▁▁▁▁▁▁▁▁▁▁▁▁	Error EEprom
▔▔▔▔▔▔▔▔▔▔▔▔▁▁▁▔▔▔▔▔▔▔▔▔▔▔▁▁▁	Wrong supply voltage
fast 0.5 second period:	
▔▁▁▁▁▁▔▁▁▁▁▁▔▁▁▁▁▁▔▁▁▁▁▁	Error digital input
▔▔▔▔▔▁▔▔▔▔▔▁▔▔▔▔▔▁▔▔▔▔▔▁	No radio signal
always on	
▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔▔	CAN ok (and no other errors for LED2)

Table 7.7.: PLVC41 LED Error Codes

All alarms are written in descending priority, i.e. if all is ok, both LEDs are on.

7.8 Pin Description Lists

7.8.1 PLVC41 Basic Device

Pin	PLC	Alternative	Connectiondata	Name	Note	User
	X301					
1			10...30VDC / 8A	PGND		GND
2			10...30VDC / 8A	Power supply		U_BAT
	X11					
1			RxD	RS232 - Data cable		Sub-D Pin 3
2			TxD			Sub-D Pin 2
3			PGND			SUB-D Pin 5
4	0		Coil 0 / 24VDC, max. 2ADC	Coil A proportional Valve 0	Also on/off Valve	
5	1		Coil 1 / 24VDC, max. 2ADC	Coil B proportional Valve 1	Also on/off Valve	
6			measurement input			GND of valve 0 (and valve 1[*2])
7			measurement input			GND of valve 1 (or valve 4 and 5[*3])
8	2		Coil 2 / 24VDC, max. 2ADC	Coil B proportional Valve 2	Also on/off Valve	
9	3		Coil 3 / 24VDC, max. 2ADC	Coil B proportional Valve 3	Also on/off Valve	
10			measurement input			GND of valve 2 (and valve3 [*2])
11			measurement input			GND of valve 3 (or valve 6 and 7 [*3])
12	%IB3.4		10VDC...30VDC	Digital input IB3.4		
13	%IB3.6		10VDC...30VDC	Digital input IB3.6		
14	%IB3.5		10VDC...30VDC	Digital input IB2.5		
15			PGND			
16			10VDC...30VDC	Emergency stop input		Emergency stop
17			5V / 200mA output	Sensor supply		
18			0...10V / 100mA output	programmable		
19	%IB3.0	Freq.0	10VDC...30VDC / max 5kHz	Digital input IB3.0	Also frequency input 0	
20	%IB3.1	Freq.1	10VDC...30VDC / max 5kHz	Digital input IB3.1	Also frequency input 1	
21	%IB3.2	Freq.2	10VDC...30VDC / max 5kHz	Digital input IB3.2	Also frequency input 2	
22		(7[*2])	PGND (or Coil 12/24VDC max. 2A[*2])	(Coil B proportional Valve 7[*2])		
23	%IW104.0		0...5 / 0...10VDC / 4...20mA[*1]	Analog input 40		
24	%IW106.0		0...5 / 0...10VDC / 4...20mA[*1]	Analog input 41		
25	%IW108.0		0...5 / 0...10VDC / 4...20mA[*1]	Analog input 42		
26	%IW110.0		0...5 / 0...10VDC / 4...20mA[*1]	Analog input 43		
27			PGND			
28			50, 100, 125, 250, 500, 100kB	CAN1_H	CAN Bus	CAN High
29			50, 100, 125, 250, 500, 100kB	CAN1_l	CAN Bus	CAN Low
30			Termination CAN	120 Ohm to CAN_H intern		Connect to pin 29[*4]
	X31					
1	%QB2.0	6 if no IPWM3	Com (or GND[*2])	(GND[*2])	qb0.4***	
2		else 20***	Relay output 5A, 30VDC or	(Coil A proportional valve 6[*2])		
3	%QB2.1	4 if no IPWM3	Com (or GND[*2])	(GND[*2])	qb0.5***	
4		else 21***	Relay output 5A, 30VDC or	(Coil A proportional valve 4[*2])		
5	%QB2.2	5 if no IPWM3	Com (or GND[*2])	(GND[*2])	qb0.6***	
6		else 22***	Relay output 5A, 30VDC or	(Coil A proportional valve 5[*2])		
7			RxD_2			
8			TxD_2			
9	%IB3.3	Freq.3	10VDC...30VDC/max. 5kHz	Digital input IB3.3	Also frequency input 3	
10			10...30VDC/8A	PGND	intern connect to X301.1	GND can replace X301.1
11			EE_UP	perm. supply for EE_Save[*5]		
12			10...30VDC/8A	Power supply		U_Bat can replace X301.2

Table 7.8.: Pin Description List PLVC41-G / PLVC41-4-G

Description:

*1 default 0...10VDC. Otherwise as ordered

*2 if double coil

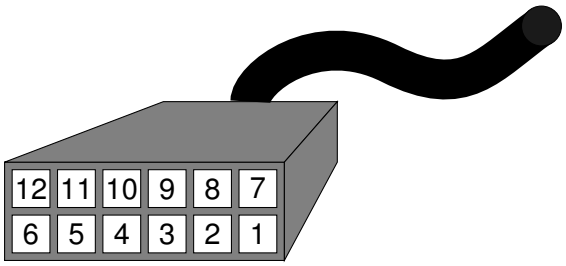
*3 if PLVC41-4

*4 if termination is used

*5 EE-Save = If wired with supply independent of main supply, device will switch off itself with a delay of 2sec. So Parameters can be saved.

*** if PWM-Mapping equals 8, the ouptuts can be used as PWM-outputs, PLVC41-4 must be configured as PLVC41 (marked yellow)

Front view MOLEX connector:



7.8.2 IPWM1 Extension

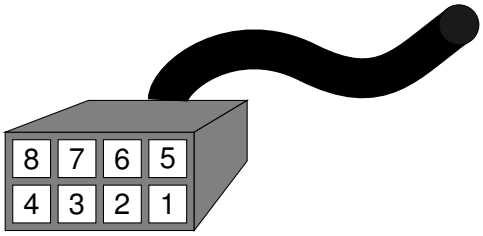
Pin	PLC	Alternative	Connection data	Name	Note	User
	X305					
1			10...30VDC / 16A	PGND		GND
2			10...30VDC / 16A	Power supply		U_BAT
	X15					
1	0		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 0	Also on/off Valve	
2	1		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 1	Also on/off Valve	
3			measurement input			GND of valve 0 and valve 1
4	2		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 2	Also on/off Valve	
5	3		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 3	Also on/off Valve	
6			measurement input			GND of valve 2 and valve 3
7	4		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 4	Also on/off Valve	
8	5		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 5	Also on/off Valve	
9			measurement input			GND of valve 4 and valve 5
10	6		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 6	Also on/off Valve	
11	7		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 7	Also on/off Valve	
12			measurement input			GND of valve 6 and valve 7
13	%IB1.0		10VDC...30VDC	Digital input IB1.0		
14	%IB1.1		10VDC...30VDC	Digital input IB1.1		
15	%IB1.2		10VDC...30VDC	Digital input IB1.2		
16	%IB1.3		10VDC...30VDC	Digital input IB1.3		
17	%IB1.4		10VDC...30VDC	Digital input IB1.4		
18	%IB1.5		10VDC...30VDC	Digital input IB1.5		
19	%IB1.6		10VDC...30VDC	Digital input IB1.6		
20	%IB1.7		10VDC...30VDC	Digital input IB1.7		
21			GND			
22	%IW24.0		0...5/0...10VDC/4...20mA*1	Analog input 0		
23	%IW26.0		0...5/0...10VDC/4...20mA*1	Analog input 1		
24	%IW28.0		0...5/0...10VDC/4...20mA*1	Analog input 2		
25	%IW30.0		0...5/0...10VDC/4...20mA*1	Analog input 3		
26	%IW32.0		0...5/0...10VDC/4...20mA*1	Analog input 4		
27	%IW34.0		0...5/0...10VDC/4...20mA*1	Analog input 5		
28	%IW36.0		0...5/0...10VDC/4...20mA*1	Analog input 6		
29	%IW38.0		0...5/0...10VDC/4...20mA*1	Analog input 7		
30			GND			
	X35					
1						
2						
3						
4						
5			10...30VDC / 8A	PGND	intern connect X305.1	GND can replace X305.1
6			10...30VDC / 8A			
7			10...30VDC / 8A	Power supply	intern connect X305.2	U_BAT can replace X305.2
8			10...30VDC / 8A			

Table 7.9.: Pin Description List IPWM1 Extension

Description:

*1 default 0...10VDC. Otherwise as ordered.
Impedance: 0-10VDC (470$k\Omega$), 0-5VDC (94$k\Omega$), 4-20mA (220$k\Omega$)

Front view MOLEX connector:



7.8.3 IPWM2 Extension

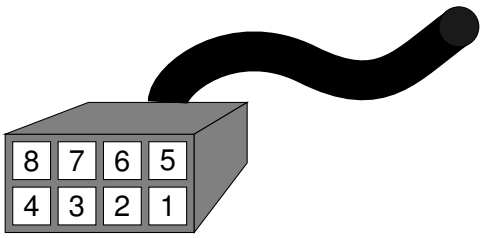
Pin	PLC	Alternative	Connection data	Name	Note	User
	X302					
1			10...30VDC / 16A	PGND		GND
2			10...30VDC / 16A	Power supply		U_BAT
	X12					
1	8		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 8	Also on/off Valve	
2	9		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 9	Also on/off Valve	
3			measurement input			GND of valve 8 and valve 9
4	10		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 10	Also on/off Valve	
5	11		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 11	Also on/off Valve	
6			measurement input			GND of valve 10 and valve 11
7	12		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 12	Also on/off Valve	
8	13		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 13	Also on/off Valve	
9			measurement input			GND of valve 12 and valve 13
10	14		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 14	Also on/off Valve	
11	15		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 15	Also on/off Valve	
12			measurement input			GND of valve 14 and valve 15
13	%IB0.0		10VDC...30VDC	Digital input IB0.0		
14	%IB0.1		10VDC...30VDC	Digital input IB0.1		
15	%IB0.2		10VDC...30VDC	Digital input IB0.2		
16	%IB0.3		10VDC...30VDC	Digital input IB0.3		
17	%IB0.4		10VDC...30VDC	Digital input IB0.4		
18	%IB0.5		10VDC...30VDC	Digital input IB0.5		
19	%IB0.6		10VDC...30VDC	Digital input IB0.6		
20	%IB0.7		10VDC...30VDC	Digital input IB0.7		
21			GND			
22	%IW40.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 8		
23	%IW42.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 9		
24	%IW44.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 10		
25	%IW46.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 11		
26	%IW48.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 12		
27	%IW50.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 13		
28	%IW52.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 14		
29	%IW54.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 15		
30			GND			
	X32					
1						
2						
3						
4						
5			10...30VDC / 8A	PGND	intern connect X302.1	GND can replace X302.1
6			10...30VDC / 8A			
7			10...30VDC / 8A	Power supply	intern connect X302.2	U_BAT can replace X302.2
8			10...30VDC / 8A			

Table 7.10.: Pin Description List IPWM2 Extension

Description:

^{*1} default 0...10VDC. Otherwise as ordered.
Impedance: 0-10VDC (470$k\Omega$), 0-5VDC (94$k\Omega$), 4-20mA (220$k\Omega$)

Front view MOLEX connector:



7.8.4 IPWM3 Extension

Pin	PLC	Alternative	Connection data	Name	Note	User
	X302					
1			10...30VDC / 16A	PGND		GND
2			10...30VDC / 16A	Power supply		U_BAT
	X12					
1	8		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 8	Also on/off Valve	
2	9		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 9	Also on/off Valve	
3			measurement input			GND of valve 8 and valve 9
4	10		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 10	Also on/off Valve	
5	11		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 11	Also on/off Valve	
6			measurement input			GND of valve 10 and valve 11
7	12		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 12	Also on/off Valve	
8	13		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 13	Also on/off Valve	
9			measurement input			GND of valve 12 and valve 13
10	14		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 14	Also on/off Valve	
11	15		12VDC / 24VDC, max. 2ADC	Coil B proportional Valve 15	Also on/off Valve	
12			measurement input			GND of valve 14 and valve 15
13	%IB0.0		10VDC...30VDC	Digital input IB0.0		
14	%IB0.1		10VDC...30VDC	Digital input IB0.1		
15	%IB0.2		10VDC...30VDC	Digital input IB0.2		
16	%IB0.3		10VDC...30VDC	Digital input IB0.3		
17	%IB0.4		10VDC...30VDC	Digital input IB0.4		
18	%IB0.5		10VDC...30VDC	Digital input IB0.5		
19	%IB0.6		10VDC...30VDC	Digital input IB0.6		
20	%IB0.7		10VDC...30VDC	Digital input IB0.7		
21			GND			
22	%IW40.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 8		
23	%IW42.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 9		
24	%IW44.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 10		
25	%IW46.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 11		
26	%IW48.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 12		
27	%IW50.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 13		
28	%IW52.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 14		
29	%IW54.0		0...5/0...10VDC/4...20mA^{*1}	Analog input 15		
30			GND			
	X32					
1	%QB5.0	16	10...30VDC / 1A	Digital output QB5.0 (QB0.0^{*2})	also qb0.0	
2	%QB5.1	17	10...30VDC / 1A	Digital output QB5.1 (QB0.1^{*2})	also qb0.1	
3	%QB5.2	18	10...30VDC / 1A	Digital output QB5.2 (QB0.2^{*2})	also qb0.2	
4	%QB5.3	19***	10...30VDC / 1A	Digital output QB5.3 (QB0.3^{*2})	also qb0.3	
5	%IW24.0	%IB24.0	0...10VDC / also digital	Analog input 0	(not available^{*3})	
6	%IW26.0	%IB26.0	0...10VDC / also digital	Analog input 1	(not available^{*3})	
7	%IW28.0	%IB28.0	0...10VDC / also digital	Analog input 2	(not available^{*3})	
8	%IW30.0	%IB30.0	0...10VDC / also digital	Analog input 3	(not available^{*3})	
9	%IW32.0	%IB32.0	0...10VDC / also digital	Analog input 4	(not available^{*3})	
10	%IW34.0	%IB34.0	0...10VDC / also digital	Analog input 5	(not available^{*3})	
11	%IW36.0	%IB36.0	0...10VDC / also digital	Analog input 6	(not available^{*3})	
12	%IW38.0	%IB38.0	0...10VDC / also digital	Analog input 7	(not available^{*3})	
13	4		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 4	Also on/off Valve	
14	5		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 5	Also on/off Valve	
15			measurement input			GND of valve 4 and valve 5
16	6		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 6	Also on/off Valve	
17	7		12VDC / 24VDC, max. 2ADC	Coil A proportional Valve 7	Also on/off Valve	
18			measurement input			GND of valve 6 and valve 7
19	%IW112.0	%IB112.0	0...10VDC / also digital	Analog input 44		
20	%IW114.0	%IB114.0	0...10VDC / also digital	Analog input 45		
21 22			10...30VDC / 8A 10...30VDC / 8A	PGND	intern connect X302.1	GND can replace X302.1
23 24			10...30VDC / 8A 10...30VDC / 8A	Power supply	intern connect X302.2	U_BAT can replace X302.2

Table 7.11.: Pin Description List IPWM3 Extension

Description:

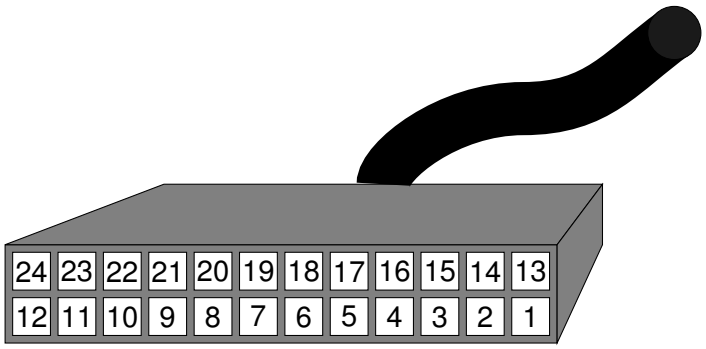
^{*1} default 0...10VDC. Otherwise as ordered.
Impedance: 0-10VDC (470$k\Omega$), 0-5VDC (94$k\Omega$), 4-20mA (220$k\Omega$)

^{*2} SPS adress change in combination with POW-Extension

^{*3} not available in combination with POW-Extension

*** if PWM-Mapping equals 1, the ouptuts can be used as PWM-outputs via qb0.0...0.3 (marked yellow)

Front view MOLEX connector:



7.8.5 PWM1 Extension

Pin	PLC	Alternative	Connectiondata	Name	Note	User
	X304					
1			10...30VDC / 16A	PGND		GND
2			10...30VDC / 16A	Power supply		U_BAT
	X14					
1	%QB0.0	Channel 16	10...30VDC / 2A	Digital / PWM output 0		
2	%QB0.1	Channel 17	10...30VDC / 2A	Digital / PWM output 1		
3	%QB0.2	Channel 18	10...30VDC / 2A	Digital / PWM output 2		
4	%QB0.3	Channel 19	10...30VDC / 2A	Digital / PWM output 3		
5			PGND			
6	%QB0.4	Channel 20	10...30VDC / 2A	Digital / PWM output 4		
7	%QB0.5	Channel 21	10...30VDC / 2A	Digital / PWM output 5		
8	%QB0.6	Channel 22	10...30VDC / 2A	Digital / PWM output 6		
9	%QB0.7	Channel 23	10...30VDC / 2A	Digital / PWM output 7		
10			PGND			
11	%IB0.0		10VDC ... 30VDC	Digital input IB 0.0		
12	%IB0.1		10VDC ... 30VDC	Digital input IB 0.1		
13	%IB0.2		10VDC ... 30VDC	Digital input IB 0.2		
14	%IB0.3		10VDC ... 30VDC	Digital input IB 0.3		
15			PGND			
16	%IB0.4		10VDC ... 30VDC	Digital input IB 0.4		
17	%IB0.5		10VDC ... 30VDC	Digital input IB 0.5		
18	%IB0.6		10VDC ... 30VDC	Digital input IB 0.6		
19	%IB0.7		10VDC ... 30VDC	Digital input IB 0.7		
20			GND			
21	%IW40.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 8		
22	%IW42.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 9		
23	%IW44.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 10		
24	%IW46.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 11		
25			PGND			
26	%IW48.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 12		
27	%IW50.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 13		
28	%IW52.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 14		
29	%IW54.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 15		
30			PGND			
	X34					
1	%QB6.0		10-30VDC 1A^{*2}	Digital output QB 6.0		
2	%QB6.1		10-30VDC 1A^{*2}	Digital output QB 6.1		
3	%QB6.2		10-30VDC 1A^{*2}	Digital output QB 6.2		
4	%QB6.3		10-30VDC 1A^{*2}	Digital output QB 6.3		
5	%QB6.4		10-30VDC 1A^{*2}	Digital output QB 6.4		
6	%QB6.5		10-30VDC 1A^{*2}	Digital output QB 6.5		
7	%QB6.6		10-30VDC 1A^{*2}	Digital output QB 6.6		
8	%QB6.7		10-30VDC 1A^{*2}	Digital output QB 6.7		
9			10...30VDC / 1A	PGND	intern connect X304.1	GND can replace X304.1
10			10...30VDC / 1A			
11			10...30VDC / 1A	Power supply	intern connect X304.2	U_BAT can replace X304.2
12			10...30VDC / 1A			

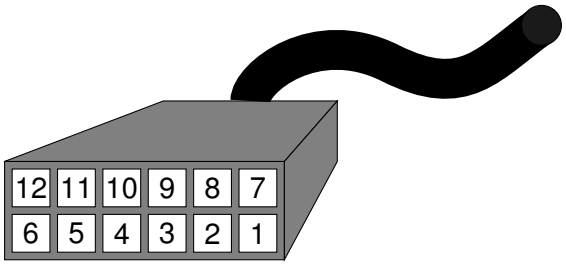
Table 7.12.: Pin Description List PLVC41-PWM1

Description:

^{*1} default 0...10VDC. Otherwise as ordered.
Impedance: 0-10VDC (470$k\Omega$), 0-5VDC (94$k\Omega$), 4-20mA (220$k\Omega$)

^{*2} Sum of QB6.0 to QB6.7 6A max.
Standard: Low-Side-Switch, optional: High-Side-Switch, built-in freewheeling diode

Front view MOLEX connector:


12
11
10
9
8
7
6
5
4
3
2
1

7.8.6 PWM2 Extension

Pin	PLC	Alternative	Connectiondata	Name	Note	User
X304						
1			10...30VDC / 16A	PGND		GND
2			10...30VDC / 16A	Power supply		U_BAT
X14						
1	%QB0.0	Channel 16	10...30VDC / 2A	Digital / PWM output 0		
2	%QB0.1	Channel 17	10...30VDC / 2A	Digital / PWM output 1		
3	%QB0.2	Channel 18	10...30VDC / 2A	Digital / PWM output 2		
4	%QB0.3	Channel 19	10...30VDC / 2A	Digital / PWM output 3		
5			PGND			
6	%QB0.4	Channel 20	10...30VDC / 2A	Digital / PWM output 4		
7	%QB0.5	Channel 21	10...30VDC / 2A	Digital / PWM output 5		
8	%QB0.6	Channel 22	10...30VDC / 2A	Digital / PWM output 6		
9	%QB0.7	Channel 23	10...30VDC / 2A	Digital / PWM output 7		
10			PGND			
11	%IB0.0		10VDC ... 30VDC	Digital input IB 0.0		
12	%IB0.1		10VDC ... 30VDC	Digital input IB 0.1		
13	%IB0.2		10VDC ... 30VDC	Digital input IB 0.2		
14	%IB0.3		10VDC ... 30VDC	Digital input IB 0.3		
15			PGND			
16	%IB0.4		10VDC ... 30VDC	Digital input IB 0.4		
17	%IB0.5		10VDC ... 30VDC	Digital input IB 0.5		
18	%IB0.6		10VDC ... 30VDC	Digital input IB 0.6		
19	%IB0.7		10VDC ... 30VDC	Digital input IB 0.7		
20			GND			
21	%IW40.0		0...5/0...10VDC / 4...20mA*1	Analog input 8		
22	%IW42.0		0...5/0...10VDC / 4...20mA*1	Analog input 9		
23	%IW44.0		0...5/0...10VDC / 4...20mA*1	Analog input 10		
24	%IW46.0		0...5/0...10VDC / 4...20mA*1	Analog input 11		
25			PGND			
26	%IW48.0		0...5/0...10VDC / 4...20mA*1	Analog input 12		
27	%IW50.0		0...5/0...10VDC / 4...20mA*1	Analog input 13		
28	%IW52.0		0...5/0...10VDC / 4...20mA*1	Analog input 14		
29	%IW54.0		0...5/0...10VDC / 4...20mA*1	Analog input 15		
30			PGND			
X34		if HS***				
1	%QB6.0	16	10-30VDC 1A*2	Digital output QB 6.0	also qb0.0	
2	%QB6.1	17	10-30VDC 1A*2	Digital output QB 6.1	also qb0.1	
3	%QB6.2	18	10-30VDC 1A*2	Digital output QB 6.2	also qb0.2	
4	%QB6.3	19	10-30VDC 1A*2	Digital output QB 6.3	also qb0.3	
5	%QB6.4	20	10-30VDC 1A*2	Digital output QB 6.4	also qb0.4	
6	%QB6.5	21	10-30VDC 1A*2	Digital output QB 6.5	also qb0.5	
7	%QB6.6	22	10-30VDC 1A*2	Digital output QB 6.6	also qb0.6	
8	%QB6.7	23	10-30VDC 1A*2	Digital output QB 6.7	also qb0.7	
9			10...30VDC / 1A	PGND	intern connect X304.1	GND can replace X304.1
10			10...30VDC / 1A			
11			10...30VDC / 1A	Power supply	intern connect X304.2	U_BAT can replace X304.2
12			10...30VDC / 1A			

Table 7.13.: Pin Description List PLVC41-PWM2

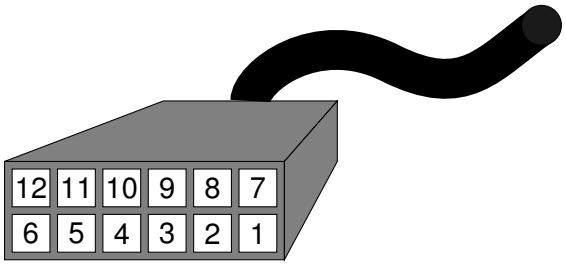
Description:

*1 default 0...10VDC. Otherwise as ordered.
Impedance: 0-10VDC (470$k\Omega$), 0-5VDC (94$k\Omega$), 4-20mA (220$k\Omega$)

*2 Sum of QB6.0 to QB6.7 6A max.
Standard: Low-Side-Switch, optional: High-Side-Switch, built-in freewheeling diode

*** if PWM-Mapping equals 2, the ouptuts can be used as PWM-outputs via qb0.0...0.7 (marked yellow)

Front view MOLEX connector:


12
11
10
9
8
7
6
5
4
3
2
1

7.8.7 POW1 Extension

Pin	PLC	Alternative	Connectiondata	Name	Note	User
	X307					
1			10...30VDC / 8A	PGND		GND
2			10...30VDC / 8A	Power supply		U_BAT
	X17					
1	%IB1.0		10VDC...30VDC	Digital input IB 1.0		
2	%IB1.1		10VDC...30VDC	Digital input IB 1.1		
3	%IB1.2		10VDC...30VDC	Digital input IB 1.2		
4	%IB1.3		10VDC...30VDC	Digital input IB 1.3		
5			PGND			
6	%IB1.4		10VDC...30VDC	Digital input IB 1.4		
7	%IB1.5		10VDC...30VDC	Digital input IB 1.5		
8	%IB1.6		10VDC...30VDC	Digital input IB 1.6		
9	%IB1.7		10VDC...30VDC	Digital input IB 1.7		
10			GND			
11	%IW24.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 0		
12	%IW26.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 1		
13	%IW28.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 2		
14	%IW30.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 3		
15			PGND			
16	%IW32.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 4		
17	%IW34.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 5		
18	%IW36.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 6		
19	%IW38.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 7		
20			PGND			
	X27					
1	%QB6.0		10-30VDC 0.5A^{*2}	Digital output QB 6.0		
2	%QB6.1		10-30VDC 0.5A^{*2}	Digital output QB 6.1		
3	%QB6.2		10-30VDC 0.5A^{*2}	Digital output QB 6.2		
4	%QB6.3		10-30VDC 0.5A^{*2}	Digital output QB 6.3		
5	%QB6.4		10-30VDC 0.5A^{*2}	Digital output QB 6.4		
6	%QB6.5		10-30VDC 0.5A^{*2}	Digital output QB 6.5		
7	%QB6.6		10-30VDC 0.5A^{*2}	Digital output QB 6.6		
8	%QB6.7		10-30VDC 0.5A^{*2}	Digital output QB 6.7		
	X37					
1	%QB5.0		Relay output 5A, 30VDC or	normally closed		
2				normally opened		
3				common		
4	%QB5.1		Relay output 5A, 30VDC or	normally closed		
5				normally opened		
6				common		
7	%QB5.2		Relay output 5A, 30VDC or	normally closed		
8				normally opened		
9				common		
10	%QB5.3		Relay output 5A, 30VDC or	normally closed		
11				normally opened		
12				common		
13	%QB5.4		Relay output 5A, 30VDC or	normally closed		
14				normally opened		
15				common		
16	%QB5.5		Relay output 5A, 30VDC or	normally closed		
17				normally opened		
18				common		
19	%QB5.6		Relay output 5A, 30VDC or	common		
20				normally opened		
21	%QB5.7		Relay output 5A, 30VDC or	common		
22				normally opened		

Table 7.14.: Pin Description List POW1 Extension

Description:

^{*1} default 0...10VDC. Otherwise as ordered.
Impedance: 0-10VDC (470$k\Omega$), 0-5VDC (94$k\Omega$), 4-20mA (220$k\Omega$)

^{*2} Transistor output to GND with pull-up

7.8.8 POW2 Extension

Pin	PLC	Alternative	Connectiondata	Name	Note	User
	X308					
1			10...30VDC / 8A	PGND		GND
2			10...30VDC / 8A	Power supply		U_BAT
	X18					
1	%IB2.0		10VDC...30VDC	Digital input IB 2.0		
2	%IB2.1		10VDC...30VDC	Digital input IB 2.1		
3	%IB2.2		10VDC...30VDC	Digital input IB 2.2		
4	%IB2.3		10VDC...30VDC	Digital input IB 2.3		
5			PGND			
6	%IB2.4		10VDC...30VDC	Digital input IB 2.4		
7	%IB2.5		10VDC...30VDC	Digital input IB 2.5		
8	%IB2.6		10VDC...30VDC	Digital input IB 2.6		
9	%IB2.7		10VDC...30VDC	Digital input IB 2.7		
10			GND			
11	%IW56.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 16		
12	%IW58.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 17		
13	%IW60.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 18		
14	%IW62.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 19		
15			PGND			
16	%IW64.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 20		
17	%IW66.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 21		
18	%IW68.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 22		
19	%IW70.0		0...5/0...10VDC / 4...20mA^{*1}	Analog input 23		
20			PGND			
	X28					
1	%QB7.0		10-30VDC 0.5A^{*2}	Digital output QB 7.0		
2	%QB7.1		10-30VDC 0.5A^{*2}	Digital output QB 7.1		
3	%QB7.2		10-30VDC 0.5A^{*2}	Digital output QB 7.2		
4	%QB7.3		10-30VDC 0.5A^{*2}	Digital output QB 7.3		
5	%QB7.4		10-30VDC 0.5A^{*2}	Digital output QB 7.4		
6	%QB7.5		10-30VDC 0.5A^{*2}	Digital output QB 7.5		
7	%QB7.6		10-30VDC 0.5A^{*2}	Digital output QB 7.6		
8	%QB7.7		10-30VDC 0.5A^{*2}	Digital output QB 7.7		
	X38					
1	%QB1.0		Relay output 5A, 30VDC or	normally closed		
2				normally opened		
3				common		
4	%QB1.1		Relay output 5A, 30VDC or	normally closed		
5				normally opened		
6				common		
7	%QB1.2		Relay output 5A, 30VDC or	normally closed		
8				normally opened		
9				common		
10	%QB1.3		Relay output 5A, 30VDC or	normally closed		
11				normally opened		
12				common		
13	%QB1.4		Relay output 5A, 30VDC or	normally closed		
14				normally opened		
15				common		
16	%QB1.5		Relay output 5A, 30VDC or	normally closed		
17				normally opened		
18				common		
19	%QB1.6		Relay output 5A, 30VDC or	common		
20				normally opened		
21	%QB1.7		Relay output 5A, 30VDC or	common		
22				normally opened		

Table 7.15.: Pin Description List POW2 Extension

Description:

^{*1} default 0...10VDC. Otherwise as ordered.
Impedance: 0-10VDC (470$k\Omega$), 0-5VDC (94$k\Omega$), 4-20mA (220$k\Omega$)

^{*2} Transistor output to GND with pull-up

8 PLVC8

8.1 Technical Data

8.1.1 General Data

Enclosure	IP 67 (IEC60529)
Temperature range	$-40°C$ to $+80°C$
Supply voltage	10V DC to 30V DC
max. total electricity	2x8A, 1A (logic)
Required external fuse	2x10A slow-blow, 1x1A slow-blow
Protection	Against reverse polarity
Permission (only for PLVC 8x2-G and PLVC8x2-X-EW)	E13-permission (ECE-R10 Rev. 3, CISPR 25 ISO 7637-2:2004 ISO 11452-2:2004 ISO 11452-5:2002)
Monitoring	short circuit, untervoltage, overvoltage, cablebreak
Connection	The neccessary connectors aren't scope of the delivery and therefore have to be ordered seperately. For detailed Information, see Table 8.2
Microcontroller 1 (basic module)	ST10F276 basic parameter memory: EEPROM 1000 words memory: flash: 768kByte memory: RAM: 420kByte
Microcontroller 2 (basic module)	32 bit memory flash: 32kByte memory RAM: 8kByte
Microcontroller 3 (extension module)	32bit memory flash: 32kByte memory RAM: 8kByte
Fixing	4xM6 screws

Continued on the next page. . .

. . . Continued from previous Page

Housingmaterial	aluminium, anodized
Ground (weight)	basic module ca 2,4kg basic module with extension module ca 2,6kg

Description	Part-number	note
connector-set	6217 2066-00	(complete inklusive 6217 2067-00; need:1x = basic module / 2x = basic module + extension module
connector-contacts	6217 2067-00	(spare part; content: 30x replacementcontacts + 15x replacementblankets)
crimping tool	6217 2068-00	(recommended)
contact-extraction-tool	6271 2069-00	(recommended)
Connector-tool	6271 2074-00	(recommended to decompose connector)

Table 8.2.: Equipment for Connection

8.1.2 Base Unit

Features of the Basic Devices

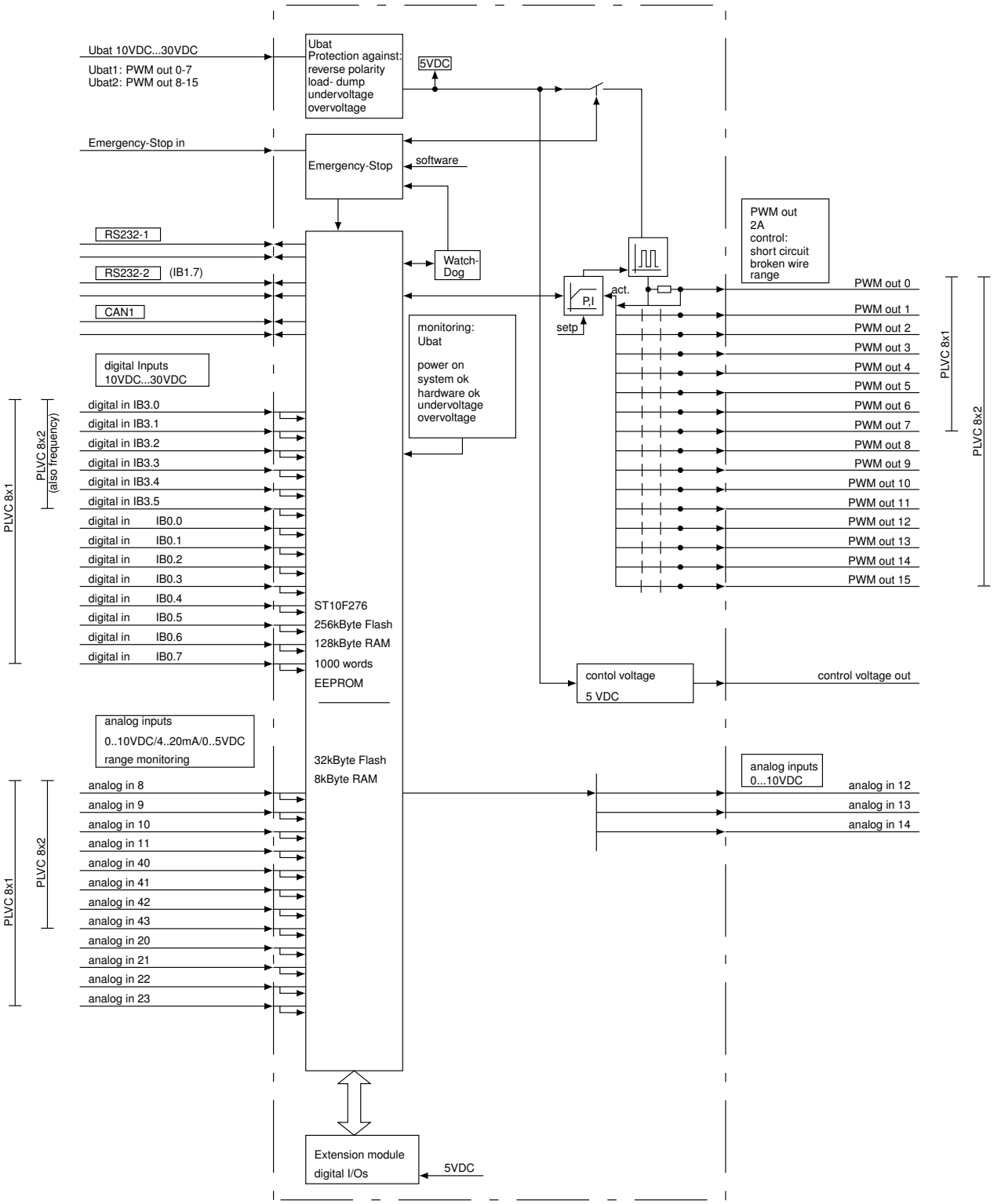
- PLVC 8x1
  - 8 outputs for prop or b/w-valves (current controlled) 2A
  - 15 analog Inputs (for joystick, potentiometer, sensors, e.g. analog pressure sensors)
  - 14 digital Inputs (for limit switches, pressure switches, buttons, calliper etc. with the possibility to be used as frequency input for encoder, tachometer, incremental encoder, etc.)
  - Emergency-stop-input
  - Interface for RS232 and CAN-Bus
  - Power supply 10. . . 30V DC, max 16A
- PLVC 8x2
  - 16 outputs for prop or b/w-valves (current controlled) 2A
  - 11 analog Inputs (for joystick, potentiometer, sensors, e.g. analog pressure sensors)
  - 6 digital Inputs (for limit switches, pressure switches, buttons, calliper etc. with the possibility to be used as frequency input for encoder, tachometer, incremental encoder, etc.)

- Emergency-stop-input
- Interface for RS232 and CAN-Bus
- Power supply 10...30V DC, max 16A

Performance of the Connections

Function	Description	Parameters
Powersupply	Nominal voltage U_N	10-30V DC
	max. total current	2x8A
prop. bzw. s/w-outputs	I_{min}	100...1200mA
(with high-side-mesurement)	I_{max}	100...2000mA
PLVC 8x2: 0-15	Ditherfrequency	33-200Hz
PLVC 8x1: 0-7	Ditheramplitude (referring to PWM)	0...48%
	cold resistance	6...35Ω
digital inputs	voltage range	10...30V DC
PLVC 8x2: IB3.0 to IB3.5	input resistance	7Ω
also as frequency inputs	Cutoff frequency	$f_{Grenz} = 5kHz$
and		
IB1.7	voltage range	10...30V DC / 3-7$k\Omega$
and		
IB0.0 to IB0.7	voltage range	10...30V DC / 24$k\Omega$
analog inputs	10bit A DC $\hat{=}$ 1024 steps	
PLVC 8x2		
8-11	configurable with software	0...5V DC / 470$k\Omega$
		0...10V DC / 100$k\Omega$
		4...20mA / 220Ω
and	configurable wit software	0...5V DC / 470$k\Omega$
		0...10V DC / 100$k\Omega$
		4...20mA / 220Ω
PLVC 8x1: same as PLVC 8x2		
and		
20-23		0...10V DC / 24$k\Omega$
analog anns digital inputs	10bit A DC $\hat{=}$ 1024steps	
PLVC 8 (x2 and x1)		
12-14	configurable with software	0...10 V DC / 100$k\Omega$
		10...30V DC / 7$k\Omega$
Interface CAN-Bus	Interface parameter	CAN interface 2.0, ISO11898
		50...1000kBit/sec
		protocol: CANOpen/J1939

Block Diagram Basic Device



8.1.3 Extension Module

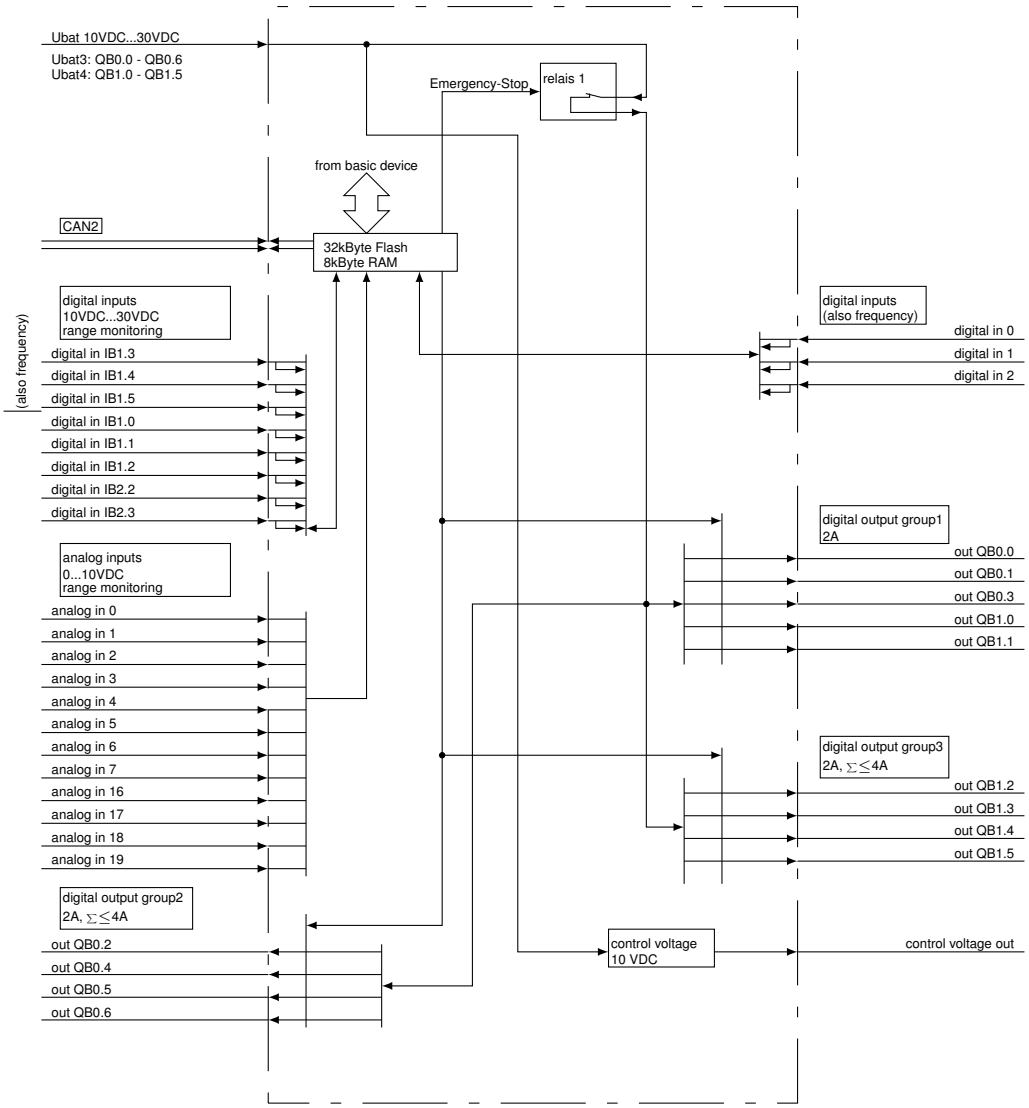
General Properties

Power supply	10...20V DC
max. total current	2x8A
necessary ext. security	2x8A
Fixing	fitted in basic unit

Features of the extension module

- 11 digital Inputs (for limit switches, pressure switches, buttons, calliper etc. with the possibility to be used as frequency input for encoder, tachometer, incremental encoder, etc.)
- 13 digital outputs for ohmic or inductive consumers
- 12 analog inputs (for joystick, potentiometer, sensors, e.g. analog pressure sensors)
- CAN-Bus
- power supply 10...30V DC, max. 16A

Block Diagramm of Extension Module



Performance of the Connections of the Extension

Function	Description	Parameters
Power supply	Nominal voltage U_N	10-30V DC
	max. total current	2x8A
digital outputs	for b/w-valves and ohmic	10...30V DC
QB0.0 to QB0.6 and	consumers	2A(max 4A per group)
QB1.0 tu QB1.5		
digital inputs	Voltage range	10...30V DC
IB1.3 to IB1.5	input resistance	7$k\Omega$
also as frequency inputs	Cutoff frequency	$f_{Grenz} = 5kHz$
and		
IB2.0	Voltage range	10...30V DC / 3-7$k\Omega$
IB1.0 to IB 1.2	Voltage range	10...30V DC / 7$k\Omega$
IB2.2 to IB2.3	Voltage range	10...30V DC / 7$k\Omega$
IB2.4 to IB2.6	Voltage range	10...30V DC / 11$k\Omega$
analog inputs 0-7	Voltage range	0...10V DC / 26$k\Omega$
can also be used digital 16-19		10...30V DC / 26$k\Omega$
(suitable with electonic switches and		
sensors)		
Interface CAN-Bus	Interface parameters	Can Interface 2.0, ISO 11898
		50...100kBits/sec
		Protocol: CANOpen/J1939

8.2 Download an Operating System

The operating system can be updated using a Windows PC.

8.2.1 With an Intact Operating System

A new operating system is easy to install via an already existing operating system. All functionality for an update is already included in the existing operating system. Connect the PLVC control via serial interface to the PC and start the appropriate upload program of the operating system.

8.2.2 Damaged Operating System

If you cannot activate your current operating system (e.g. by an aborted operating system download), a new operating system can be installed though.
For this, the PLVC must be started in a spcial mode.
First the control should be connected to a PC via serial interface (RS232).
The following steps are neccessary:

- Switch control off

- Put pin G2 high (10. . . 30V)
- Switch control on
- Start operating system upload
- Take voltage away from pin G2

8.3 Mechanical Installation

8.4 Pin Description Lists

8.4.1 PLVC8x1-G

Pin	PLC	Alternative	Connectiondata	Name	Note	User
Y3			10...30V DC / 8A	Valve Supply (Coils 0...7)	ESTOP Transistor1	U_POW 1
Y1	0		Coil 0 / 24V DC, max. 2A DC	Coil A proportional Valve 0	Also on/off Valve	
Y2	1		Coil 1 / 24VDC, max. 2ADC	Coil B proportional Valve 1	Also on/off Valve	
X2			Measurement input for Y1,Y2		measurement input	
W2	2		Coil 2 / 24V DC, max. 2A DC	Coil A proportional Valve 2	Also on/off Valve	
X1	3		Coil 3 / 24V DC, max. 2A DC	Coil B proportional Valve 3	Also on/off Valve	
W1			Measurement input for W2, X1		measurement input	
T1	4		Coil 4 / 24V DC, max. 2A DC	Coil A proportional Valve 4	also on/off Valve	
T2	5		Coil 5 / 24V DC, max. 2A DC	Coil B proportional Valve 5	Also on/off Valve	
T3			Measurement input for T1, T2		measurement input	
S2	6		Coil 6 / 24V DC, max. 2A DC	Coil A proportional Valve 6	Also on/off Valve	
S1	7		Coil 7 / 24VDC, max. 2ADC	Coil B proportional Valve 7	Also on/off Valve	
S3			Measurement input for S2, S1		measurement input	
A3			NOT USED			
A1	IB0.0		10VDC...30VDC 94kΩ	Digital Input IB0.0		
A2	IB0.1		10VDC...30VDC 94kΩ	Digital Input IB0.1		
B2	IW64.0	IB64.0	0...10VDC 100kΩ	Analog Input 20		
C2	IB0.2		10VDC...30VDC 94kΩ	Digital Input IB0.2		
B1	IB0.3		10VDC...30VDC 94kΩ	Digital Input IB0.3		
C1	IW66.0	IB66.0	0...10VDC 100kΩ	Analog Input 21		
D1	IB0.4		10VDC...30VDC 94kΩ	Digital Input IB0.4		
D2	IB0.5		10VDC...30VDC 94kΩ	Digital Input IB0.5		
D3	IW68.0	IB68.0	0...10VDC 100kΩ	Analog Input 22		
E2	IB0.6		10VDC...30VDC 94kΩ	Digital Input IB0.6		
E1	IB0.7		10VDC...30VDC 94kΩ	Digital Input IB0.7		
E3	IW70.0	IB70.0	0...10VDC 100kΩ	Analog Input 23		
M2			50, 100, 125, 250, 500, 1000kB	CAN1_H	CAN Bus	
M3			50, 100, 125, 250, 500, 1000kB	CAN1_L	CAN Bus	
K3			RXD_1	RS232 Data cable	ST10 RS-232 RX	
J3			TXD_1	RS232 Data cable	ST10 RS-232 TX	
H3	IB1.7		RXD_2	RS232 second Controller	also dig. Input*3	
J2			TXD_2	RS232 second Controller		
M1	IB3.1	Fq0	10..30VDC 7kΩ 5kHz	Digital Input IB3.1	also Frequency	
N1	IB3.2	Fq1	10..30VDC 7kΩ 5kHz	Digital Input IB3.2	also Frequency	
P1	IB3.0	Fq2	10..30VDC 7kΩ 5kHz	Digital Input IB3.0	also Frequency	
R1	IB3.3	Fq3	10..30VDC 7kΩ 5kHz	Digital Input IB3.3	also Frequency	
R2	IB3.4	Fq4	10..30VDC 7kΩ 5kHz	Digital Input IB3.4	also Frequency	
P3	IB3.5	Fq5	10..30VDC 7kΩ 5kHz	Digital Input IB3.5	also Frequency	
G3	IW40.0	IB40.0	0..5 / 0..10VDC / 4..20mA*1	Analog Input 8 C2	for Joystick/Pot	
F2	IW42.0	IB42.0	0..5 / 0..10VDC / 4..20mA*1	Analog Input 9 C2	for Joystick/Pot	
F1	IW44.0	IB44.0	0..5 / 0..10VDC / 4..20mA*1	Analog Input 10 C2	for Joystick/Pot	
G1	IW46.0	IB46.0	0..5 / 0..10VDC / 4..20mA*1	Analog Input 11 C2	for Joystick/Pot	
H1	IW104.0	IB104.0	0..5 / 0..10VDC / 4..20mA*1	Analog Input 40	for Joystick/Pot	
J1	IW106.0	IB106.0	0..5 / 0..10VDC / 4..20mA*1	Analog Input 41	for Joystick/Pot	
K1	IW108.0	IB108.0	0..5 / 0..10VDC / 4..20mA*1	Analog Input 42	for Joystick/Pot	
K2	IW110.0	IB110.0	0..5 / 0..10VDC / 4..20mA*1	Analog Input 43	for Joystick/Pot	
L3	IW48.0	IB48.0	0..10VDC / 10..30VDC*2	Analog Input 12 / dig. Input C2		
L2	IW50.0	IB50.0	0..10VDC / 10..30VDC*2	Analog Input 13 / dig. Input		
L1	IW52.0	IB52.0	0..10VDC / 10..30VDC*2	Analog Input 14 / dig. Input		
H2	IB3.7	IB4.0	10..30VDC	Emergency Stop Input	for both Controllers	ESTOP
G2			BSL	Firmware Download*4	for both Controllers	
R3			10...30VDC / 2A	Supply Input for Controllers		U_BAT
F3	IW54.0	IB54.0	10 ... 30VDC / 2A Permanent	Backup Supply Voltage*5	also Analog Input 15	
N3			5V / 500mA	U_SENSOR		
P2			reserved			
X3			PGND			GND
C3			PGND			GND
B3			PGND			GND
W3			Sensor GND			
N2			Termination	120 Ohm to CAN_Low int.	Connect. to Pin M3*6	

Table 8.3.: Pin Description List PLVC8x1

Description

^{*1} Analog Input: the configuration can be changed via software Parameters.
Input resistance: 0...5V DC≙470kΩ / 0...10V DC≙100kΩ /^{C1} 4...20mA≙220Ω /^{C2} 4...20mA≙150Ω

^{*2} Analog or digital input: the configuration can be changed via software parameters.
Input resistance: 0...10V DC≙100kΩ / digital≙7kΩ

^{*3} Can be used alternatively as digital input.
Input resistance: 5kΩ

^{*4} Use after interrupted firmware-download

^{*5} Used for EE-Save or can be used alternatively as analog input.
EE-Save=If wired with supply independent of main supply, device will switch off itself with a delay of 2sec. So Parameters can be safed.

^{*6} Connect to M3 if termination is used.

^{C2} Input works on the second controller.

8.4.2 PLVC8x2-X-EW

Pin	PLC	Alternative	Connection data	Name	Note	
A3	IB1.6 *15		10...30V DC / 8A	Supply (QB0.0...QB0.6)		U_POW 3
A1	QB0.0	Channel 16	10VDC...30VDC	Digital / PWM-Output*09	2A	
A2	QB0.1	Channel 17	10VDC...30VDC	Digital / PWM-Output*09	2A	
B1	QB0.2	Channel 18	10VDC...30VDC	Digital / PWM-Output*09	2A*10	
C2	QB0.3	Channel 19	10VDC...30VDC	Digital / PWM-Output*09	2A	
B3			PGND			
D1	QB0.4	Channel 20	10VDC...30VDC	Digital / PWM-Output*09	2A*10	
D2	QB0.5	Channel 21	10VDC...30VDC	Digital / PWM-Output*09	2A*10	
E1	QB0.6	Channel 22	10VDC...30VDC	Digital / PWM-Output*09	2A*10	
C3			PGND			
B2	IW24.0	IB24.0	0..10VDC 26kΩ	Analog Input 0	also digital Input	
C1	IW26.0	IB26.0	0..10VDC 26kΩ	Analog Input 1	also digital Input	
D3	IW28.0	IB28.0	0..10VDC 26kΩ	Analog Input 2	also digital Input	
E2	IW30.0	IB30.0	0..10VDC 26kΩ	Analog Input 3	also digital Input	
E3	IW32.0	IB32.0	0..10VDC 26kΩ	Analog Input 4	also digital Input	
F1	IW34.0	IB34.0	0..10VDC 26kΩ	Analog Input 5	also digital Input	
F2	IW36.0	IB36.0	0..10VDC 26kΩ	Analog Input 6	also digital Input	
F3	IW38.0	IB38.0	0..10VDC 26kΩ	Analog Input 7	also digital Input	
G1	IW56.0	IB56.0	0..10VDC 26kΩ	Analog Input 16	also digital Input	
G3	IW58	IB58.0	0..10VDC 26kΩ	Analog Input 17	also digital Input	
H1	IB1.0		10..30VDC 7kΩ	Digital Input^{C1}		
H2	IB1.1		10..30VDC 7kΩ	Digital Input^{C1}		
J1	IB1.2		10..30VDC 7kΩ	Digital Input^{C1}		
J2	IB1.3	fq8	10..30VDC 7kΩ 5kHz	Digital Input^{C1}	also Frequency	
J3			TXD_1	RS-232 transmit		
K1			CAN2_H	CAN Bus 2		
K2			CAN2_L	CAN Bus 2		
K3	IB2.0		RXD_1	RS-232 receive	also dig. Input*13	
H3			10V / 200mA	U_SENSOR from Basic		
G2			BSL	Software Download Extension Controller*12		
Y3	IB1.7		10...30V DC / 8A	Supply (QB1.0...QB1.5)		U_POW
Y1	QB1.0	Channel 24	10VDC ... 30VDC	Digital / PWM-Output*09	2A	
Y2	QB1.1	Channel 25	10VDC ... 30VDC	Digital / PWM-Output*09	2A	
W2	QB1.2	Channel 26	10VDC ... 30VDC	Digital / PWM-Output*09	2A*11	
X1	QB1.3	Channel 27	10VDC ... 30VDC	Digital / PWM-Output*09	2A*11	
T1	QB1.4	Channel 28	10VDC ... 30VDC	Digital / PWM-Output*09	2A*11	
T2	QB1.5	Channel 29	10VDC ... 30VDC	Digital / PWM-Output*09	2A*11	
W3				GND		
X3				Sensor GND		
X2	IW60.0	IB60.0	0..10VDC 26kΩ	Analog Input 18	also digital Input	
W1	IW62.0	IB62.0	0..10VDC 26kΩ	Analog Input 19	also digital Input	
L2	IB1.4	fq6	10..30VDC 7kΩ 5kHz	Digital Input	also Frequency	
L1	IB1.5	fq7	10..30VDC 7kΩ 5kHz	Digital Input	also Frequency	
M1	IB2.2	(fq)	10..30VDC 7kΩ	Digital Input		
M2	IB2.3	(fq)	10..30VDC 7kΩ	Digital Input		
N1	IB2.4		10..30VDC 11kΩ	Digital Input		
N2	IB2.5		10..30VDC 11kΩ	Digital Input		
P1	IB2.6		10..30VDC 11kΩ	Digital Input		
R3	ib7.5		10...30V DC, 9k	Supply Input for 5V *14		
N3	ib2.1		10V / 200mA, 2kΩ	10V Output (Option)*14		
R2	ib5.6	(fq)	10..30VDC 7kΩ 5kHz	digital Input^{C2}		
R1	ib5.7	(fq)	10..30VDC 7kΩ 5kHz	digital Input^{C2}		
P2	ib7.0		10..30VDC 7kΩ	Digital Input		
P3	ib7.1		10..30VDC 7kΩ	Digital Input		
L3	ib7.4		10..30VDC 27kΩ	Digital Input		
S1	ib7.2	(qb1.6) (fq)	10..30VDC 7kΩ	Digital Input		
S2	ib7.3	(qb1.7)(fq)	10..30VDC 7kΩ	Digital Input		
S3	ib7.7		10..30VDC 27kΩ	Digital Input		
T3	ib7.6		10..30VDC 27kΩ	Digital Input		

Table 8.4.: Pin Description List PLVC8x2-X-EW

Description

delivered since Oct 2012, additional Features marked yellow, features in brackets possible but not realized yet.

*09 since software version 12/2009 not only digital outputs are supported, but also PWM.

*10 Output-Group 1: Max.current of single output: 2A. Max. current of group: 4A

*11 Output-Group 2: Max.current of single output: 2A. Max. current of group: 4A

*12 Used for firmware download

*13 Can be used alternativley as digital input.
Input resistance: 5kΩ

*14 If 10V option is used (-X-EW/10V), R3 needs UBAT to generate 10V Output at N3; if option not used, dig. Inputs for R3 N3 available

*15 For diagnosis only, not a real dig. Input

8.4.3 PLVC8x2-G

Pin	PLC	Alternative	Connection data	Name	Note	User
Y3			10...30V DC / 8A	Valve Supply (Coils 0...7)	ESTOP Transistor 1	U_POW 1
Y1	0		Coil 0 / 24V DC, max. 2A DC	Coil A proportional Valve 0	Also on/off Valve	
Y2	1		Coil 1 / 24VDC, max. 2ADC	Coil B proportional Valve 1	Also on/off Valve	
X2			Measurement input for Y1,Y2		measurement input	
W2	2		Coil 2 / 24V DC, max. 2A DC	Coil A proportional Valve 2	Also on/off Valve	
X1	3		Coil 3 / 24V DC, max. 2A DC	Coil B proportional Valve 3	Also on/off Valve	
W1			Measurement input for W2, X1		measurement input	
T1	4		Coil 4 / 24V DC, max. 2A DC	Coil A proportional Valve 4	also on/off Valve	
T2	5		Coil 5 / 24V DC, max. 2A DC	Coil B proportional Valve 5	Also on/off Valve	
T3			Measurement input for T1, T2		measurement input	
S2	6		Coil 6 / 24V DC, max. 2A DC	Coil A proportional Valve 6	Also on/off Valve	
S1	7		Coil 7 / 24VDC, max. 2ADC	Coil B proportional Valve 7	Also on/off Valve	
S3			Measurement input for S2, S1		measurement input	
A3			10 ... 30VDC / 8A	Valve Supply (Coils 8...15)	ESTOP Transistor 2	U_POW 2
A1	8		Coil 8 / 24VDC, max. 2ADC	Coil A proportional Valve 8	Also on/off Valve	
A2	9		Coil 9 / 24VDC, max. 2ADC	Coil B proportional Valve 9	Also on/off Valve	
B2			Measurement input for A1, A2			
C2	10		Coil 10 / 24VDC, max. 2ADC	Coil A proportional Valve 10	Also on/off Valve	
B1	11		Coil 11 / 24VDC, max. 2ADC	Coil B proportional Valve 11	Also on/off Valve	
C1			Measurement input for C2, B1			
D1	12		Coil 12 / 24VDC, max. 2ADC	Coil A proportional Valve 12	Also on/off Valve	
D2	13		Coil 13 / 24VDC, max. 2ADC	Coil B proportional Valve 13	Also on/off Valve	
D3			Measurement input for D1, D2			
E2	14		Coil 14 / 24VDC, max. 2ADC	Coil A proportional Valve 14	Also on/off Valve	
E1	15		Coil 15 / 24VDC, max. 2ADC	Coil B proportional Valve 15	Also on/off Valve	
E3			Measurement input for E2, E1			
M2			50, 100, 125, 250, 500, 1000kB	CAN1_H	CAN Bus	
M3			50, 100, 125, 250, 500, 1000kB	CAN1_L	CAN Bus	
K3			RXD_1	RS232 Data cable	ST10 RS-232 RX	
J3			TXD_1	RS232 Data cable	ST10 RS-232 TX	
H3	IB4.1		RXD_2	RS232 second Controller	also dig. Input $*^{3\,C2}$	
J2			TXD_2	RS232 second Controller		
M1	IB3.1	Fq0	10..30VDC 7kΩ 5kHz	Digital Input IB3.1	also Frequency	
N1	IB3.2	Fq1	10..30VDC 7kΩ 5kHz	Digital Input IB3.2	also Frequency	
P1	IB3.0	Fq2	10..30VDC 7kΩ 5kHz	Digital Input IB3.0	also Frequency	
R1	IB3.3	Fq3/IB5.3	10..30VDC 7kΩ 5kHz	Digital Input IB3.3 $^{C1/C2}$	also Frequency	
R2	IB3.4	Fq4/IB5.4	10..30VDC 7kΩ 5kHz	Digital Input IB3.4 $^{C1/C2}$	also Frequency	
P3	IB3.5	Fq5/IB5.5	10..30VDC 7kΩ 5kHz	Digital Input IB3.5 $^{C1/C2}$	also Frequency	
G3	IW40.0	IB40.0	0..5 / 0..10VDC / 4..20mA^{*1}	Analog Input 8 ^{C2}	for Joystick/Pot	
F2	IW42.0	IB42.0	0..5 / 0..10VDC / 4..20mA $*^{1}$	Analog Input 9 ^{C2}	for Joystick/Pot	
F1	IW44.0	IB44.0	0..5 / 0..10VDC / 4..20mA^{*1}	Analog Input 10 ^{C2}	for Joystick/Pot	
G1	IW46.0	IB46.0	0..5 / 0..10VDC / 4..20mA^{*1}	Analog Input 11 ^{C2}	for Joystick/Pot	
H1	IW104.0	IB104.0	0..5 / 0..10VDC / 4..20mA^{*1}	Analog Input 40 ^{C1}	for Joystick/Pot	
J1	IW106.0	IB106.0	0..5 / 0..10VDC / 4..20mA^{*1}	Analog Input 41 ^{C1}	for Joystick/Pot	
K1	IW108.0	IB108.0	0..5 / 0..10VDC / 4..20mA^{*1}	Analog Input 42 ^{C1}	for Joystick/Pot	
K2	IW110.0	IB110.0	0..5 / 0..10VDC / 4..20mA^{*1}	Analog Input 43 ^{C1}	for Joystick/Pot	
L3	IW48.0	IB48.0	0..10VDC / 10..30VDC^{*2}	Analog Input 12 ^{C2}		
L2	IW50.0	IB50.0/IB6.7	0..10VDC / 10..30VDC^{*2}	Analog Input 13 ^{C1}/ dig. Inp. ^{C2}	also dig. Input for C2	
L1	IW52.0	IB52.0	0..10VDC / 10..30VDC^{*2}	Analog Input 14 ^{C1}		
H2	IB3.7	IB4.0	10..30VDC	Emergency Stop Input	for both Controllers	ESTOP
G2			BSL	Firmware Download^{*4}	for both Controllers	
R3				Supply Input for Controllers		U_BAT
F3	IW54.0	IB54.0	10 ... 30VDC / 2A Permanent	Backup Supply Voltage^{*5}	also Analog Input 15	
N3			**5V / 450mA**	U_SENSOR		
P2			reserved			
X3			PGND			GND
C3			PGND			GND
B3			PGND			GND
W3			Sensor GND			
N2			Termination	120 Ohm to CAN_Low int.	Connect. to Pin M3^{*6}	

Table 8.5.: Pin Description List PLVC8x2-G*

Description

* Delivered since Oct. 2010, additions to older version marked red

^{*1} Analog input: the configuration can be changed via software parameters.
Input resistance: 0..5VDC = 470kΩ / 0..10VDC = 100kΩ / C1 4..20mA = 220Ω / C2 4..20mA = 150Ω

^{*2} can also be used as dig. Input

^{*3} Can be used alternatively as digital input.
Input resistance: 5kΩ max 30V!

^{*4} Use after interrupted firmware-download

^{*5} Used for EE-Save or can be used alternatively as analog input.
EE-Save = If wired with supply independend of main supply, device will switch off itself with a delay of 2 sec. So parameters can be safed.

^{*6} Connect to M3 if termination is used.

^{C2} Input read by second controller.
$^{C1/C2}$Input read by both controller.

^{C1} Input read by main controller.

8.4.4 PLVC8x2-G-J

Pin	PLC	Alternative	Connection data	Name	Note	User
Y3			10...30V DC / 8A	Valve Supply (Coils 0...7)	ESTOP Transistor 1	U_POW 1
Y1	0		Coil 0 / 24V DC, max. 2A DC	Coil A proportional Valve 0	Also on/off Valve	
Y2	1		Coil 1 / 24VDC, max. 2ADC	Coil B proportional Valve 1	Also on/off Valve	
X2			Measurement input for Y1,Y2		measurement input	
W2	2		Coil 2 / 24V DC, max. 2A DC	Coil A proportional Valve 2	Also on/off Valve	
X1	3		Coil 3 / 24V DC, max. 2A DC	Coil B proportional Valve 3	Also on/off Valve	
W1			Measurement input for W2, X1		measurement input	
T1	4		Coil 4 / 24V DC, max. 2A DC	Coil A proportional Valve 4	also on/off Valve	
T2	5		Coil 5 / 24V DC, max. 2A DC	Coil B proportional Valve 5	Also on/off Valve	
T3			Measurement input for T1, T2		measurement input	
S2	6		Coil 6 / 24V DC, max. 2A DC	Coil A proportional Valve 6	Also on/off Valve	
S1	7		Coil 7 / 24VDC, max. 2ADC	Coil B proportional Valve 7	Also on/off Valve	
S3			Measurement input for S2, S1		measurement input	
A3			10 ... 30VDC / 8A	Valve Supply (Coils 8...15)	ESTOP Transistor 2	U_POW 2
A1	8		Coil 8 / 24VDC, max. 2ADC	Coil A proportional Valve 8[*7]	Also on/off Valve	
A2	9		Coil 9 / 24VDC, max. 2ADC	Coil B proportional Valve 9[*7]	Also on/off Valve	
B2	IW64.0	IB64.0	0-10V DC 20kΩ	Analog Input 20		
C2	10		Coil 10 / 24VDC, max. 2ADC	Coil A proportional Valve 10[*7]	Also on/off Valve	
B1	11		Coil 11 / 24VDC, max. 2ADC	Coil B proportional Valve 11[*7]	Also on/off Valve	
C1	IW66.0	IB66.0	0-10V DC 20kΩ	Analog Input 21		
D1	12		Coil 12 / 24VDC, max. 2ADC	Coil A proportional Valve 12[*7]	Also on/off Valve	
D2	13		Coil 13 / 24VDC, max. 2ADC	Coil B proportional Valve 13[*7]	Also on/off Valve	
D3	IW68.0	IB68.0	0-10V DC 20kΩ	Analog Input 22		
E2	14		Coil 14 / 24VDC, max. 2ADC	Coil A proportional Valve 14[*7]	Also on/off Valve	
E1	15		Coil 15 / 24VDC, max. 2ADC	Coil B proportional Valve 15[*7]	Also on/off Valve	
E3	IW70.0	IB70.0	0-10V DC 20kΩ	Analog Input 23		
M2			50, 100, 125, 250, 500, 1000kB	CAN1_H	CAN Bus	
M3			50, 100, 125, 250, 500, 1000kB	CAN1_L	CAN Bus	
K3			RXD_1	RS232 Data cable	ST10 RS-232 RX	
J3			TXD_1	RS232 Data cable	ST10 RS-232 TX	
H3	IB4.1		RXD_2	RS232 second Controller	also dig. Input [*3]	
J2			TXD_2	RS232 second Controller		
M1	IB3.1	Fq0	10..30VDC 7kΩ 5kHz	Digital Input IB3.1	also Frequency	
N1	IB3.2	Fq1	10..30VDC 7kΩ 5kHz	Digital Input IB3.2	also Frequency	
P1	IB3.0	Fq2	10..30VDC 7kΩ 5kHz	Digital Input IB3.0	also Frequency	
R1	IB3.3	Fq3	10..30VDC 7kΩ 5kHz	Digital Input IB3.3	also Frequency	
R2	IB3.4	Fq4	10..30VDC 7kΩ 5kHz	Digital Input IB3.4	also Frequency	
P3	IB3.5	Fq5	10..30VDC 7kΩ 5kHz	Digital Input IB3.5	also Frequency	
G3	IW40.0	IB40.0	0..5 / 0..10VDC / 4..20mA[*1]	Analog Input 8 [C2]	for Joystick/Pot	
F2	IW42.0	IB42.0	0..5 / 0..10VDC / 4..20mA [*1]	Analog Input 9 [C2]	for Joystick/Pot	
F1	IW44.0	IB44.0	0..5 / 0..10VDC / 4..20mA[*1]	Analog Input 10 [C2]	for Joystick/Pot	
G1	IW46.0	IB46.0	0..5 / 0..10VDC / 4..20mA[*1]	Analog Input 11 [C2]	for Joystick/Pot	
H1	IW104.0	IB104.0	0..5 / 0..10VDC / 4..20mA[*1]	Analog Input 40	for Joystick/Pot	
J1	IW106.0	IB106.0	0..5 / 0..10VDC / 4..20mA[*1]	Analog Input 41	for Joystick/Pot	
K1	IW108.0	IB108.0	0..5 / 0..10VDC / 4..20mA[*1]	Analog Input 42	for Joystick/Pot	
K2	IW110.0	IB110.0	0..5 / 0..10VDC / 4..20mA[*1]	Analog Input 43	for Joystick/Pot	
L3	IW48.0	IB48.0	0..10VDC / 10..30VDC[*2]	Analog Input 12 / dig Input[C2]		
L2	IW50.0	IB50.0	0..10VDC / 10..30VDC[*2]	Analog Input 13 / dig. Input		
L1	IW52.0	IB52.0	0..10VDC / 10..30VDC[*2]	Analog Input 14 / dig. Input		
H2	IB3.7	IB4.0	10..30VDC	Emergency Stop Input	for both Controllers	ESTOP
G2			BSL	Firmware Download[*4]	for both Controllers	
R3			10...30V DC / 2A	Supply Input for Controllers		U_BAT
F3	IW54.0	IB54.0	10 ... 30VDC / 2A Permanent	Backup Supply Voltage[*5]	also Analog Input 15	
N3			**5V / 500mA**	U_SENSOR		
P2			reserved			
X3			PGND			GND
C3			PGND			GND
B3			PGND			GND
W3			Sensor GND			
N2			Termination	120 Ohm to CAN_Low int.	Connect. to Pin M3[*6]	

Table 8.6.: Pin Description List PLVC8x2-G-J

Description PLVC8x2-G-J

*1 Analog input: the configuration can be changed via software parameters.
Input resistance: 0..5VDC = 470kΩ / 0..10VDC = 100kΩ / C1 4..20mA = 220Ω / C2 4..20mA = 150Ω

*2 Analog or digital input: the configuration can be changed via software parameters.
Input resistance: 0..10VDC = 100kΩ / digital = 7kΩ

*3 Can be used alternatively as digital input.
Input resistance: 5kΩ

*4 Use after interrupted firmware-download

*5 Used for EE-Safe or can be used alternatively as analog input.
EE-Safe = If wired with supply independent of main supply, device will switch of itself with a delay of 2 sec. So parameters can be safed.

*6 Connect to M3 if termination is used.

*7 Only PWM mode without ground measurement

c2 Input works on the second controller.

8.5 How to remove cripm contacts from PLVC8 plug

To remove crimp contacts from a PLVC8 plug a universal terminal removal tool was designed, unlatching and pushing out the terminated wires.

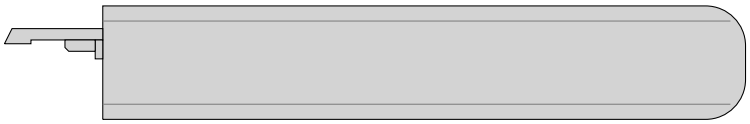
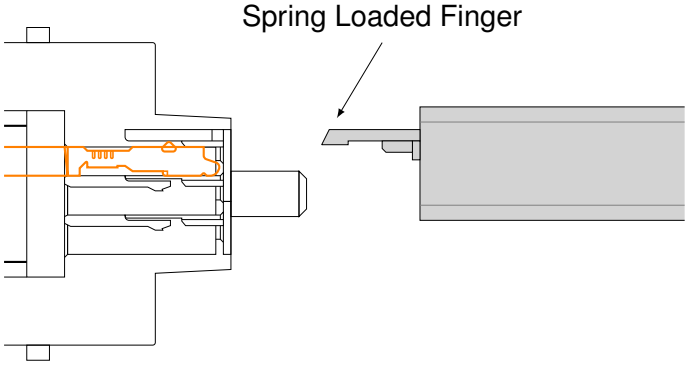
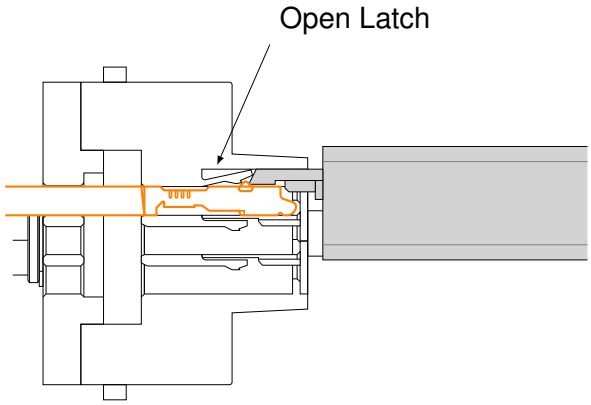


Figure 8.1.: Universal terminal removal tool

The tool is designed so that the longer spring loaded finger lifts the latch out of the terminal locking window.



The shorter finger is stationary and will push out the terminal once the latch is lifted.



Watch out to insert the removal tool carefully and don't push it all the way against the connector. Otherwise the plastic spring, which is lifted by the tool will break.

Also take care to adjust the removal tool right, as the following figure illustrates.

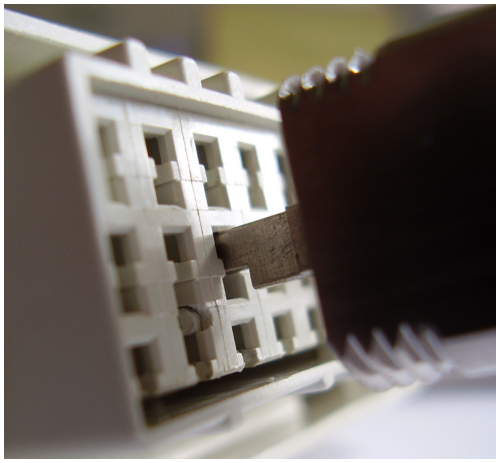


Figure 8.2.: RIGHT!

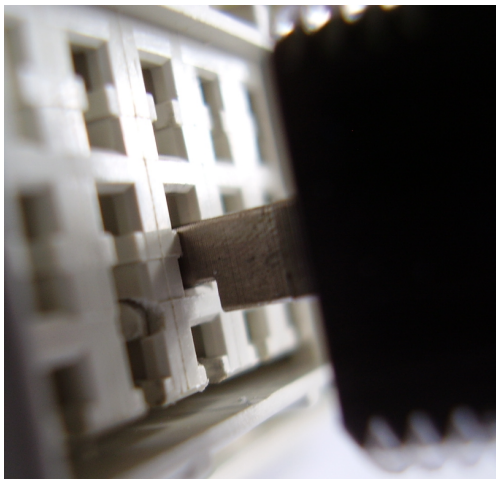


Figure 8.3.: WRONG!

Part III.

Configuration, Diagnosis and Programming

9 Basic Information

- PLC programming with ST (Structured Text in accordance to IEC61131)
- Parameterization during operating time
- CAN bus support is integrated in the operating system
- Operating system: support for typical hydraulic applications such as valve positioning, measurement control, ramp definition, current control

10 Configuration- and Diagnosis-Software “PLVC Visual Tool”

10.1 Free Version

To configure and monitor the controls of the PLVC, the free Windows software ’PLVC Visual Tool’ is available. This software provides following functionality:

- Monitor and configure all inputs and outputs of the control
- Create a project for each control
- Free choice of naming all the inputs and outputs
- Export of terminal diagrams in various formats (PDF, Excel)
- Load and save programs and parameters
- Transfer new operating system
- Update via the internet
- And much more. . .

10.2 Extended Version With Costs

In addition to the free software version, an extended software version is available. It contains an integrated oscilloscope.

The oscilloscope has following functionality:

- Recording of up to 20 signals (inputs and outputs as well as internal variable values from the running control programm)
- Recording period up to 24h
- Graphic export of the recording as bitmap, JPEG, GIF, postscript, PDF, PCX, SVG
- Export of the individual values as text, HTML, XML or excel
- Import of stored records
- Automatic scaling
- Show or hide a legend
- View a statistic

- And much more...

11 Configuration- and Diagnosis-Software “Terminal”

The terminal program allows low-level configuration of the PLVC. For advanced user-friendly configuration and commissioning please use the PLVC VT software.

11.1 Introduction

The terminal chapter has been written for the different types of PLVC and their respective extensions (i.e. PLVC2, PLVC41 and PLVC8). Some displays and menus may therefore differ slightly on your PLVC or not be available. Other variations are outlined separately in this manual.

Our free terminal program HAWEE.TRM offers you a simple means to check with your PC (with serial interface RS-232) all internal states of the PLVC as well as apply parameter settings of the operational system.

You can also apply these via cell-phone with built-in modem (has to be enabled).

All inputs and outputs, their error messages and PLC setpoints are displayed in text form.

Overview

PLVC menus have a hierarchical tree-structure, as shown in figure 11.1.

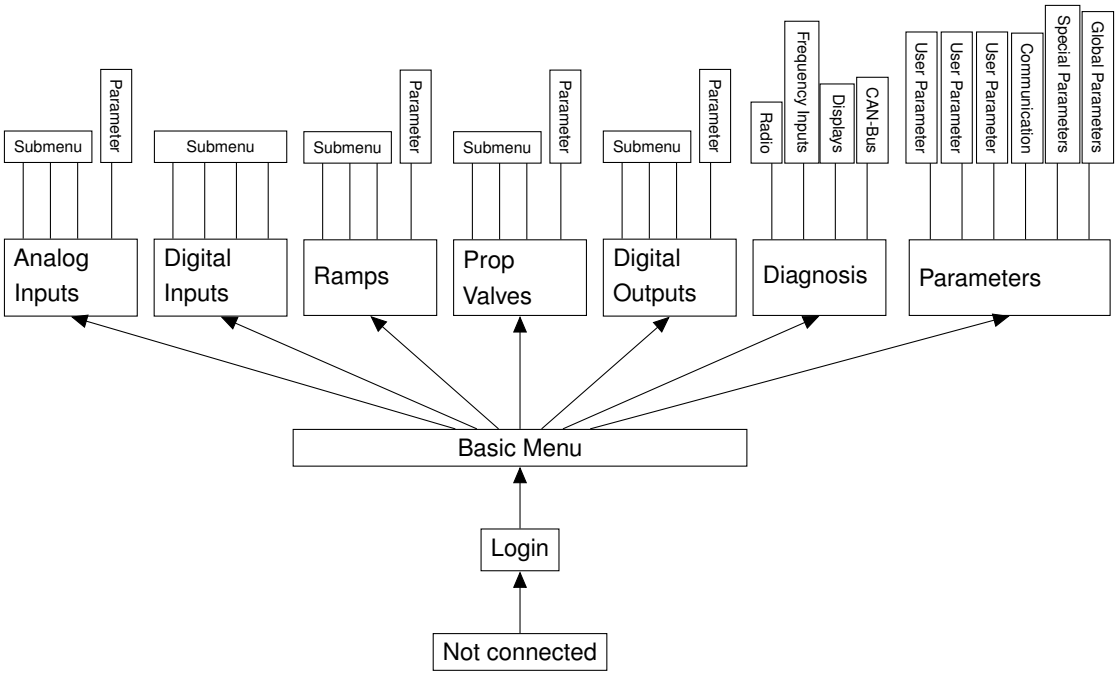


Figure 11.1.: Einleitung

Login will always return you to the basic menu. Click the **go back** button of the terminal software to return to the next lower level as shown to this illustration.

11.2 Login

Start the terminal program on a running PLVC and the terminal program displays an empty screen.

If you start the terminal program first and then the PLVC the screen in figure 11.2 is displayed.

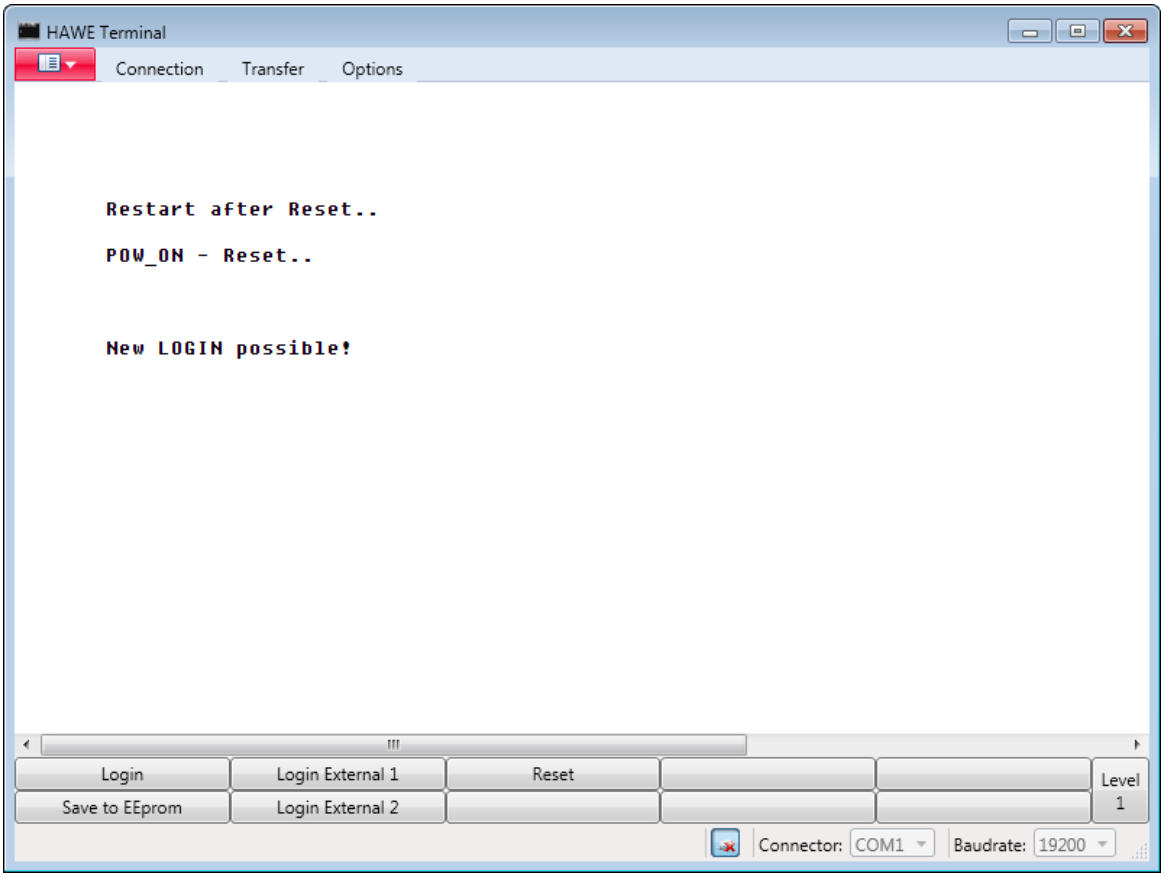


Figure 11.2.: Login

In both cases the lower section of the screen displays five upper and five lower buttons for mouse-click.

Login:	Login to local PLVC (establish connection)
Save to EEprom:	Save parameter changes permanently
Login External 1:	Login to first PLVC connected via CAN bus
Login External 2:	Login to second PLVC connected via CAN bus
Reset:	Reset local device

Click the **Login** button to access the basic menu. All other menus can be accessed from there.

11.2.1 Basic Menu

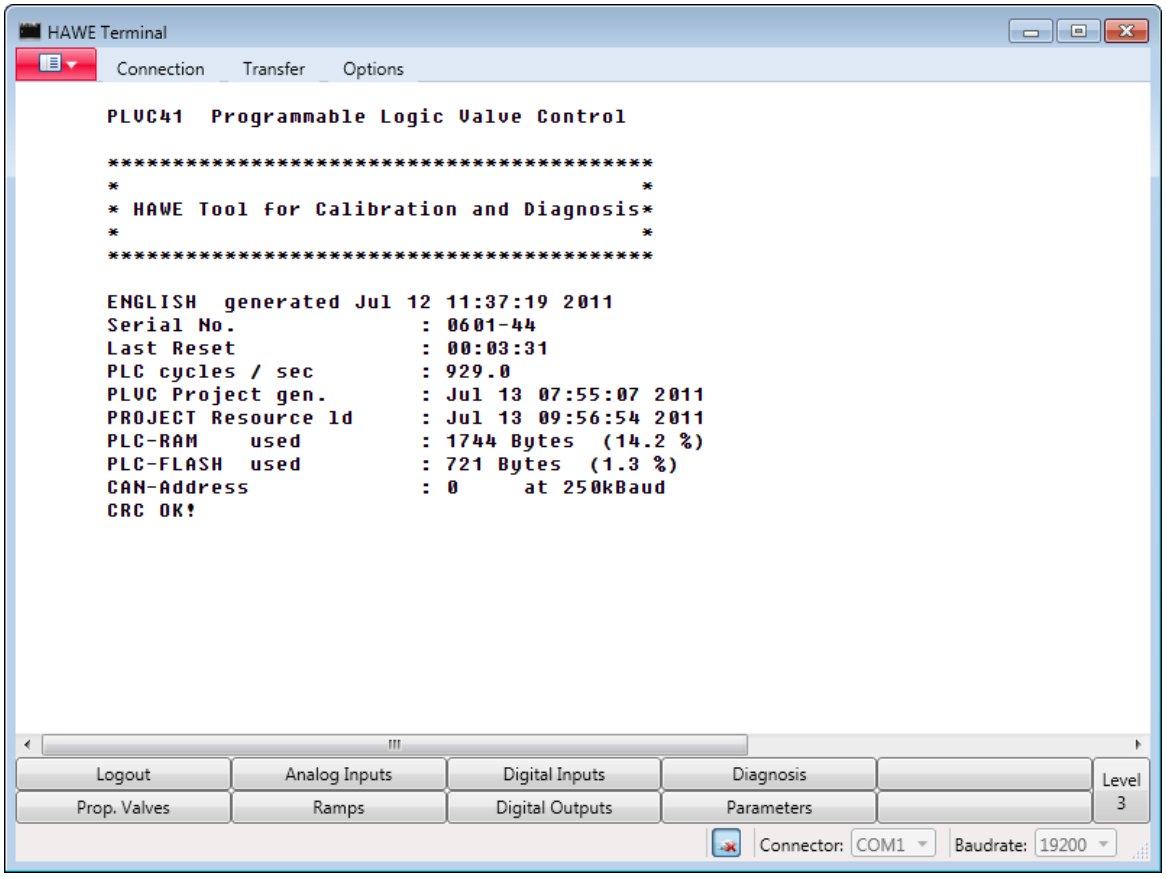


Figure 11.3.: Basic Menu

Figure 11.3 shows an overview of the most important data. PLVC2 will show in addition, whether profibus is installed (line 2), PLVC41 will show extensions in line 2. PLVC8 shows - additionally to the extensions - possible reasons for E-stop. Those will be explained in the following table.

EStop	E-stop input not supplied
C2?	Second controller not available, check software !
MainTr. off!	Main transistor is off by C2
MT1 not sup.	Main Transistor not supplied, check pin Y3
MT2 not sup.	Main Transistor not supplied, check pin A3
WD C2!	Controller 2 Watchdog reset
CRC C2!	Controller 2 CRC error, reinstall firmware C2
CRC PCS!	CRC of OpenPCS-Code wrong. Depends on Parameter \submenu5\c) > 1
QB2.3!	QB2.3, estop-out was set by openPCS or by OpenPCS-Error
PCS!	Openpcs not running (any more), probably timeout
SCT07!	Current measurement on input pins 0,2,4,6, although output not set. → short circuit? Detection can be deactivated via menu \porp-valve\submenu 8

Continued on the next page. . .

Continued from previous page. . .

SCT 8f	Current measurement on input pins 8,10,12,14, although output not set. → short circuit? Detection can be deactivated via menu \porp-valve\submenu 8
MT1 C-Prg!	Main transistor1 off because of C-program=safety.c
MT1 Err!	Main transistor1 off because of Error detected by Controler 2.. Probably one of the outputs 0..7 is supplied from outside
MT2 C-Prg	Main transistor2 off because of C-program=safety.c
MT2 Err	Main transistor2 off because of Error detected by Controler 2.. Probably one of the outputs 8..15 is supplied from outside
U_bat!	Power supply > 30V, please reduce
WD!	Main controller Watchdog reset.
STACK!	Stack error

Table 11.2.: Possible reasons for E-stop of PLVC8

Serial No.:	Serial number
Last Reset:	Time elapsed since last reset
PLC cycles / sec:	Number of PLC cycles per second

Two lines for the software names: Project name and resource name of PLC.

PLC-RAM used::	Number of bytes in RAM occupied by PLC-program
PLC-FLASH used:	Available memory that can be occupied by PLC-program
CAN-Address:	CAN-address. Identical to address inserted in **Parameters** → **Communication** for transmission of digital inputs

11.3 Proportional Valves

In the basic menu click the button **Prop. Valves** to access the settings and data on the proportional valves. The screen shown in figure 11.4 will be displayed.

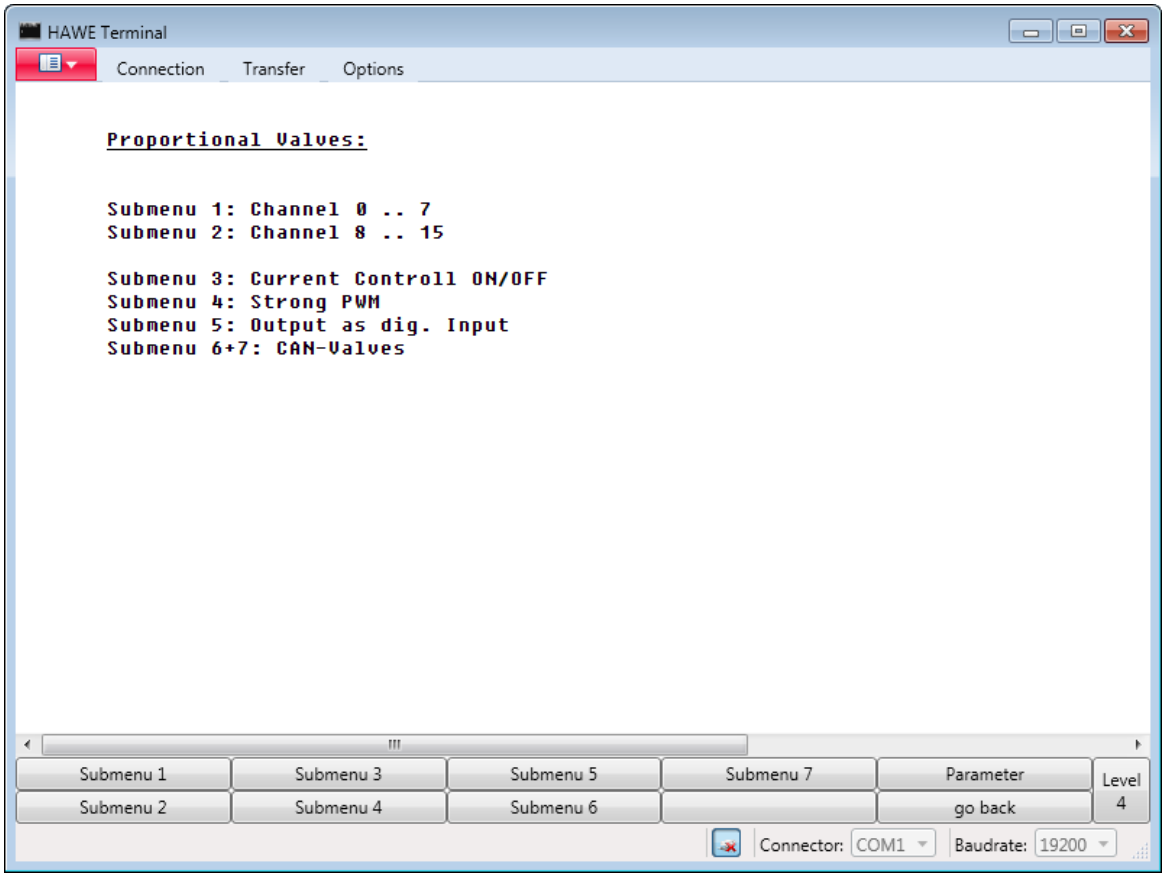


Figure 11.4.: Proportional Valves

You can activate up to 16 proportional valves. Depending on your PLVC-version these may be assigned to different channels. The data-sheet “Connectors and PLC-Addresses” that comes with your PLVC lists their exact assignment. The data-sheet also contains the logical addresses with which the individual channels can be activated through programming.

11.3.1 Proportional Valves Data

Simply click the corresponding **Submenu** button to display the data for the proportional valves and their coils (figure 11.5)

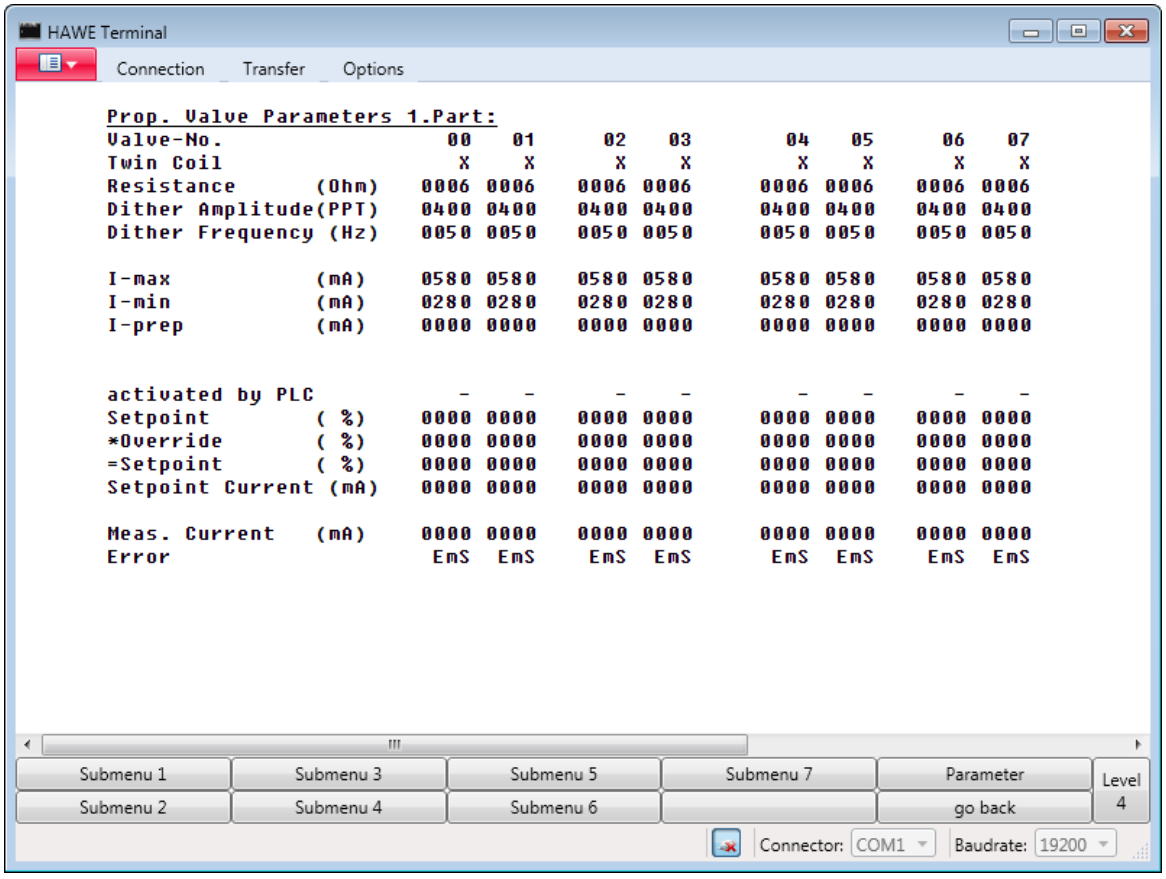


Figure 11.5.: Proportional Valves Data

The respective lines provide the following data:

Valve No.:	Number of valve. Identical with the channels selected on previous screen
Twin Coil:	In case this valve is operated with twin coils, it will be indicated by displaying an “X”
Resistance:	Preset resistance of coil in Ohm
Dither Amplitude:	Preset dither amplitude in ppt (parts per thousand)
Dither Frequency:	Preset dither frequency in Hertz. Increased amplitude and decreased frequency will cause stronger dither!
I-max:	Maximal current flow of this coil in mA at setpoint 100%
I-min:	Minimal current flow of this coil in mA at setpoint 1%
I-prep:	Current preset for testing purposes in Parameters screen.
Saturation:	(will only be shown when **Parameters** → **Submenu 7** → **A. Saturation** has been selected)
activated by PLC:	In case this output was activated by the PLC it will be indicated by displaying an “X” (Function ACT_VALVE).
Setpoint:	Designation of setpoints in percent (here 100% at valve 8)

Continued on the next page. . .

. . . Continued from previous Page

*Override:	Possible reduction of setpoint (here 100% of original setpoint at 100% = 100%)
*Saturation:	(will only be shown when **Parameters** → **Submenu 7** → **A. Saturation** has been selected)
+Closed Loop:	(will only be shown when **Parameters** → **Submenu 7** → **Closed Loop** has been selected)
=Setpoint:	Setpoint in percent after override computation, i.e. multiplication of setpoint and override.
Setpoint Current:	Current, which control wants to set in mA
Meas. Current:	Actual current flowing in mA
Error:	EmS for emergency stop, OK for in order, OPN for open (no coil connected), RNG for range error (current not accessible, supply voltage too low), SCT for short circuit to ground

11.3.2 Preset Proportional Valves

In one of the menus for the proportional valves click **Parameter**. The screen in figure 11.6 is displayed.

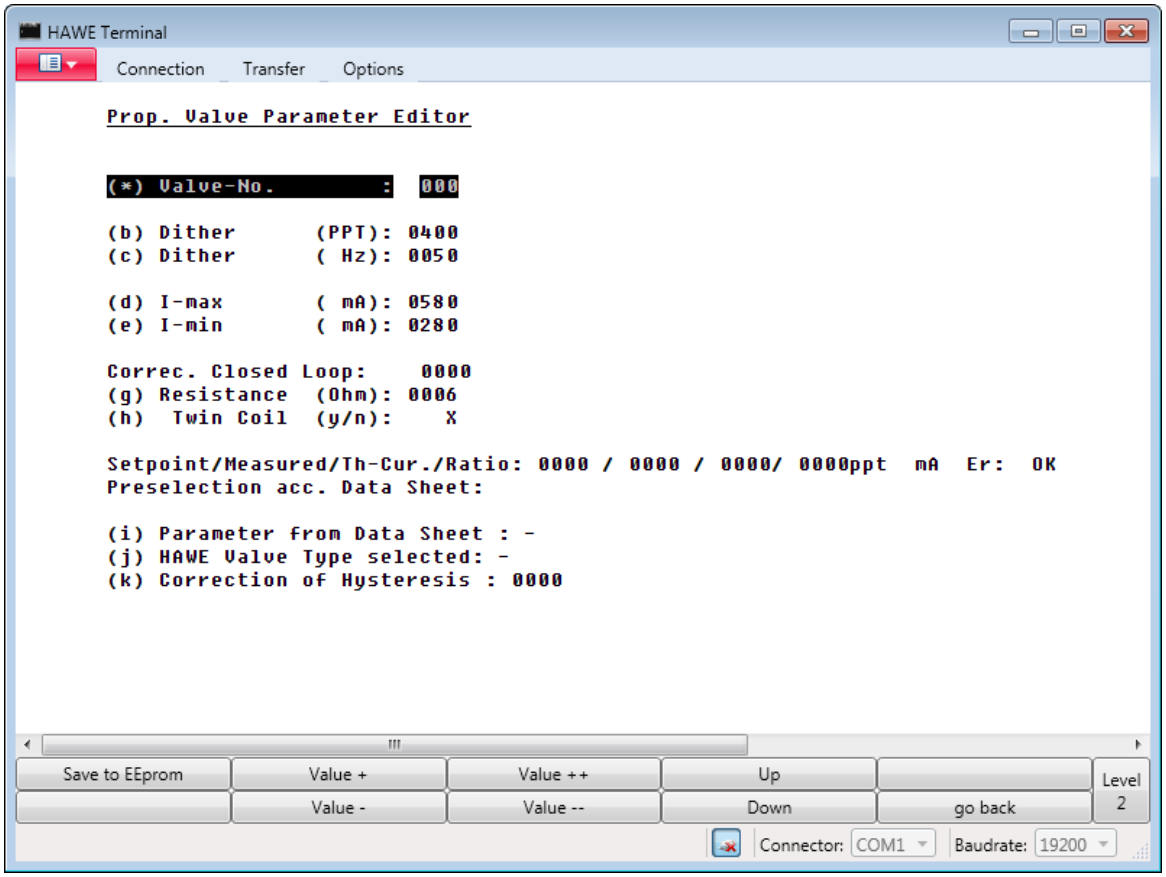


Figure 11.6.: Preset Proportional Valves

Preset the parameter values for the proportional valves in this screen dialog.

Toggle between the buttons **Up** and **Down** to access the various lines. The activated line is highlighted. Even faster: you can select the parameter that you want to change, by typing the letter in brackets before the parameter with your keyboard.

Use the buttons **Value +** and **Value ++** to increase the highlighted value in increments of 1 or multiples. Use the buttons **Value -** and **Value –** to decrease the highlighted value in increments of 1 or multiples.

You can also directly insert the value via your keyboard: type a “v” in the end to change toggle to negative values, type a “w” to refresh the screen and restart insertion of numbers.

The adjusted values will instantly become effective but are deleted at next reset if not saved to EEprom.

Save to EEprom will save your settings permanently in the device’s EEPROM.

The respective values displayed are listed in table 11.6.

Valve No.:	Valve selected. Identical with the channels selected on previous screen.
Dither (PPT):	Dither amplitude in ppt from 0 to 480
Dither (Hz):	Dither frequency in Hertz. Values range from 25 to 250 Hz. Due to the internal processing of the values in the PLVC not allindividual increments can be set in the range between 25-250 Hz.
I-max:	Maximal current flow of this valve in mA at setpoint 100%, 60-2200 mA
I-min:	Minimum current in mA
Resistance:	Resistance of coil in Ohm from 2 to 36
Twin Coil:	Activate this parameter to switch valve as single or twin valve. If switched as twin valve, it will be indicated by displaying an “X”.
Setpoint/Measured/Th-Cur./Ratio:	Display of current values setpoint/actual current
Parameter from Data Sheet:	Click the Value + button to select parameters of a selected HAWE valve type.
HAWE Valve Type selected:	Click Value + and/or VALUE - buttons to select from various different HAWE valve types.
Correction of Hysteresis:	Preset a correction value for the hysteresis in units of 2,5mA for this line. This hysteresis correction is used when the current is reduced or ramped down.

Make sure to save all your changes. Click **go back** to return to the proportional valve menu from where you can return to the basic menu by clicking **go back** a second time.

11.4 Analog Inputs

Click the **Analog Inputs** button in the basic menu to access settings and data for the analog inputs. The screen in figure 11.7 will be displayed.

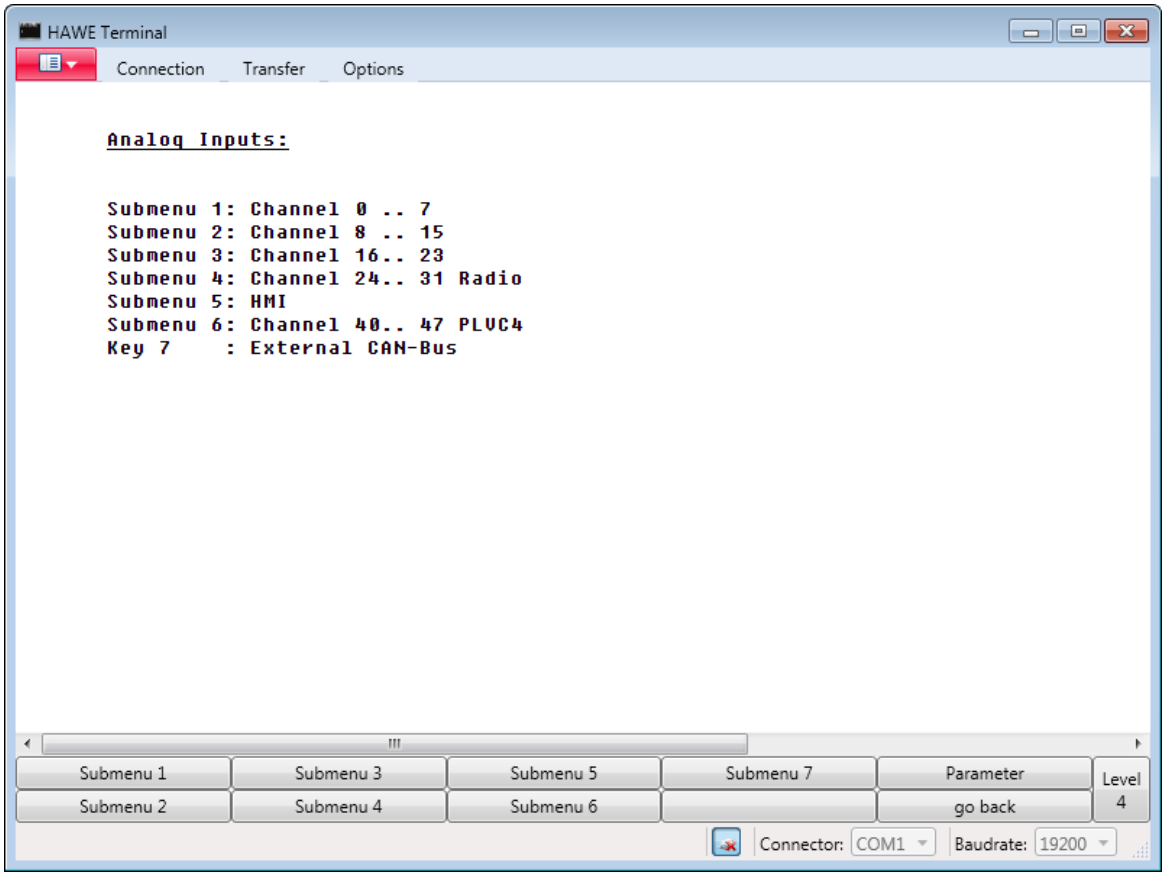


Figure 11.7.: Analog Inputs

Analog inputs of the local PLVC (channel 0-23) can be seen in submenus 1 to 3.

Analog inputs from radio commands can be seen in submenu 4, additional analog inputs can come from the analog CAN nodes 11-14 or from external PLVCs (see submenu 7). Inputs of max. 2 additional PLVCs can be addressed. These must be specified in the menu **Communication**. The addresses used for scanning the inputs via PLC are listed on the data-sheet “Connectors and PLC-Addresses” of your PLVC.

11.4.1 Analog Inputs Data

Simply click the corresponding **Submenu** button to display the data on the inputs (figure 11.8).

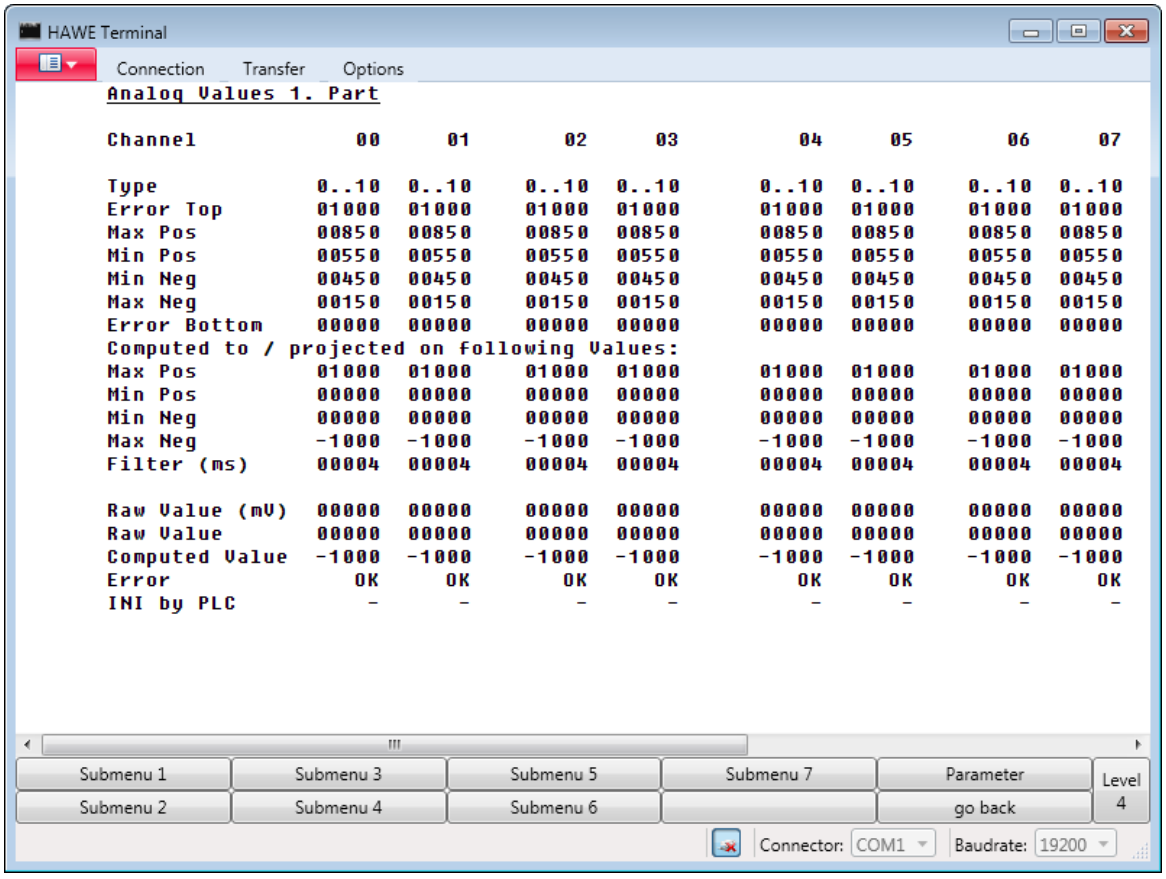


Figure 11.8.: Analog Inputs Data

The respective lines provide the following data:

Channel:	The channel selected
Type:	The input range selected. Absolute ranges are from: 0-10V, 4-20mA , angle or those read relative to the supply voltage (ratiometric): POT = potentiometer, joystick (see description below)
Error Top:	If this value is exceeded, error message (RMX) is displayed and the computed value returns the value 0.
Max Pos:	Positive maximum value to be read-in
Min Pos:	Positive minimum value to be read-in
Min Neg:	Negative minimum value to be read-in
Max Neg:	Negative maximum value to be read-in
Error Bottom:	If the raw value is below this value, error message (RMN) is displayed and the computed value returns the value 0.
Computed to / projected on following Values:	Up to now the actual measured raw values were treated; now the values on which the variables are mapped
Max Pos:	Positive maximal value, on which above max pos is computed
Min Pos:	Positive minimal value used for computation
Min Neg:	Negative minimal value, on which above max neg is computed

Continued on the next page. . .

. . . Continued from previous Page

Max Neg:	Negative maximal value used for computation
Filter:	Filter time constant: Returns strength of the digital filter. Analog value has a digital low-pass of 1st order. Time constant is within 1/100 sec. The values are smoothed.
Raw Value (mV):	Value read-in in mV
Raw Value:	Value read-in
Computed Value:	Value computed/scaled (see below)
Error:	OK for in order; RMX for upper error, RMN for lower error
INI by PLC:	If this channel was activated by PLC-program (Function ANA_INI), it will be indicated by displaying an “X”.

11.4.2 Preset Analog Inputs

Short explanation of the various types of analog inputs:

Joystick:

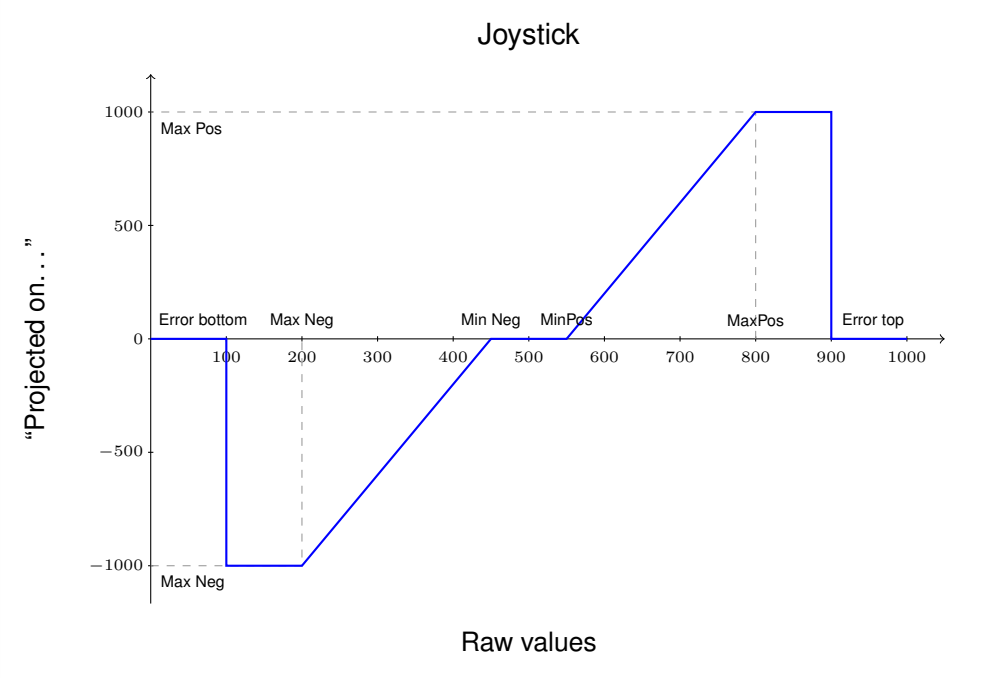


Figure 11.9.: Preset of analog inputs: Joystick

In the middle of the joystick-type is a dead center (figure 11.9). This is preset through the values inserted in MinNeg and MinPos. The return value in between is always 0.

The following illustration shows the return value based on the raw values. The values MaxNeg and MaxPos on both axis' determine the line's progress. Outside the error limits the value 0 is always returned.

The analog input of the type Joystick is ratiometric.

Potentiometer:

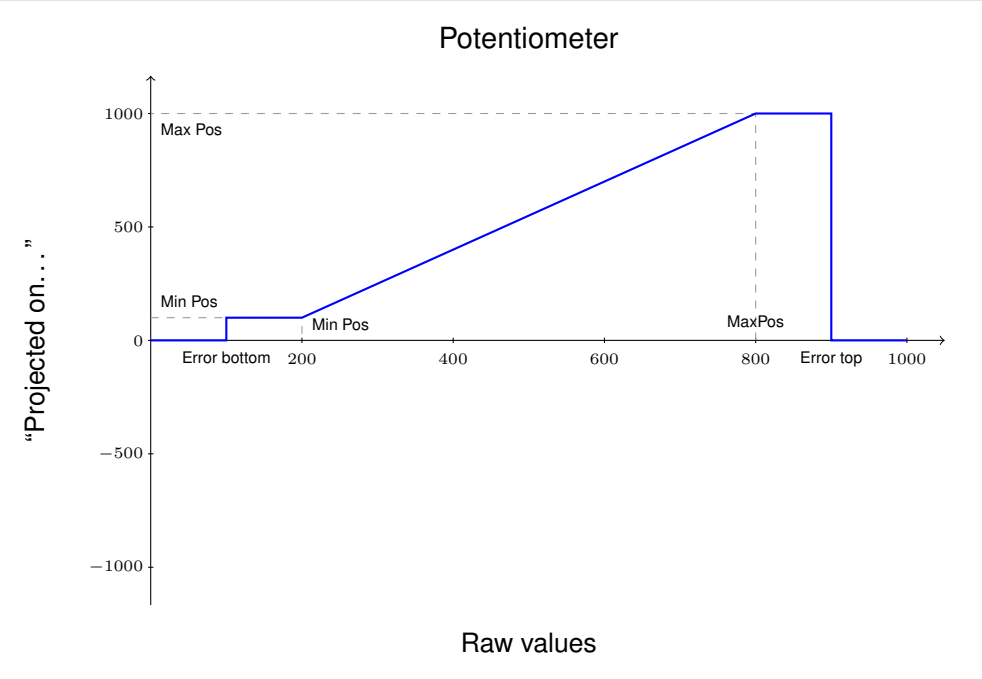


Figure 11.10.: Preset of analog inputs: Potentiometer

Type potentiometer is relatively similar to type joystick but has no negative range (figure 11.10).

Values MinPos and MaxPos in the section "Projected on" are here also allocated along the y-axis. In this case MinPos stands at 100 and MaxPos at 1000.

The analog input of the type Potentiometer is ratiometric.

Angle Sensor:

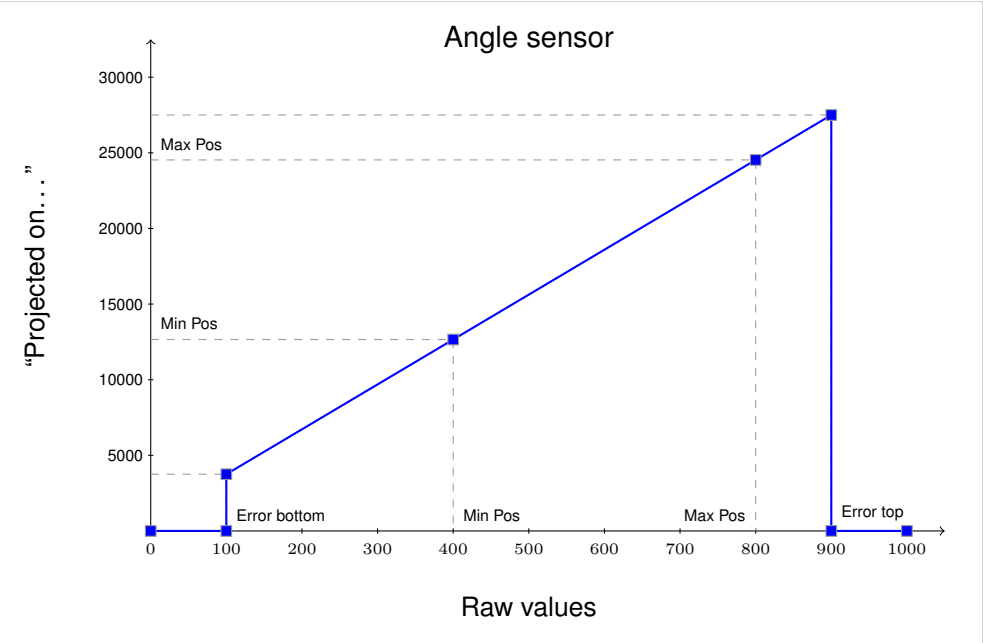


Figure 11.11.: Preset of analog inputs: Winkelgeber

Angle is similar to poti, potentiometer (figure 11.11). The values of MinPos and MaxPos are unlimited here though. The value pairs (MinPos/MinPos) and (MaxPos/MaxPos) therefore need not be ultimate values. This is particularly convenient with sensors, whose ultimate values are very difficult to measure.

Click **Parameter** in one of the menus for the analog inputs. The screen in figure 11.12 will be displayed.

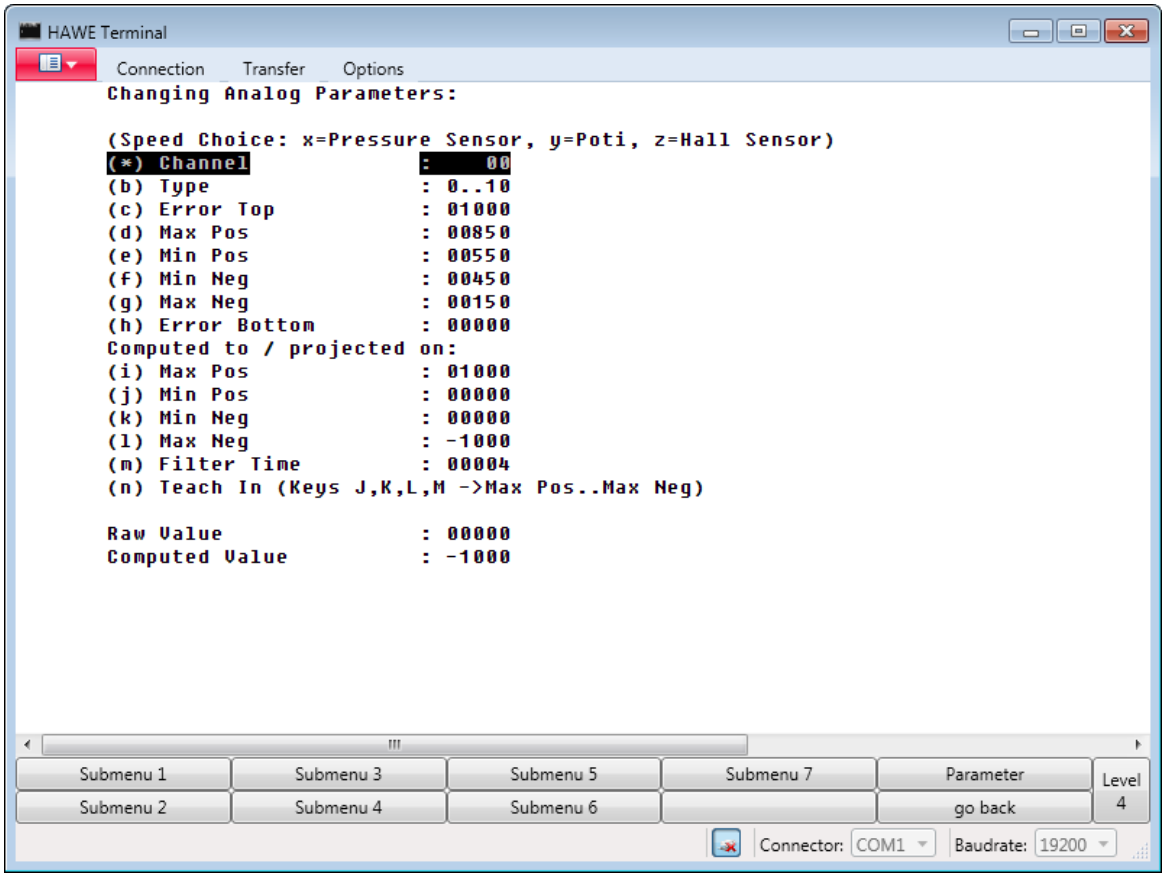


Figure 11.12.: Preset Analog Inputs

Toggle between the buttons **Up** and **Down** to access the various lines. The activated line is highlighted. Even faster: you can select the parameter that you want to change, by typing the letter in brackets before the parameter with your keyboard.

Use the buttons **Value +** and **Value ++** to increase the highlighted value in increments of 1 or multiples. Use the buttons **Value -** and **Value –** to decrease the highlighted value in increments of 1 or multiples.

You can also directly insert the value via your keyboard: type a “v” in the end to change toggle to negative values, type a “w” to refresh the screen and restart insertion of numbers. Also see chapter Advanced User Guide (11.10).

The adjusted values will instantly become effective but are deleted at next reset if not saved to EEprom.

Save to EEprom will save your settings permanently in the device’s EEPROM.

Be aware, that with PLVC8 all raw values go from 0 ..10.000 instead of 0..1.000.

With PLVC8 analog inputs 8..11 and 40..43 can be switched to 4..20mA via Parameter, instead of having to solder some jumpers.
With PLVC21 analog inputs 0..3 have same property.

The respective values displayed are (table 11.8):

Channel:	Input channel
TYPE*:	Input range. Between 0-10V, 4-20mA, potentiometer, joystick or angle.
Error Top:	Enter threshold value, which triggers error message RMX (Upper Error) and which will return value 0. In case this value is 1000, error RMX will never occur.
Max Pos:	Positive maximum value to be read-in
Min Pos:	Positive minimum value to be read-in
Min Neg:	Negative minimum value to be read-in
Max Neg:	Negative maximum value to be read-in
Error Bottom:	Threshold value, which triggers error message RMN (lower error) and which will return value 0. In case this value is 0, error RMN will never occur.
Computed to / projected on:	Displayed on:
Max Pos:	Positive maximum value for computation (except for angle)
Min Pos:	Positive minimum value for computation (except for angle)
Min Neg:	Negative minimum value for computation
Max Neg:	Negative maximum value for computation
Filter Time:	Returns strength of the digital filter
Teach In:	Teach-in mode can also be used to read-in maximum and minimum values. For this the joystick or potentiometer must be fully deflected in the different directions and the corresponding key must be triggered (e.g.keyboard-key “J” for maximum value of positive deflection)
Raw Value:	Value read-in
Computed Value:	Value computed via parameters above. Value ranges for raw values extend from 0 to 1000. Illustrations cover a range from -1000 to 1000 (resp. 0 for type angle)

*New possible TYPE-Values:

- 5V A : configured to read 5V inputs at 470kOhm Resistance. Value interpreted as absolute value (not Like POTI or JOY). Note that the PLVC8 has an option to remove a voltage divider by software, so 5V will not return 5,000 but 10,000.
- 5V Rt: configured to read 5V inputs at 470kOhm Resistance. Value interpreted as relative value (like POTI or JOY), ratiometric to Sensor Voltage. Note that the PLVC8 has an option to remove a voltage divider by software, so 5V will not return 5,000 but 10,000.
- 12V Rt: configured to read 12V inputs. Value is interpreted as relative value. If the supply is 12V and the input reads 3V it shows 3/12*1000=250. If the voltage then drops to 10, value will increase to 3/10*1000=300

- 24V Rt: configured to read 24V inputs. Value is interpreted as relative value. If the supply is 24V and the input reads 3V it shows 3/24*1000=125. If the voltage then drops to 12, value will increase to 3/12*1000=250
- 20mA : configured to read 0..20mA inputs at 220Ohm Resistance. Value interpreted as absolute value
- 20mAJ : for future use

If Voltage applied to 20mA - Input is too high, input is switched back to 0..10V-Mode to protect 220Ohm resistance, which would otherwise be destroyed. If this is detected (for example by supplying more than 10V) the Error 4mA OVL will be shown in terminal and the calculated value will become zero.

Only PLVC8:
For analog inputs 12, 13, 14, setting TYPE to 20mA will result in switching the input to Resistor value of 7KOhm, making it more suitable for use as digital input.

Make sure to save all your changes. Click **go back** to return to the analog input menu, from where you can return to the basic menu by clicking **go back** a second time.

11.4.3 Analog Inputs CAN

Using Menu AnalogInputs (figure 11.7) submenu 7, 8, 9, A, B or C (by pressing key on your keyboard) the values of CAN telegrams 181_{hex} bis $1F8_{hex}$ und 281_{hex} bis $2F8_{hex}$ are displayed.

Refer to the note at the beginning of chapter 13.7 please!

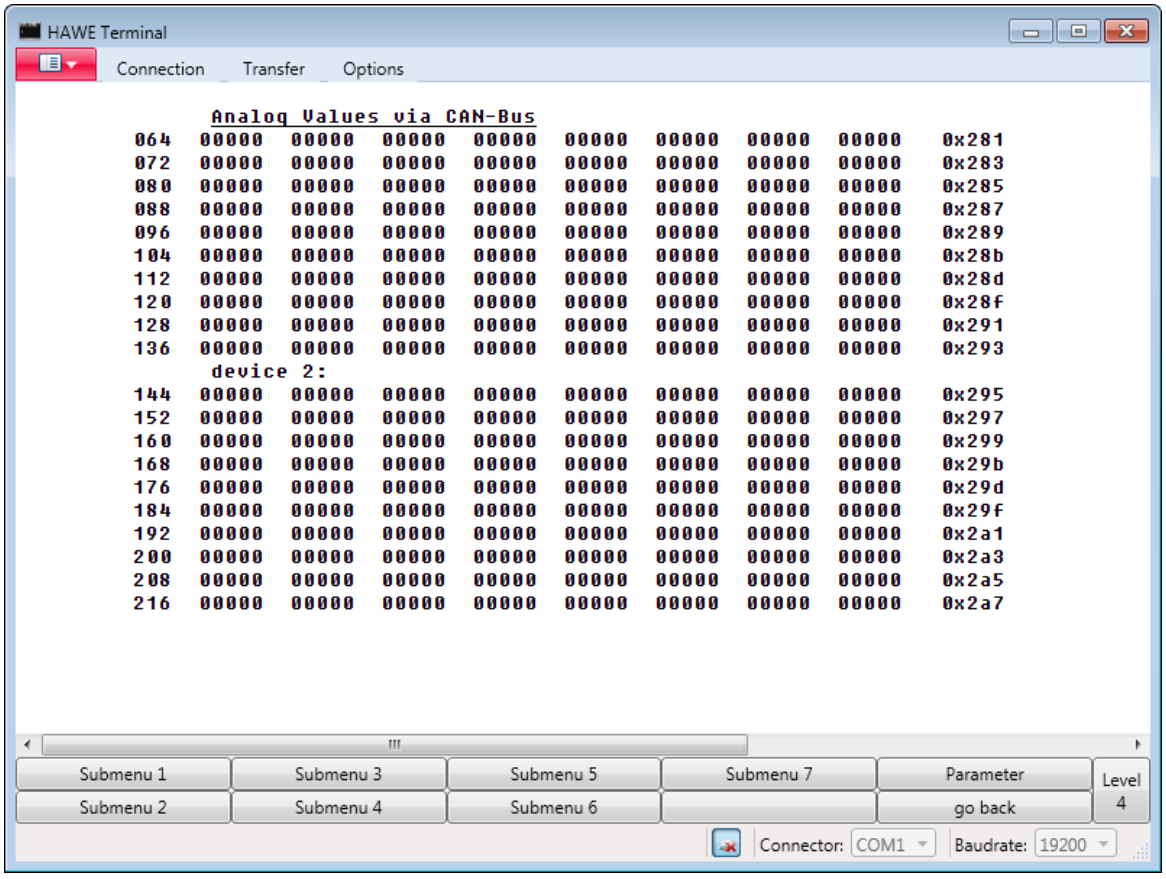


Figure 11.13.: Preset Analog Inputs

The following figure (11.14) shows some ditails of AnalogInput submenu 8.

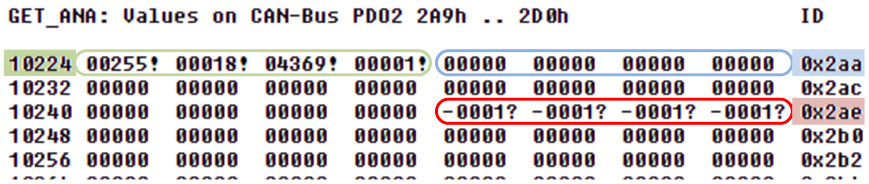


Figure 11.14.: Analog Inputs Submenu 8

The shown values are 16 bit long signed integer. Each number presents the content of two bytes from an 8-byte telegram.

The color-marked areas means:

- blue - CAN ID $2AA_{hex}$ - no telegram has been received. All Values in this Telegram are shown as 0
- green - CAN ID $2A9_{hex}$ - telegrams has been received. The result of byte 0 and byte 1 together is 255. The result of byte 2 and byte 3 together is 18.... The exclamation mark means: The telegram is ok. At least 1 telegram within last 200ms.

- red - CAN ID $2AE_{hex}$ - telegrams were received. But there was no telegram within last 200ms. Timeout. All values are set to -1.

For more informations about these data and how to use it read dokumentation of function module GET_ANA in chapter 12.6.4 please.

11.5 Ramps

Ramps shall prevent jumps in the input values and proportional valve flows. This enables smooth start-up operations of cylinders even if the joystick movements are jerky.

In the basic menu click the button **Ramps** to access the settings and data for the ramps. The screen in figure 11.15 will be displayed:

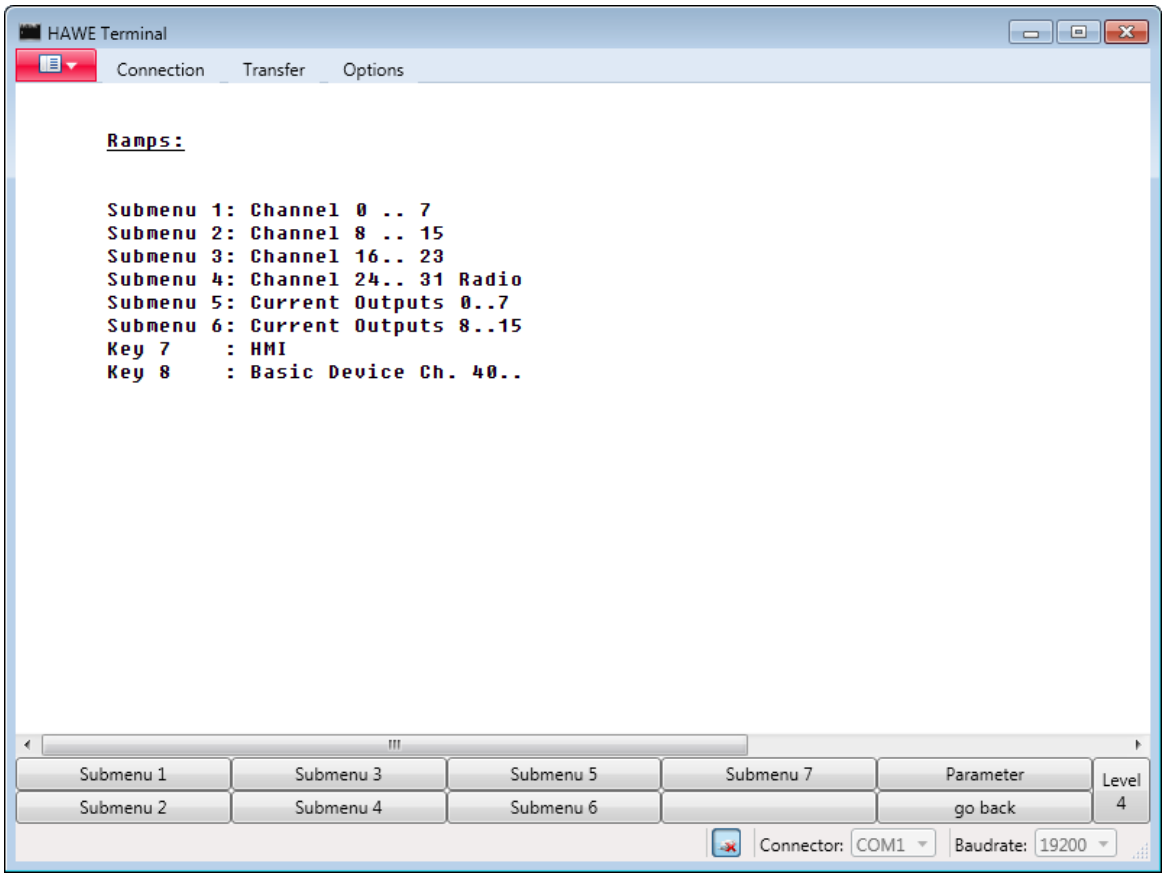


Figure 11.15.: Ramps

Submenus 1-4 are allocated to the analog input values, both local and by radio telegram. Submenus 5-6 are allocated to ramps for electrical outputs (ramps for proportional valves are on channels 32-47). These ramps can convert digital input signals into output signals with smooth increasing or decreasing edges. Buttons 8 and 9 access menus for the values received from the analog CAN nodes.

11.5.1 Ramps Data

Simply click the corresponding **Submenu** button to display the data for the ramps (figure 11.16). For submenus 7 and 8 enter the numbers “7” and “8” respectively. All submenus share the same structure.

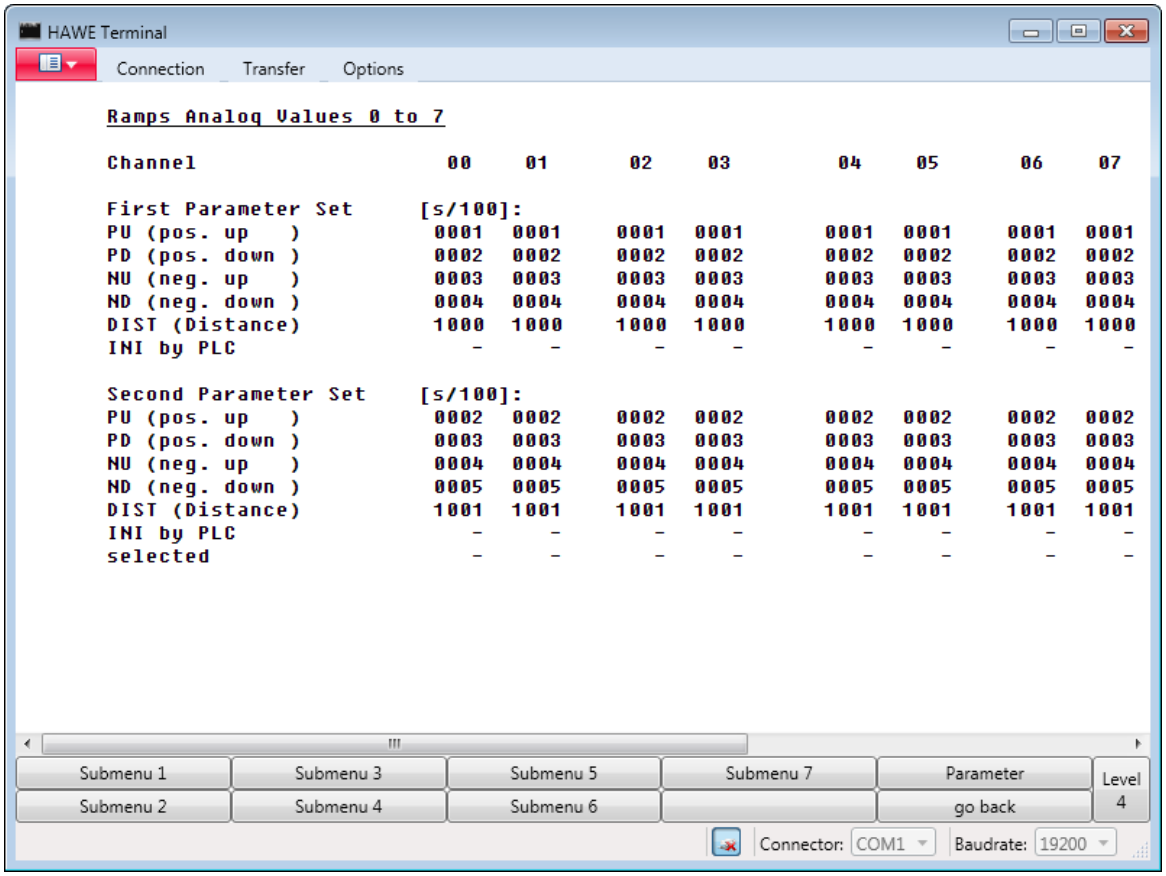


Figure 11.16.: Ramps Data

The respective lines provide the following data (table 11.9):

Channel:	The channel selected
First Parameter Set:	Two independent sets are available:
PU:	“Positive up” = Ramp time during acceleration in positive direction in 1/100 sec
PD:	“Positive down” = Ramp time during deceleration from positive direction in 1/100 sec
NU:	“Negative up” = Ramp time during acceleration in negative direction in 1/100 sec
ND:	“Negative down” = Ramp time during deceleration from negative direction in 1/100 sec

Dist:	Distance to which time refers (generally 1000). Screen example shows PU1 at 220 (i.e. 2,2 sec). The value 1000 at dist means that the ramp time of 2,2 sec stretches over the entire route. If dist were to return value 100, ramp time would cover merely a 1/10 of the route. The ramp would then run off 10 times slower.
INI by PLC:	In case this channel was activated by the PLC, it will be indicated by displaying an “X”
Second Parameter Set:	See first parameter set.
Selected:	In case the second parameter set was activated for this channel, it will be indicated by displaying an “X”

11.5.2 Preset Ramps

Click **Parameter** in one of the menus for the Ramps. The screen in figure 11.17 will be displayed.

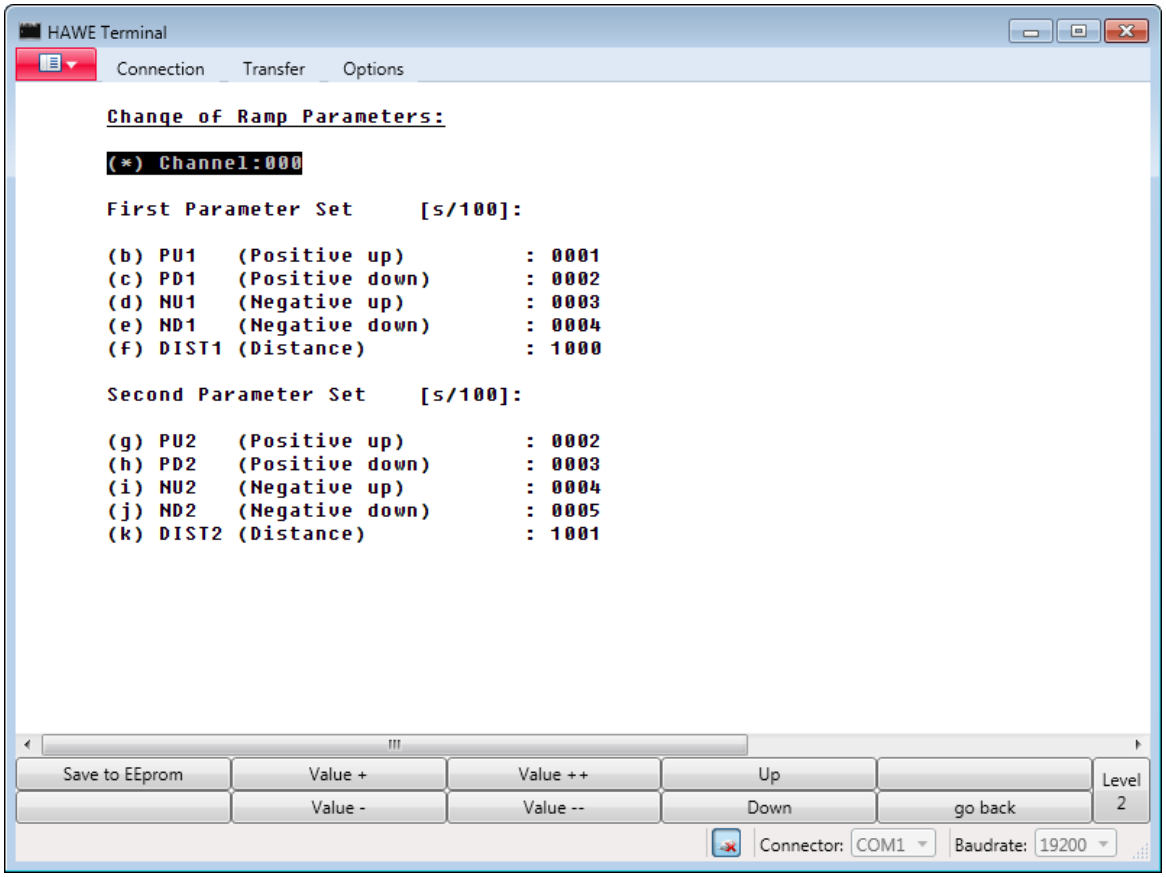


Figure 11.17.: Preset Ramps

Toggle between the buttons **Up** and **Down** to access the various lines. The activated line is highlighted. Even faster: you can select the parameter that you want to change, by typing the letter in brackets before the parameter with your keyboard.

Use the buttons **Value +** and **Value ++** to increase the highlighted value in increments of 1 or multiples.
Use the buttons **Value -** and **Value –** to decrease the highlighted value in increments of 1 or multiples.

You can also directly insert the value via your keyboard: type a “v” in the end to change toggle to negative values, type a “w” to refresh the screen and restart insertion of numbers.

The adjusted values will instantly become effective but are deleted at next reset if not saved to EEprom.

Save to EEprom will save your settings permanently in the device’s EEPROM.

The respective values displayed are (table 11.10)

Channel:	Select the channel for input.
First parameter set:	
PU:	“Positive up” = Ramp time during acceleration in positive direction in 1/100 sec
PD:	“Positive down” = Ramp time during deceleration from positive direction in 1/100 sec
NU:	“Negative up” = Ramp time during acceleration in negative direction in 1/100 sec
ND:	“Negative down” = Ramp time during deceleration from negative direction in 1/100 sec
Dist:	Distance to which time refers (generally 1000).
INI by PLC:	In case this channel was activated by the PLC, it will be indicated by displaying an “X”.
Second parameter set:	See first parameter set.
Selected:	In case the second parameter set was activated for this channel, it will be indicated by displaying an “X”.

Times can be set from 0 hundredth seconds to 320 seconds.

All analog In- and Outputs have adjustable ramps, each of them with two selectable parameter sets. It is possible to toggle smoothly between two complete parameter sets. For reasons of simplification the second parameter set for analog values is chosen by “digital outputs”:

%QB17.0 . . . %QB17.7 for ramps **%IW24.0** to **%IW38.0** (Basic device)

%QB18.0 . . . %QB19.7 for ramps **%IW40.0** to **%IW70.0** (Extension)

%QB20.0 . . . %QB20.7 for ramps **%IW72.0** to **%IW86.0** (Radio)

%QB21.0 . . . %QB22.7 for ramps **32 . . . 47** (current outputs 0 . . . 15)

%QB23.0 . . . %QB24.7 for ramps **%IW88.0** to **%IW118.0** (analog CAN nodes)

Make sure to save all your changes. Click **go back** to return to the ramps menu from where you can return to the basic menu by clicking **go back** a second time.

11.6 Digital Inputs

In the basic menu click the button **Digital Inputs** to access the screen in figure 11.18.

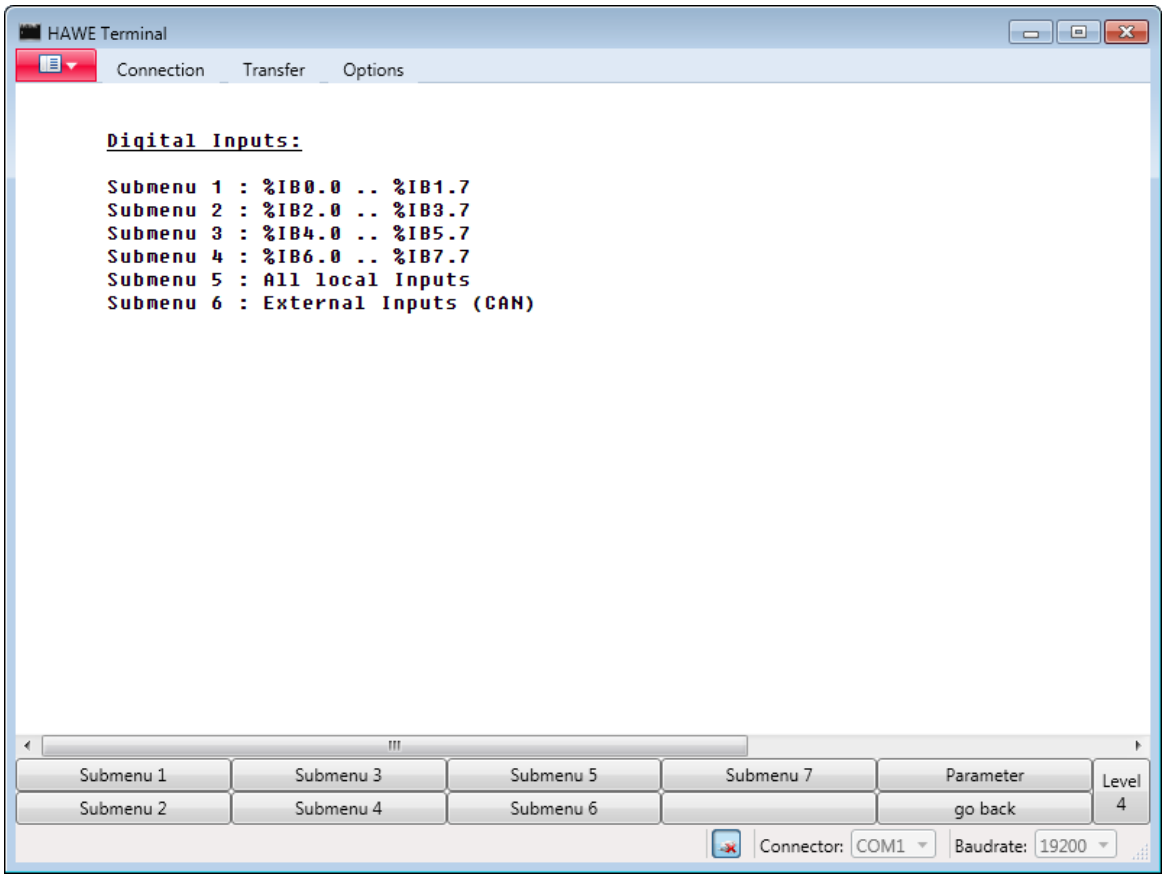


Figure 11.18.: Digital Inputs

The various **Submenu** buttons provide detailed information on the inputs selected (figure 11.19):

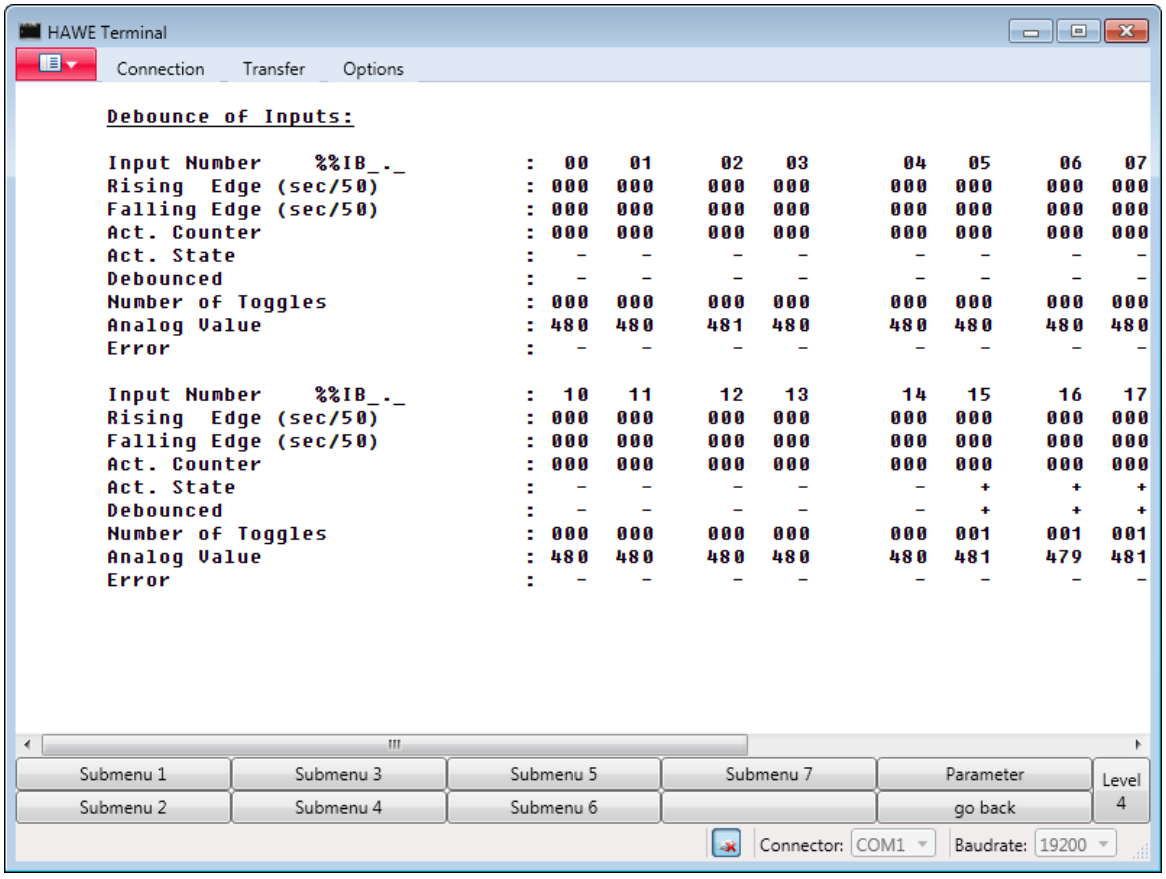


Figure 11.19.: Digital Inputs

The respective lines provide the following data (table 11.11):

Input Number:	Address of the digital channel
Rising Edge:	De-bounce time on activation
Falling Edge:	De-bounce time on de-activation
Act. Counter:	Number of cycles (de-bounce time) already processed
Act. State:	Actual state of input
Debounced:	Actual de-bounced state of input
Number of Toggles:	Number of times this input was toggled (since reset)
Analog Value:	Re-read analog value of digital input (auxiliary value)
Error:	Indicates an error. Analog values of digital inputs are monitored. Non-defined analog values result in error messages.

Submenu 5 displays a condensed overview of all digital inputs (figure 11.20).

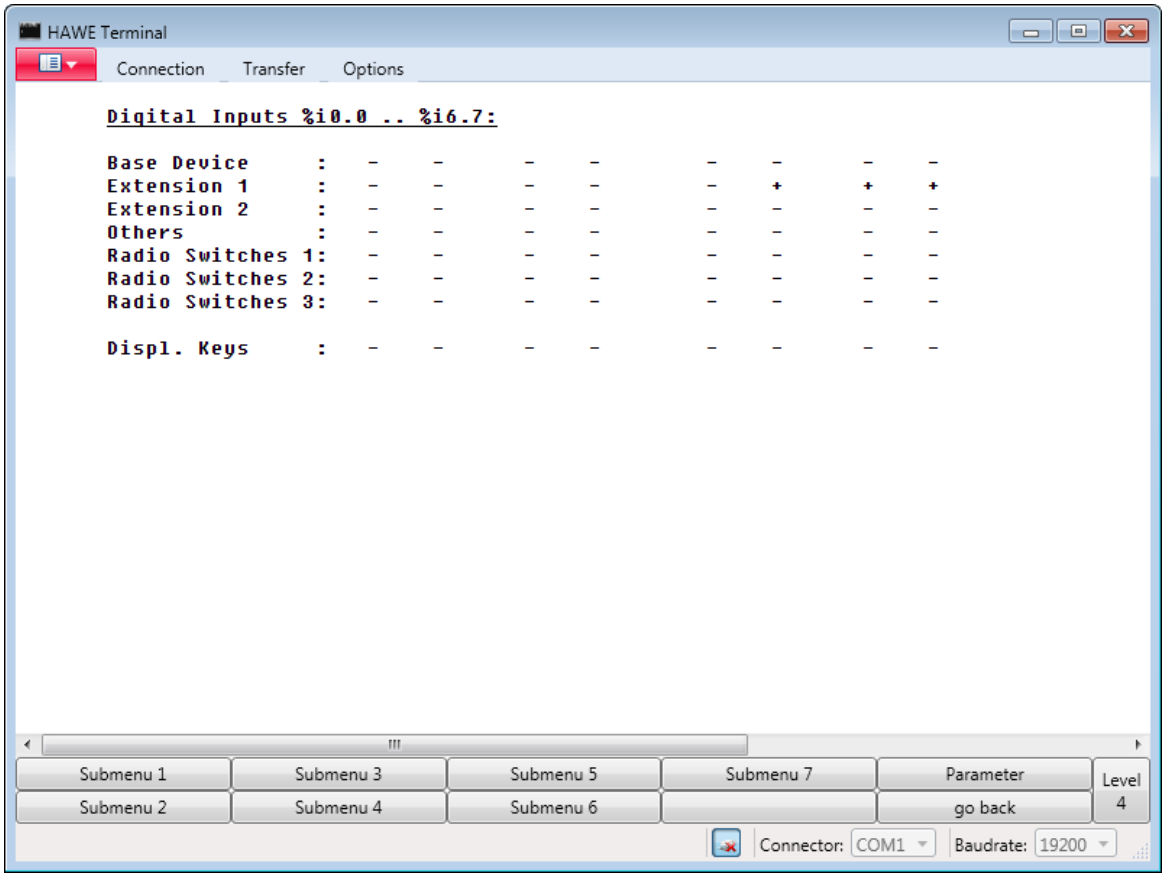


Figure 11.20.: Digital Inputs

Every activated input is prefaced with “+”.

Submenu 6 displays the states of the external digital inputs on the PLVCs connected via CAN bus (figure 11.21).

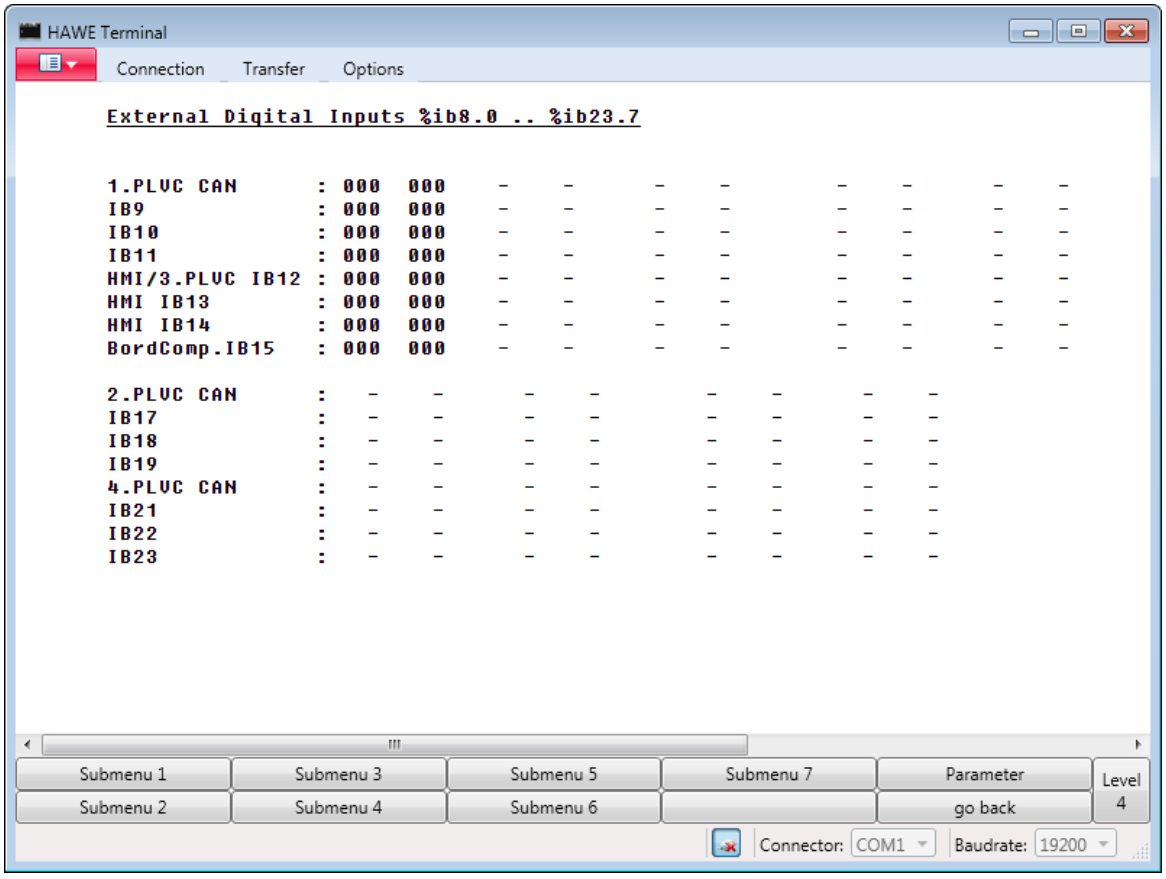


Figure 11.21.: Digital Inputs

11.7 Digital Outputs

Click the **Digital Outputs** button in the basic menu to access settings and data for the digital outputs.

The following menu is displayed (figure 11.22):

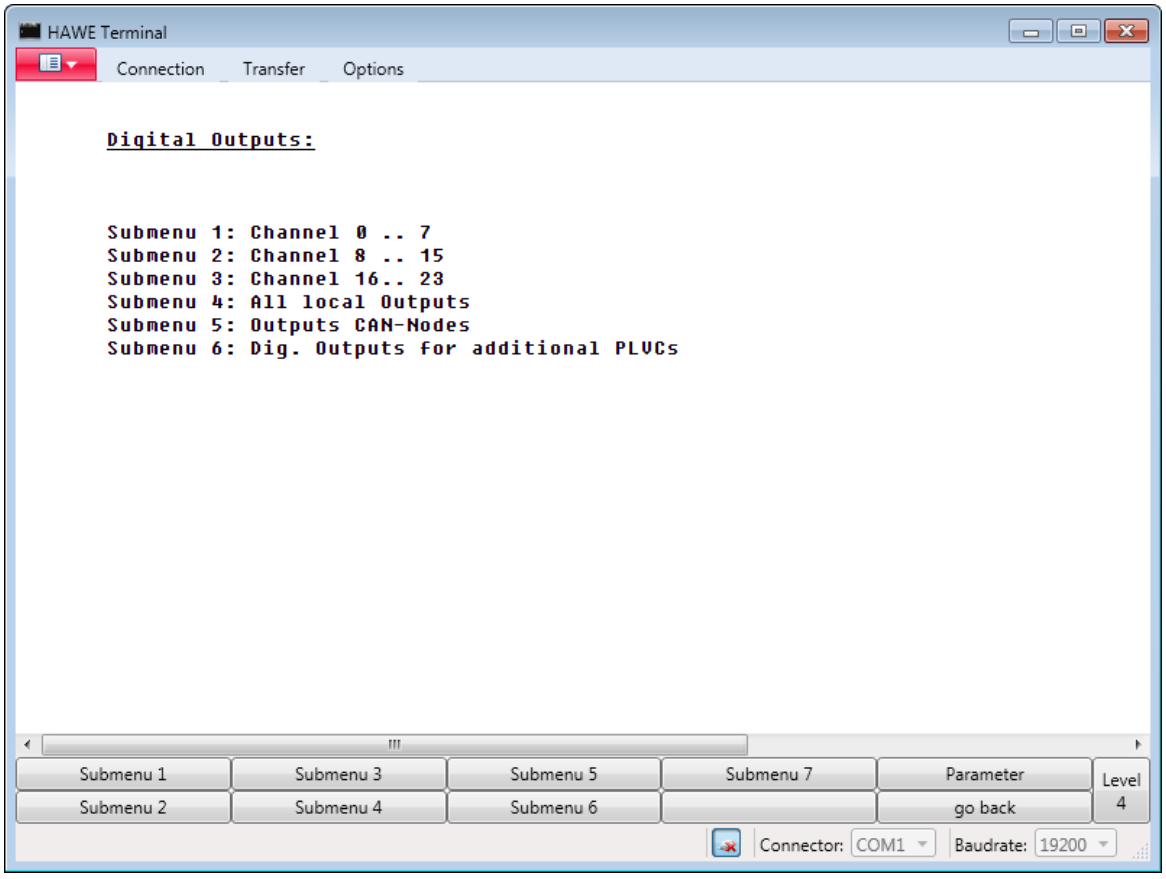


Figure 11.22.: Digital Outputs

Submenu 1-4 are local outputs.

Submenu 5 is allocated to devices connected via CAN bus.

Submenu 6 is allocated to additional PLVCs connected via CAN bus.

11.7.1 Digital Outputs (PWM) Data

Simply click the corresponding **Submenu** button to display the data for the PWM-outputs.

Submenu 1-2 is shown in figure 11.23:

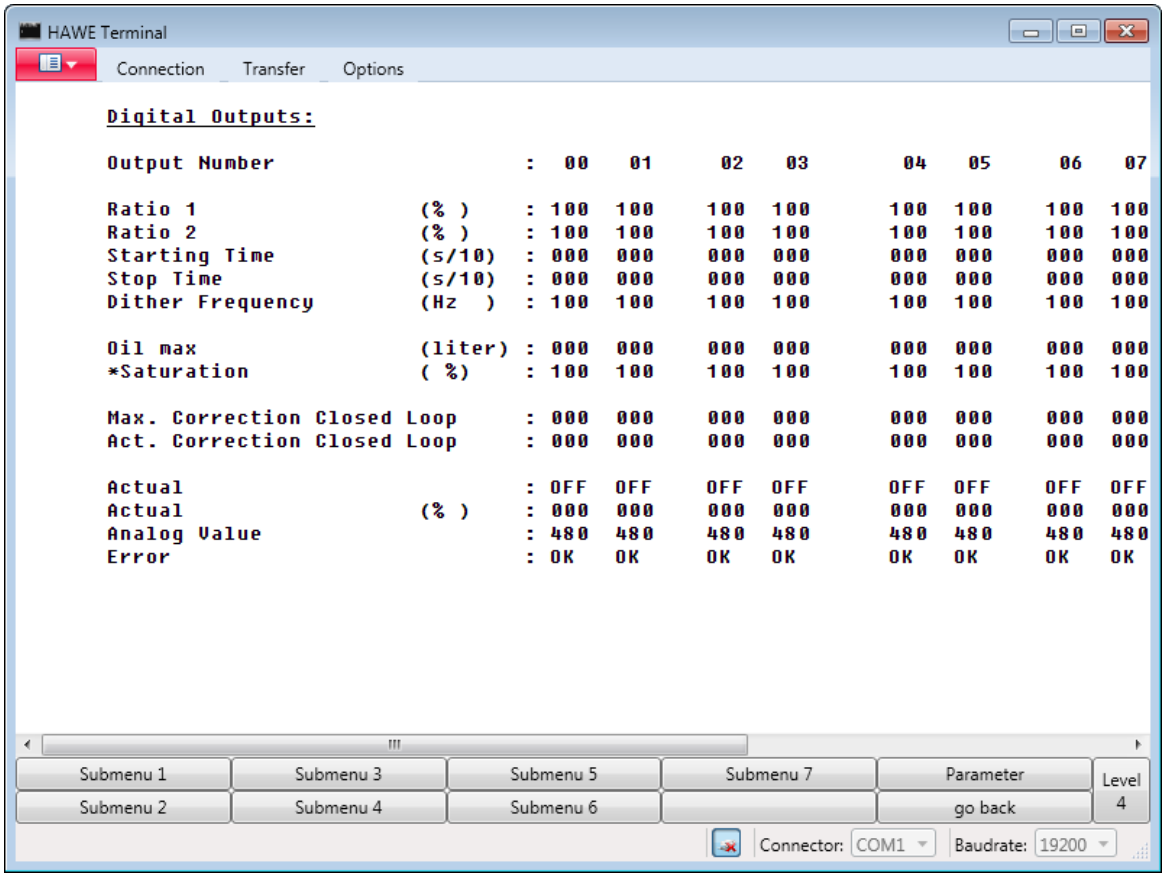


Figure 11.23.: Digital Outputs (PWM) Data

The respective lines provide the following data (table 11.12):

Output Number:	Channel of digital output
Ratio 1:	Power-output in percent produced in state 1, when started
Ratio 2:	Power-output in percent produced in state 2, reached after starting time
Starting Time:	Time until state 2 is reached (in 1/10 sec)
Stop Time:	Time-delay until switch-off (see image 11.24) in 1/10 sec
Dither Frequency:	Dither frequency in Hz, (50, 100 or 200)
Oil max:	(will only be shown when **Parameters** → **Submenu 7** → **A. Saturation** has been selected)
*Saturation:	(will only be shown when **Parameters** → **Submenu 7** → **A. Saturation** has been selected)
max. Correction Closed Loop:	(will only be shown when **Parameters** → **Submenu 7** → **Closed Loop** has been selected)
Act. Correction Closed Loop:	(will only be shown when **Parameters** → **Submenu 7** → **Closed Loop** has been selected)
Actual:	Actual state (ON, OFF or PWM)

Continued on the next page. . .

. . . Continued from previous Page

Actual (%):	Power-output in percent
Analog Value:	Re-read value from 0 to 1000. This value enables error detection.
Error:	Error code: OK = in order, SCT = short circuit to ground, HI! = erroneous configuration, signal although output not activated

See example channel 9: Output at 50% PWM. Measured analog value 0 → short circuit!

11.7.2 Preset Analog Outputs (PWM)

Preliminary brief introduction of PWM-outputs function:

PWM-outputs index the supply voltage via transistor. Voltage at the output is determined by the ratio selected (i.e. 50% PWM at 24V supply voltage = 12V). Current is determined by the internal resistance of the user connected and by Ohm's law ($I = \frac{U}{R}$).

Ratios 1 and 2 enable three operating modes to be adjusted.

Standard Digital Output:

Ratio 1 and ratio 2 are both set at 100% in this mode. There is no delay when voltage is switched on and off. Starting and stop time are irrelevant for this mode. Figure 11.24 shows output 10 operating in this mode.

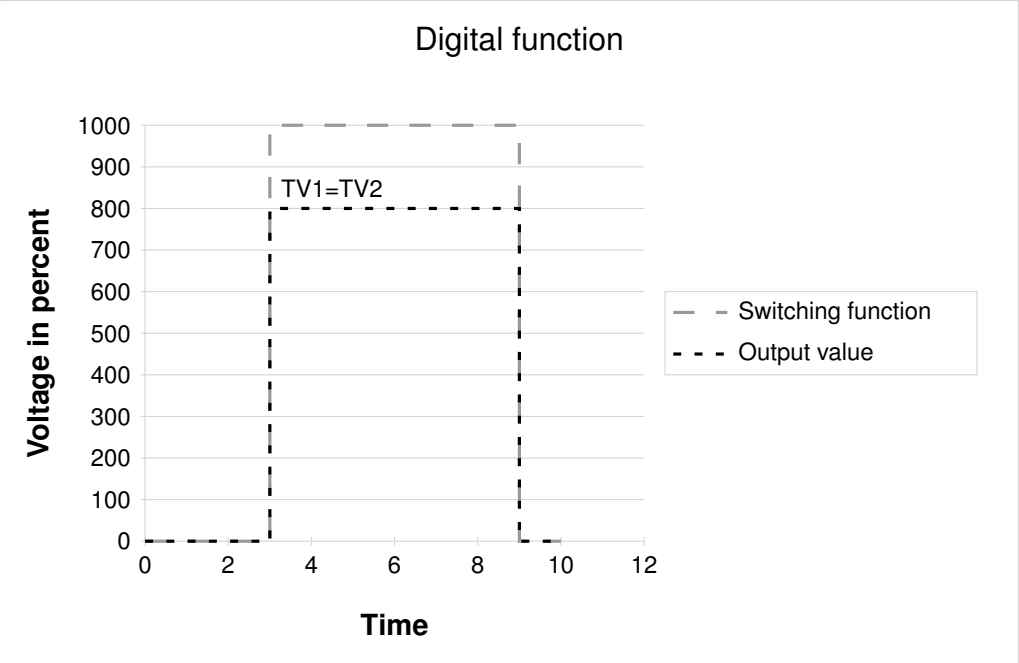


Figure 11.24.: Preset Analog Outputs (PWM)

Power-Saving Function:

Here ratio 2 is less than ratio 1. Immediately on start-up voltage of ratio 1 is activated, to switch an on/off - valve. After a preset time (starting time) voltage is reduced to ratio 2. This is often possible because the valve was already switched and requires less power to maintain its new state. Again, for deactivation stop time need not be considered. In figure 11.24 channel 9 is operated in this mode.

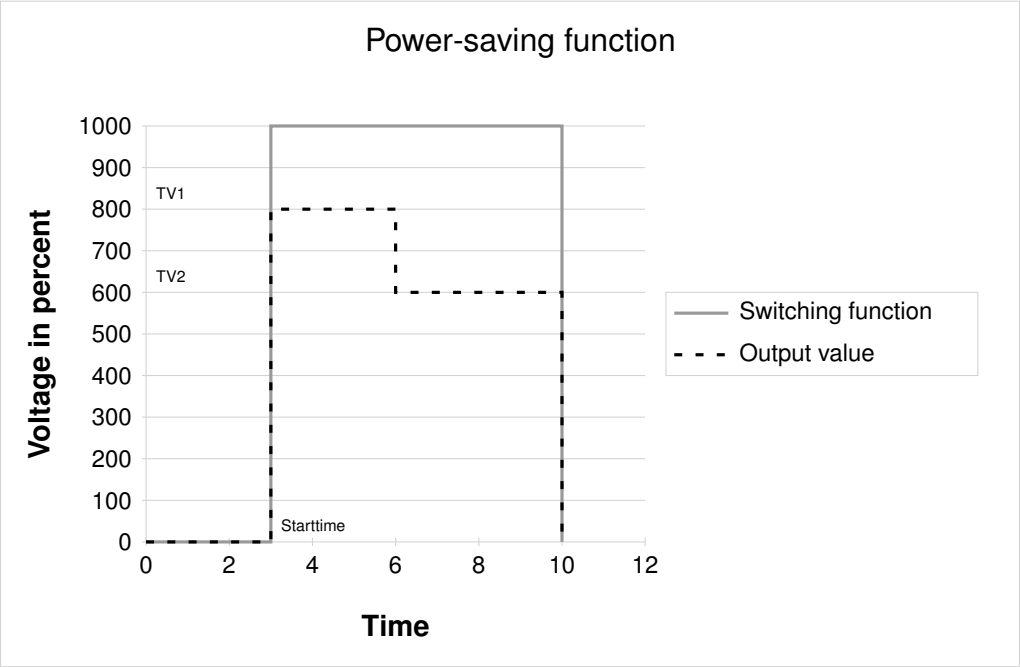


Figure 11.25.: Preset Analog Outputs (PWM)

Proportional Valve Function:

A ramp function is automatically generated in case ratio 1 is less than ratio 2. As the figure indicates, voltage immediately increases to ratio 1 and rises during starting time to reach ratio 2. In case the selected stop time is not 0, voltage decreases to TV1 during this time period and is then set at 0. Figure 11.26 shows channel 8 operating in this mode. Moreover, it shows that the read-in value does not yet correspond to the full “analog value”. Consequently, this image was recorded at the end of starting time. (Ratio 2 = 85%)

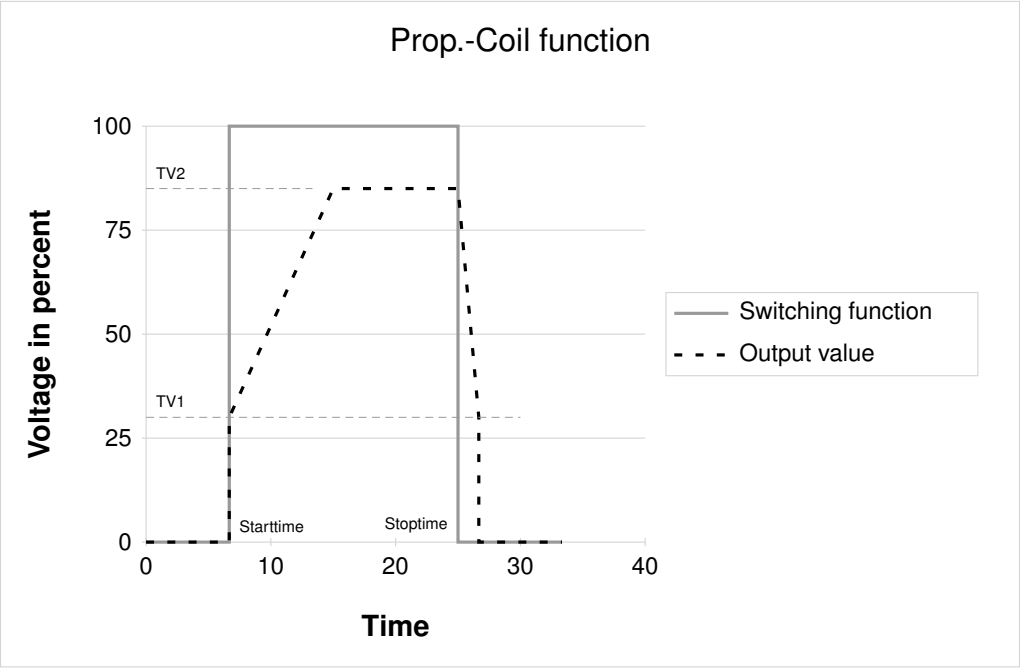


Figure 11.26.: Preset Analog Outputs (PWM)

Click **Parameter** in one of the menus for the digital outputs. Figure 11.27 will be displayed:

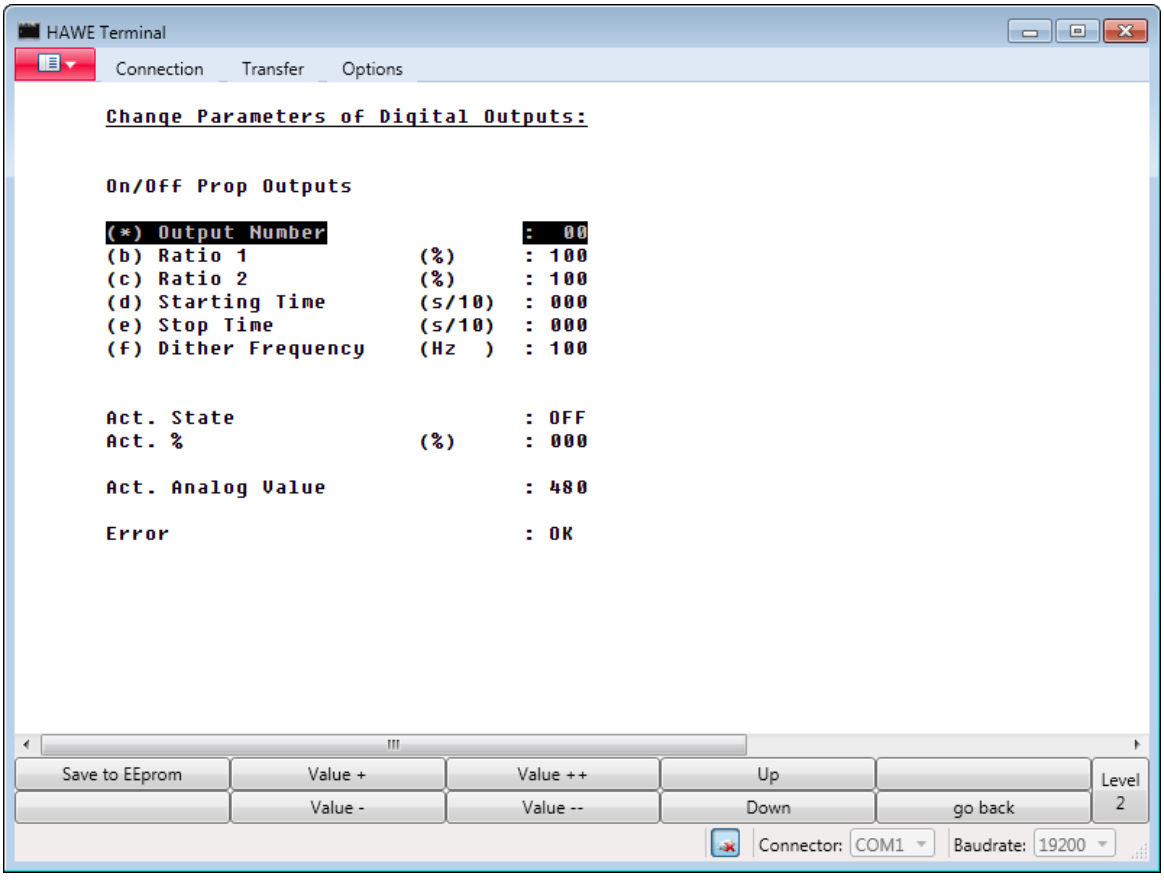


Figure 11.27.: Preset Analog Outputs (PWM)

Toggle between the buttons **Up** and **Down** to access the various lines. The activated line is highlighted. Even faster: you can select the parameter that you want to change, by typing the letter in brackets before the parameter with your keyboard.

Use the buttons **Value +** and **Value ++** to increase the highlighted value in increments of 1 or multiples. Use the buttons **Value -** and **Value –** to decrease the highlighted value in increments of 1 or multiples.

You can also directly insert the value via your keyboard: type a “v” in the end to change toggle to negative values, type a “w” to refresh the screen and restart insertion of numbers.

The adjusted values will instantly become effective but are deleted at next reset if not saved to EEprom.

Save to EEprom will save your settings permanently in the device’s EEPROM.

The respective values displayed are (table 11.13):

Output Number:	Channel of digital output
Ratio 1:	Power-output in percent during state 1. Can be adjusted in increments of 5% (1% for PLVC8x2-X-EW).
Ratio 2:	Power-output in percent during state 2. Can be adjusted in increments of 5% (1% for PLVC8x2-X-EW).
Starting Time:	Duration of starting phase until power-output 2 is reached (in 1/10 sec)
Stop Time:	Time for power-output to be decreased (in 1/10 sec)
Dither Frequency:	Dither frequency (adjust to 0Hz≙50Hz, 1≙100Hz or 2Hz≙200Hz)*
Act. State:	Actual state of the output (ON, OFF or PWM)
Act. %:	Actual state in percent
Act. Analog Value:	Analog value re-measured
Error:	OK for in order, SCT for short circuit, HI for erroneous configuration.

* for PLVC8x2-X-EW the already mentioned values can be used, but there is also the possibility to adjust the Dither frequency with a finer resolution:
3-110 ≙ 0,3Hz - 11Hz, resolution: 0,1Hz
111-200 ≙ 11Hz - 100Hz, resolution: 1Hz
201-250 ≙ 101Hz - 500Hz, resolution: 10Hz

Make sure to save all your changes. Click **go back** to return to the start menu from where you can return to the basic menu by clicking **go back** a second time.

Other sub-groups can be accessed via the submenu **Digital Outputs**.

Submenu 3 (figure 11.28) displays the actual state of the relays on motherboard. Relay on right (%q2.3) is the emergency-stop relay. “-” indicates that emergency-stop is activated and outputs are without power.

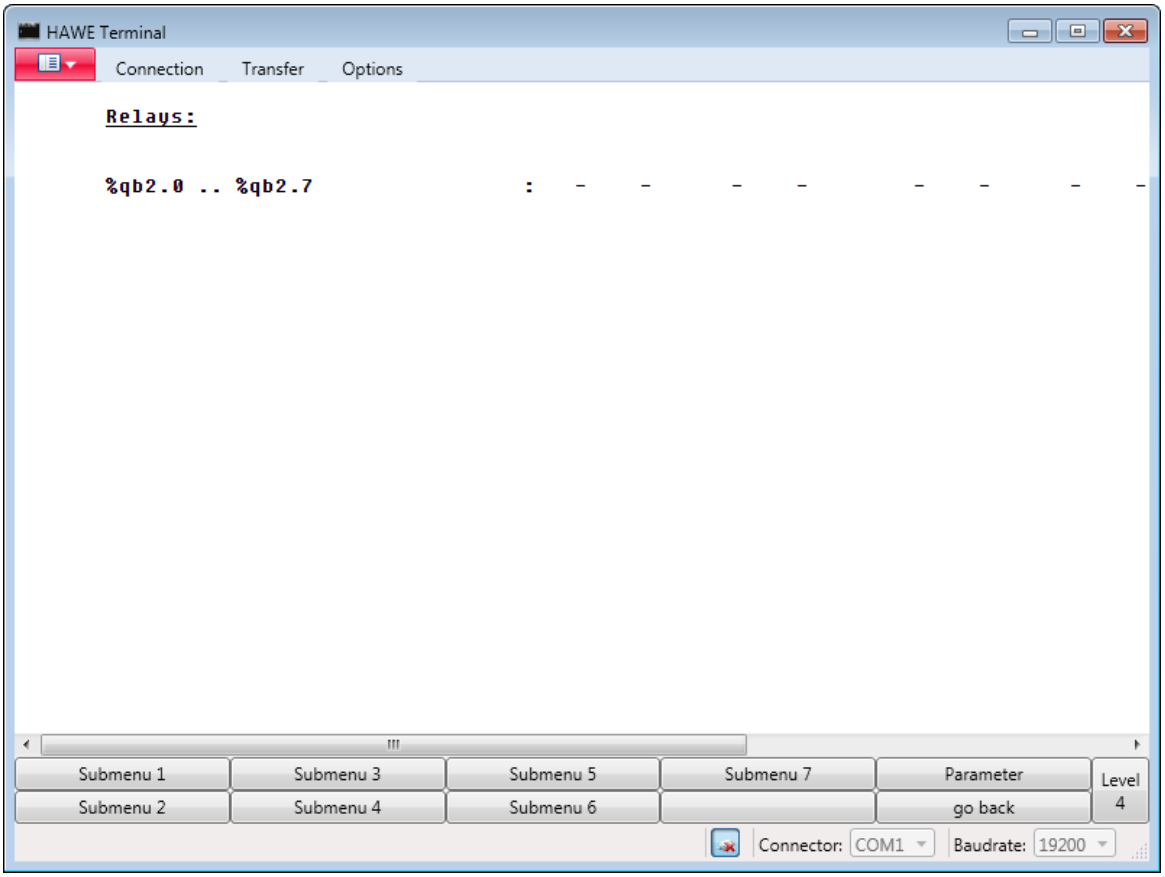


Figure 11.28.: Preset Analog Outputs (PWM)

Submenu 4 shows an overwiew over all local digital outputs (figure 11.29).

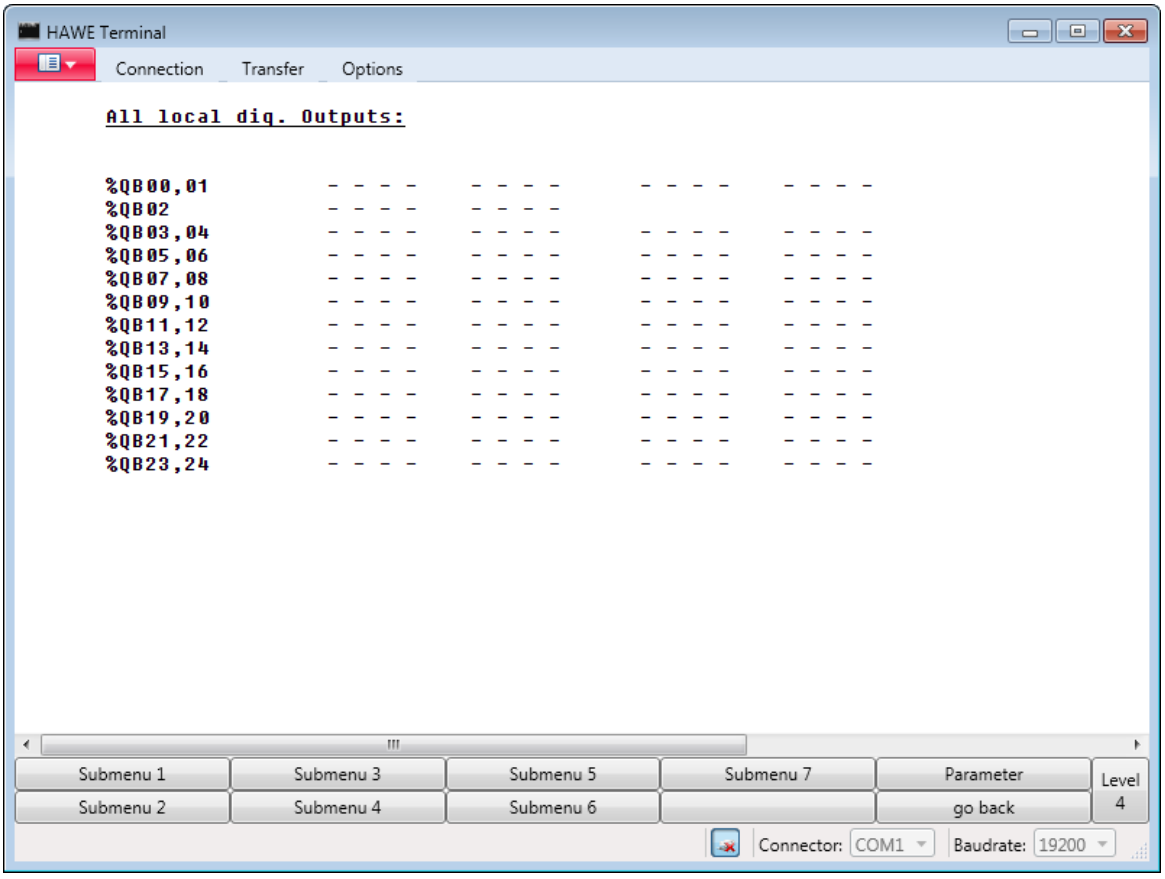


Figure 11.29.: Preset Analog Outputs (PWM)

In **Submenu 5** you can see an overview over all digital outputs (figure 11.30):

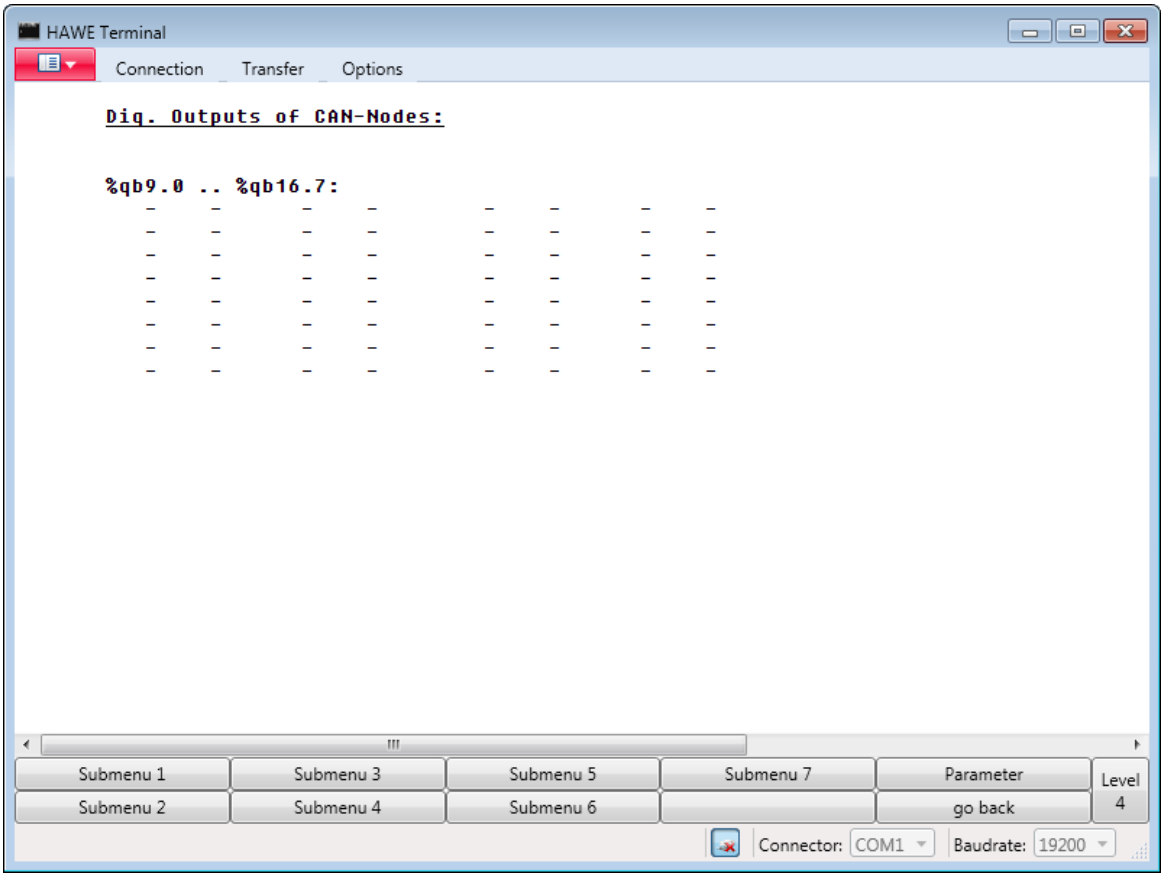


Figure 11.30.: Preset Analog Outputs (PWM)

This screen shows state of digital outputs of CAN nodes connected via CAN bus. They can be:

- CAN POWER-Node with 8 relays.
- CAN Display with 1 relay.
- CAN Analog-Node with 3 relays.

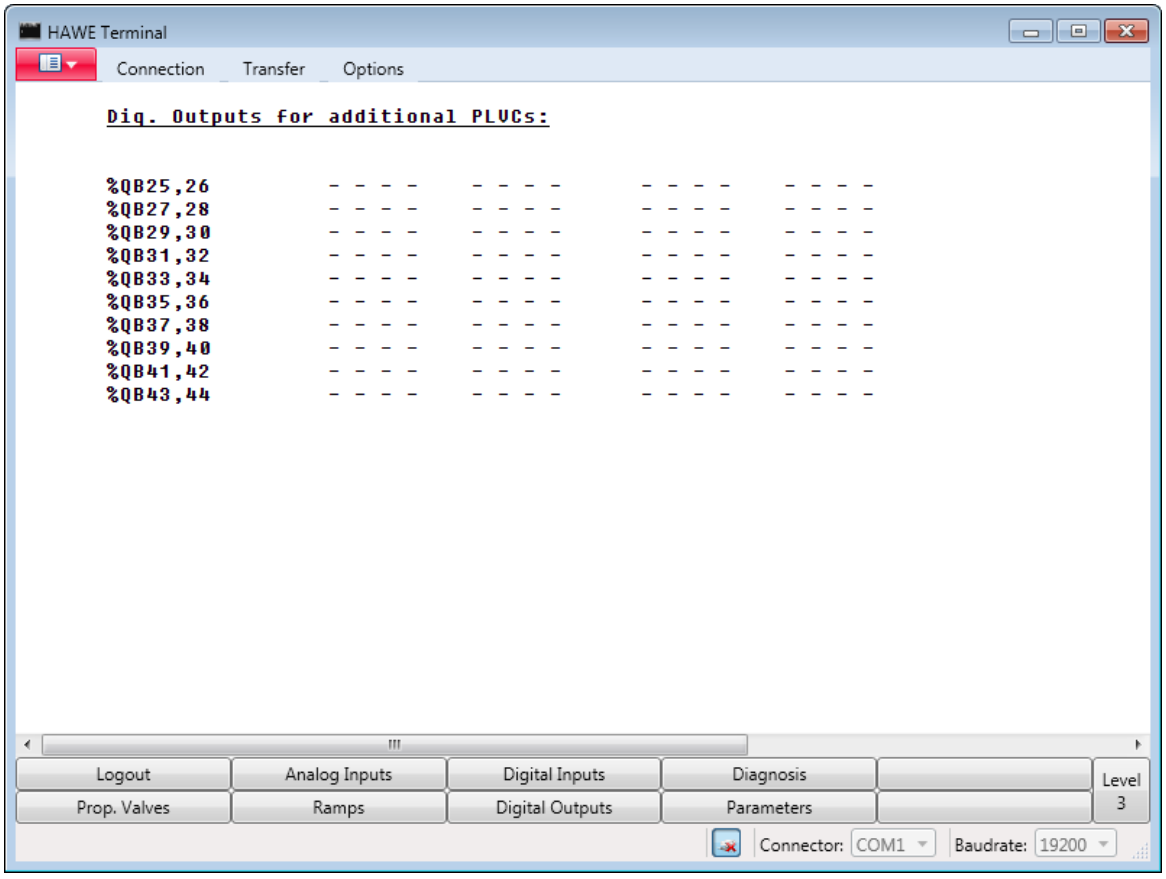


Figure 11.31.: Preset Analog Outputs (PWM)

Submenu 6 (11.31) provides an overview over all digital outputs for additional PLVCs, connected via CAN bus. Here you can see %QB27.1, which is %QB2.1 on the remote device.

11.8 Menu Diagnosis: Specific Information

Click **Diagnosis** to access the diagnosis selection (figure 11.32).

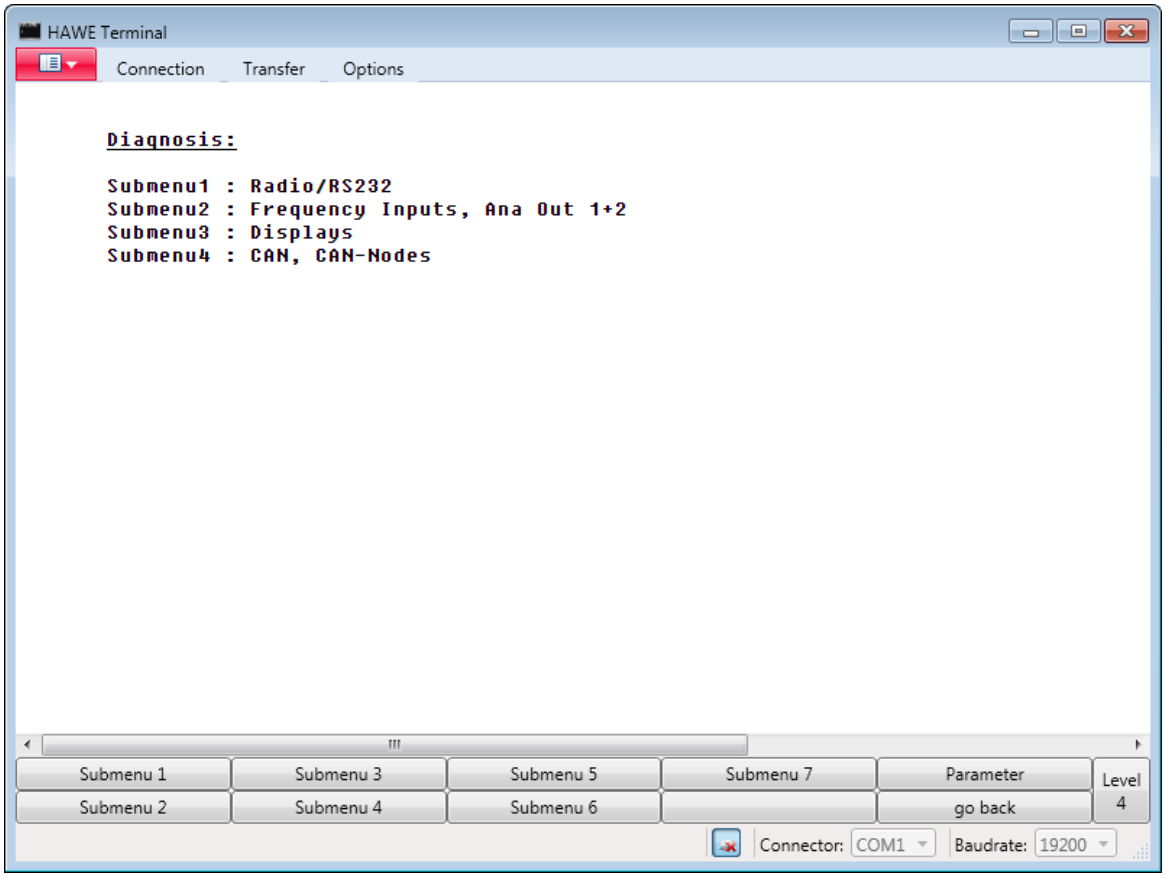


Figure 11.32.: Menu Diagnosis: Specific Information

If PLVC2 detects a profibus line “Submenu 1” will return the entry “Profibus” instead of “Radio”.

The submenus are explained in the following.

11.8.1 Radio/Profibus

Radio

Access the diagnosis menu (figure 11.33) **Diagnosis Radio** by clicking the button **Submenu 1**.

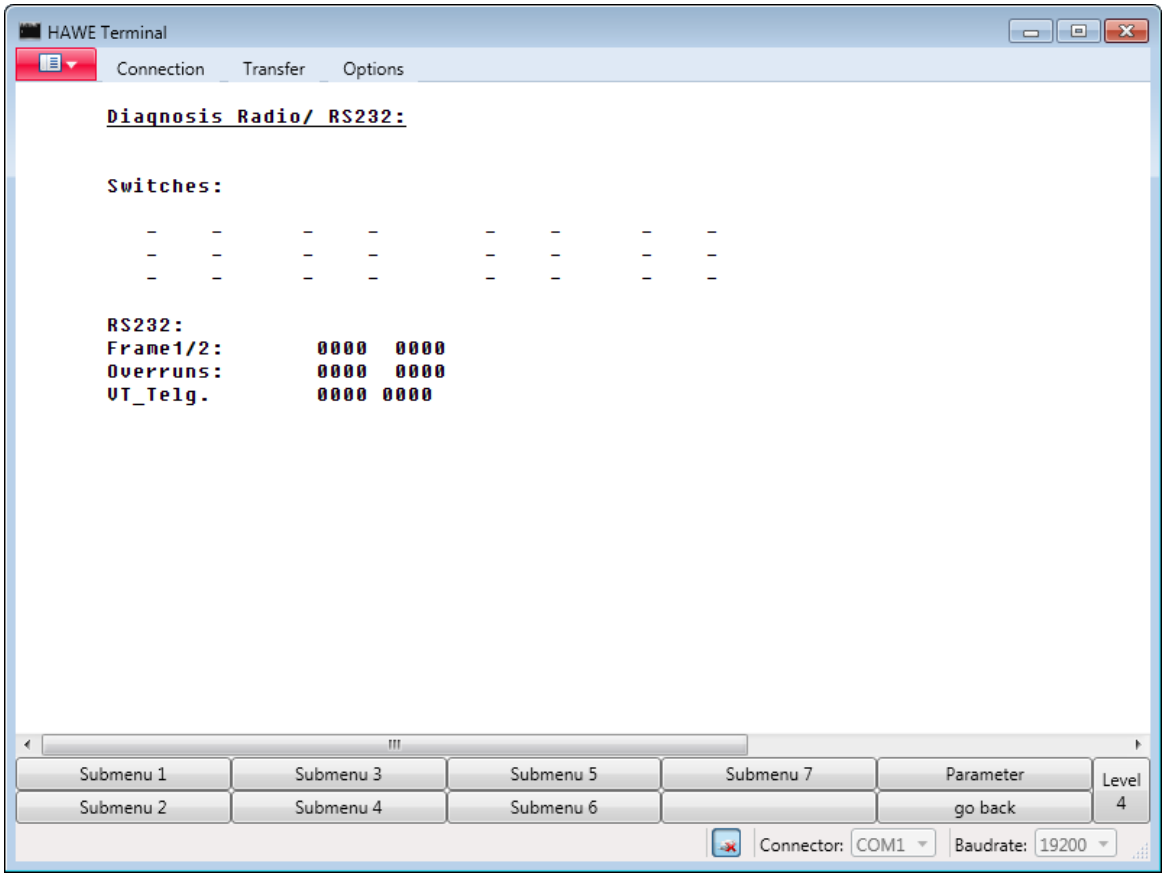


Figure 11.33.: Radio/Profibus

The respective lines provide the following data (table 11.14):

Switches:	Digital inputs radio, IB4-IB6
RS232:	Diagnosis of RS232 interfaces
Frame1/2:	Number of RS232 frame errors
Overruns:	Number of RS232 overrun errors: Telegram not processed on time
VT_Telg.:	Number of successful telegrams

Profibus (only with PLVC2)

Click the **Submenu 1** button to display the following screen. It replaces radio in the overview.

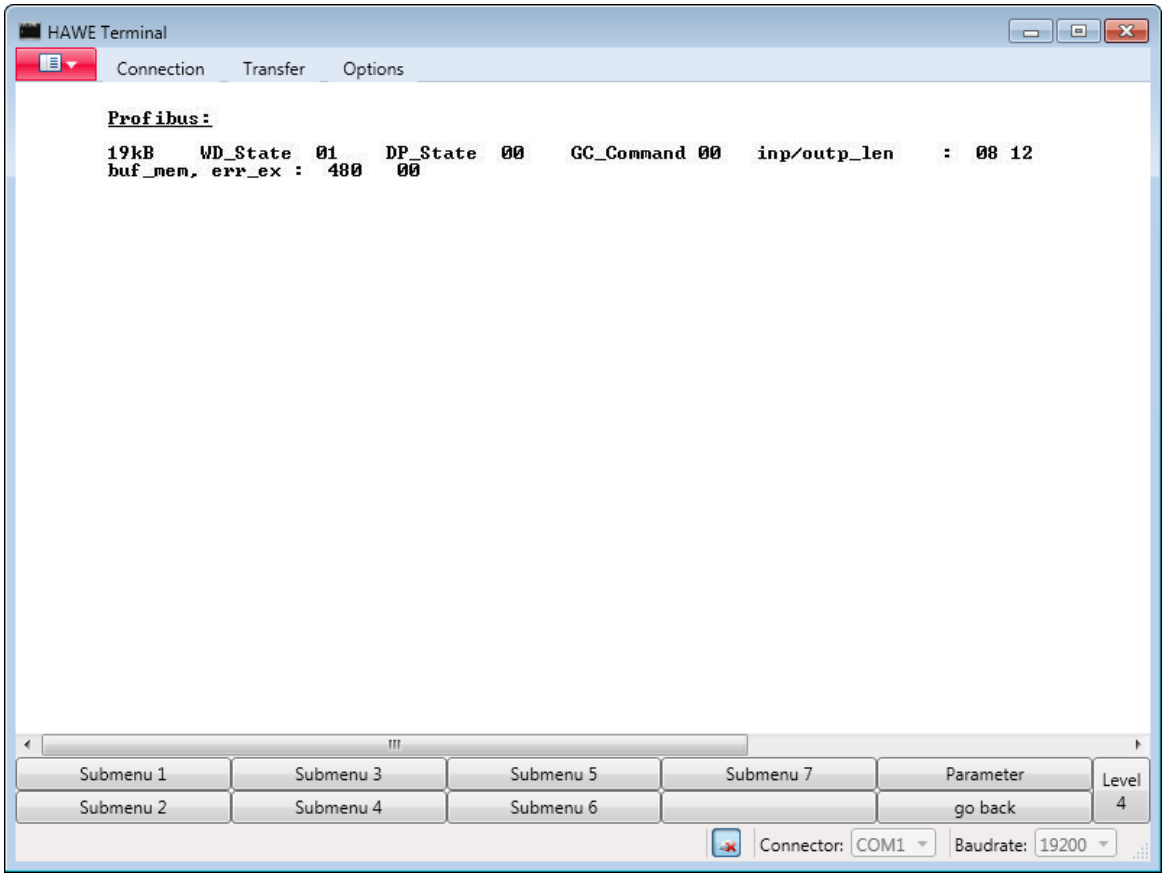


Figure 11.34.: Radio/Profibus

The following values are provided (table 11.15):

19kB:	Speed of connection, baud rate
WD_State:	Watch-dog (State 2=OK)
DP_State:	Profibus state (0=offline, 1=baud rate detected, 2=OK)
GC_Command:	Counter for global profibus commands
Inp/outp_len:	Number of bytes for in- and output bytes
Buf_mem, err_ex:	Total number of used bytes on profibus chip

This screen is still under development.

11.8.2 Frequency Inputs

Access the diagnosis menu **Frequency Inputs** by clicking the button **Submenu 2** (figure 11.35):

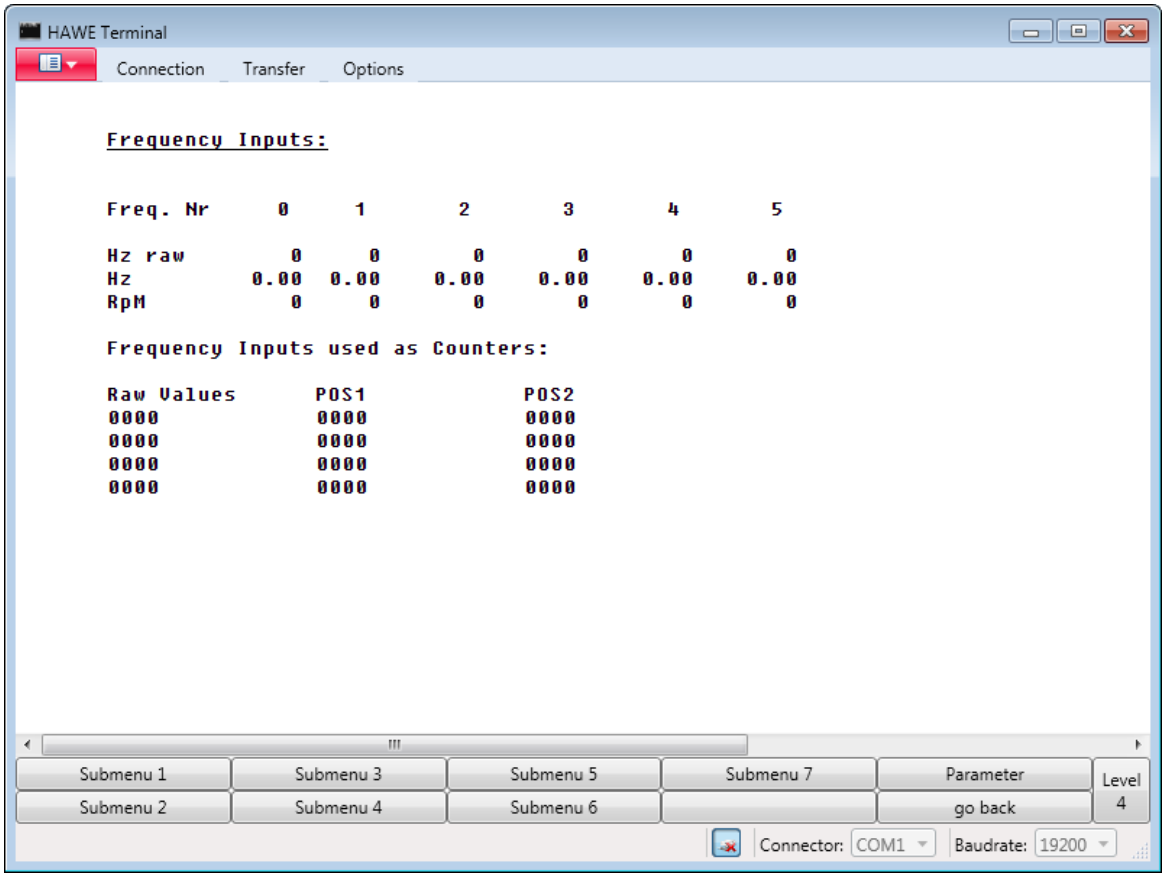


Figure 11.35.: Frequency Inputs

The respective lines provide the following data (table 11.16):

Hz raw:	Directly measured impulses per second
Hz:	Frequency in 1/4, 1/8 or 1/16 Hz (preset in function FQ_INI)
RpM:	Computed rpm.

You can also utilize frequency inputs as impulse counters (incremental encoders), where the counted value means for example a position.

These “positions” agree shown in the last three lines.

11.8.3 Displays

Access the diagnosis menu (figure 11.36) **Displays** by clicking the button **Submenu 3**.

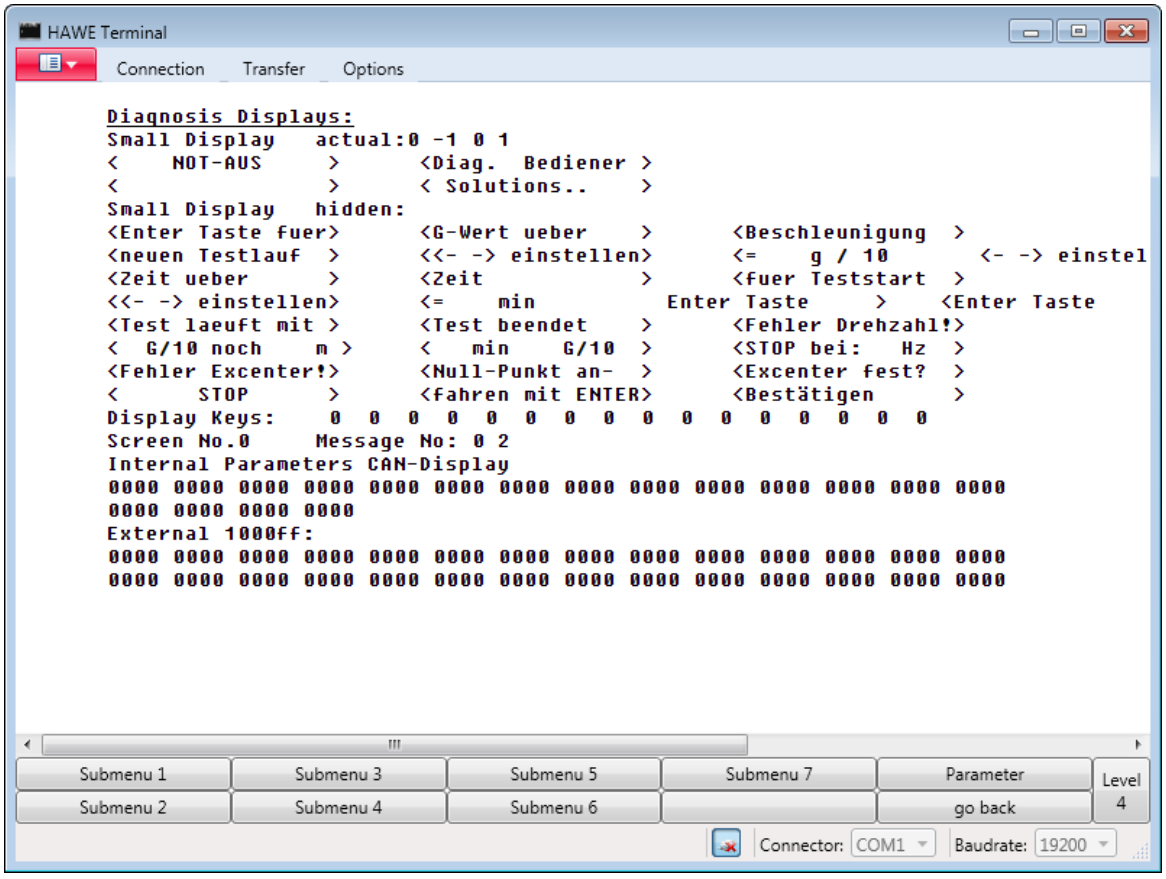


Figure 11.36.: Displays

The respective lines provide the following data (table 11.17):

Small Display actual:	Displays the visible lines, when small display (CAN BC) is connected
Small Display hidden:	Lines, which the display holds in its internal storage and that can be activated.
Display Keys:	Keys of the graphic display IB7
Internal Parameters CAN-Display:	Internal parameters of the CAN display. They are stored in the display where they can be altered via function buttons.
External 1000ff:	External variables of the display

11.8.4 CAN Bus

Access the diagnosis menu (figure 11.37) **CAN-Bus** by clicking the button **Submenu 4**:

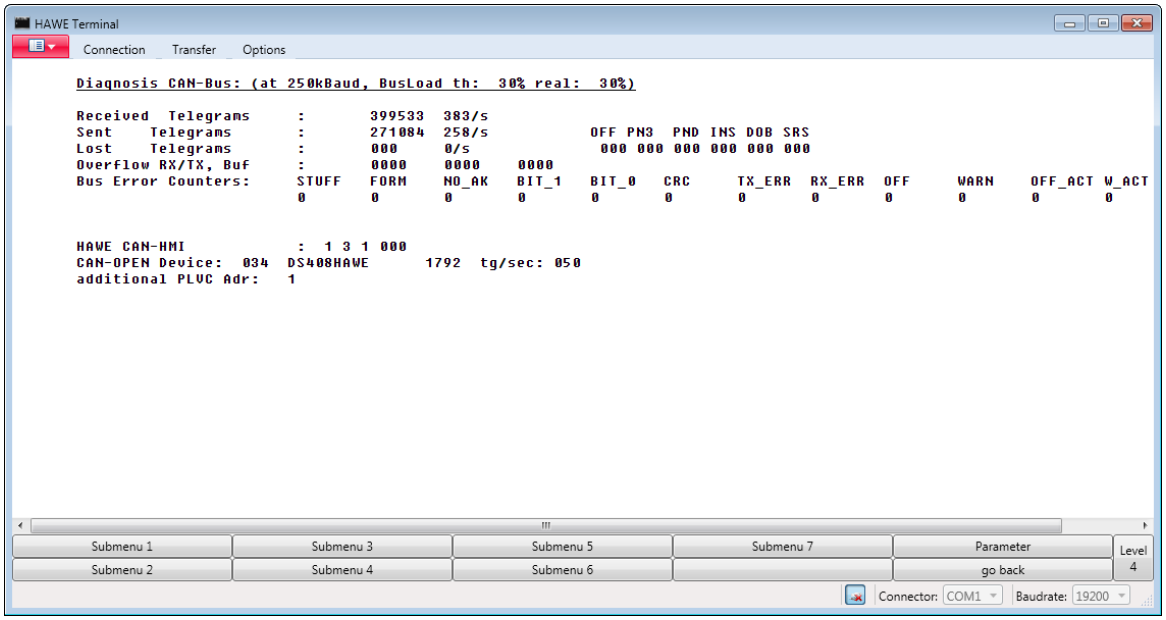


Figure 11.37.: CAN Bus

The respective lines provide the following data (table 11.19):

Received Telegrams:	Number of CAN bus telegrams received. This number will increase, when bus is functional.
Sent Telegrams:	Number of telegrams sent.
Lost Telegrams:	Number of telegrams lost. They are a consequence of faulty lines.
Overflow RX/TX Buf:	Displays the number of telegrams lost because they were not sent out in time, followed by display of RX-FIFO and TX-FIFO (sending and receiving memory).
Bus Error Counters:	Number of serious bus errors/warnings.

The following lines show which other CAN bus devices were discovered, e.g.:

HAWE-CAN-HMI:	HMI
CAN-OPEN Device:	CANopen device
additional PLVC Adr:	additional PLVC

Bus Error Counter Considerations:

- If all except CRC count, there is a connection with wrong baudrate.
- If NO_AK is counting alone, there is no other CAN bus participant listening: Either not connected, or powered off. This is not an error, but a warning.

- If BIT_0 is counting alone, there is a short circuit beteen CAN_HIGH and CAN_LOW.

Bus Load Considerations:

- If the third value in the line overflow RX/TX, buf never becomes zero, the bus load is too high, so not all telegrams can be transmitted.
- Make sure to deactivate telegrams that are not necessary in **Menu** → **Parameter** → **Submenu 8**.
- Increase the value of CAN DELAY in **Menu** → **Parameter** → **Submenu 4**!

Termination Considerations:

NOTE

The CAN bus must be terminated at the two most remote points of the bus.

This is usually done via bridges between two connector pins.

- If power is switched off, you should measure 60Ω between CAN_HIGH and CAN_LOW - pins.
- If you measure 120Ω, you have only one termination.
- If the bus is small, it will work nevertheless.
- If you measure 40Ω, you have three terminations, so remove one of them.
- If you measure some kΩ, you have no termination.

11.9 Preset Parameters

In the basic menu click the button **Parameters** to display the parameters menu in figure 11.38:

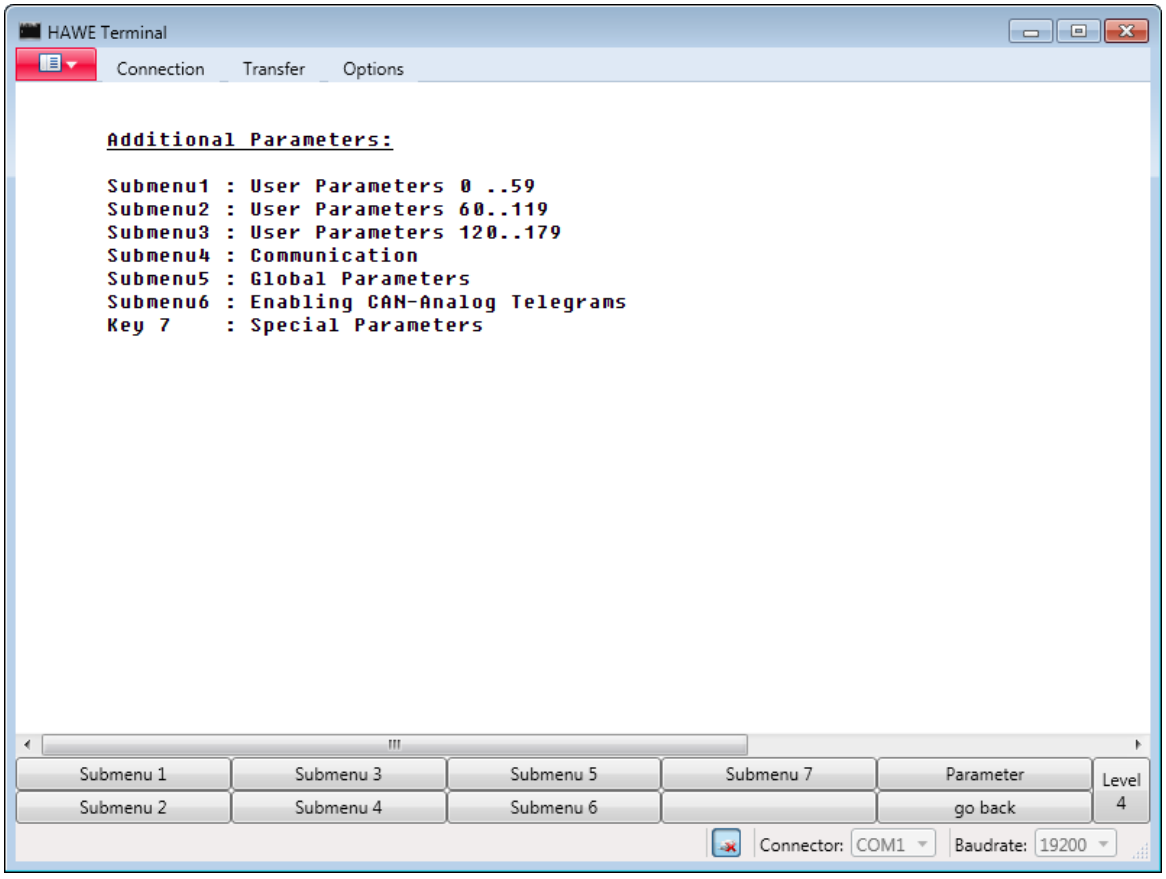


Figure 11.38.: Preset Parameters

11.9.1 User Parameters

Access the user parameters via **Submenu 1-3** (figure 11.39).

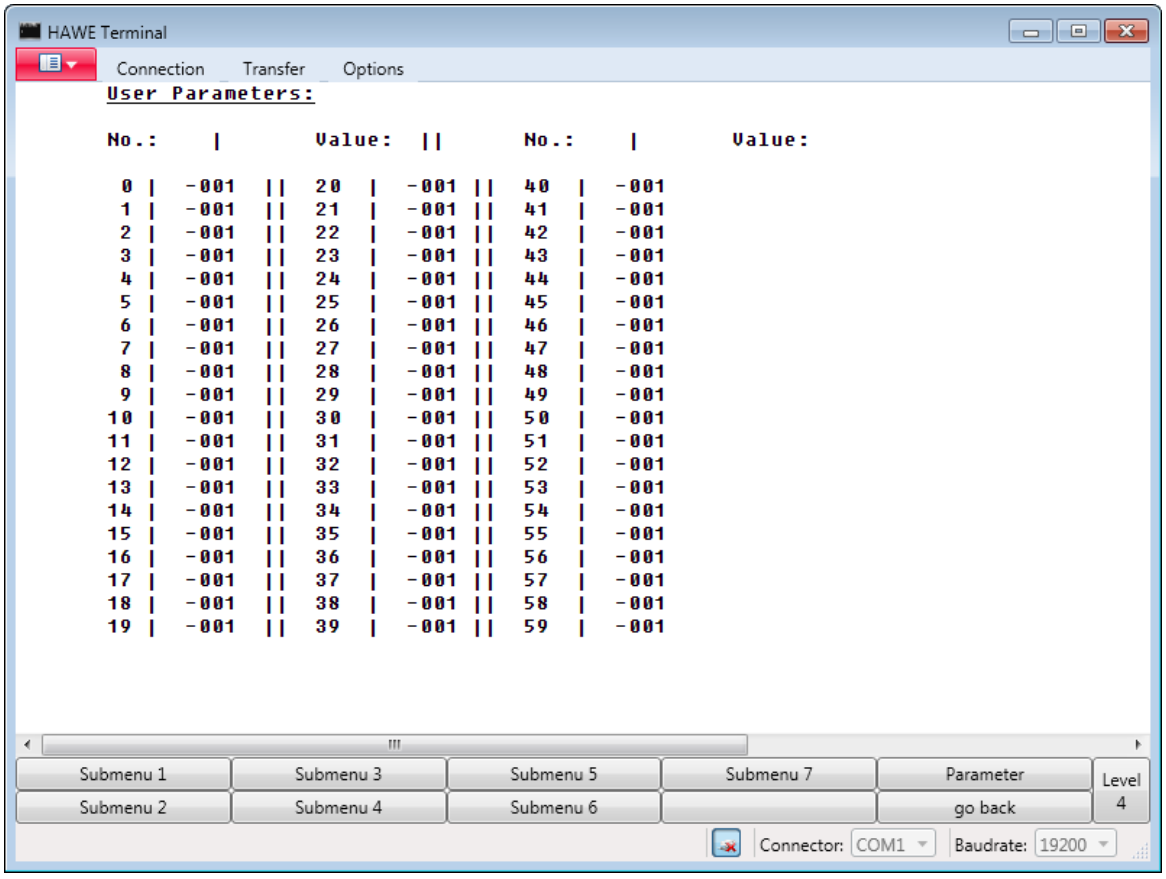


Figure 11.39.: User Parameters

PLC can access user parameters. Values that are not yet determined during programming should be defined as user parameters. This enables users without specific programming skills to change them later.

Hit the **Parameter**-button to get to the screen shown in figure 11.40:

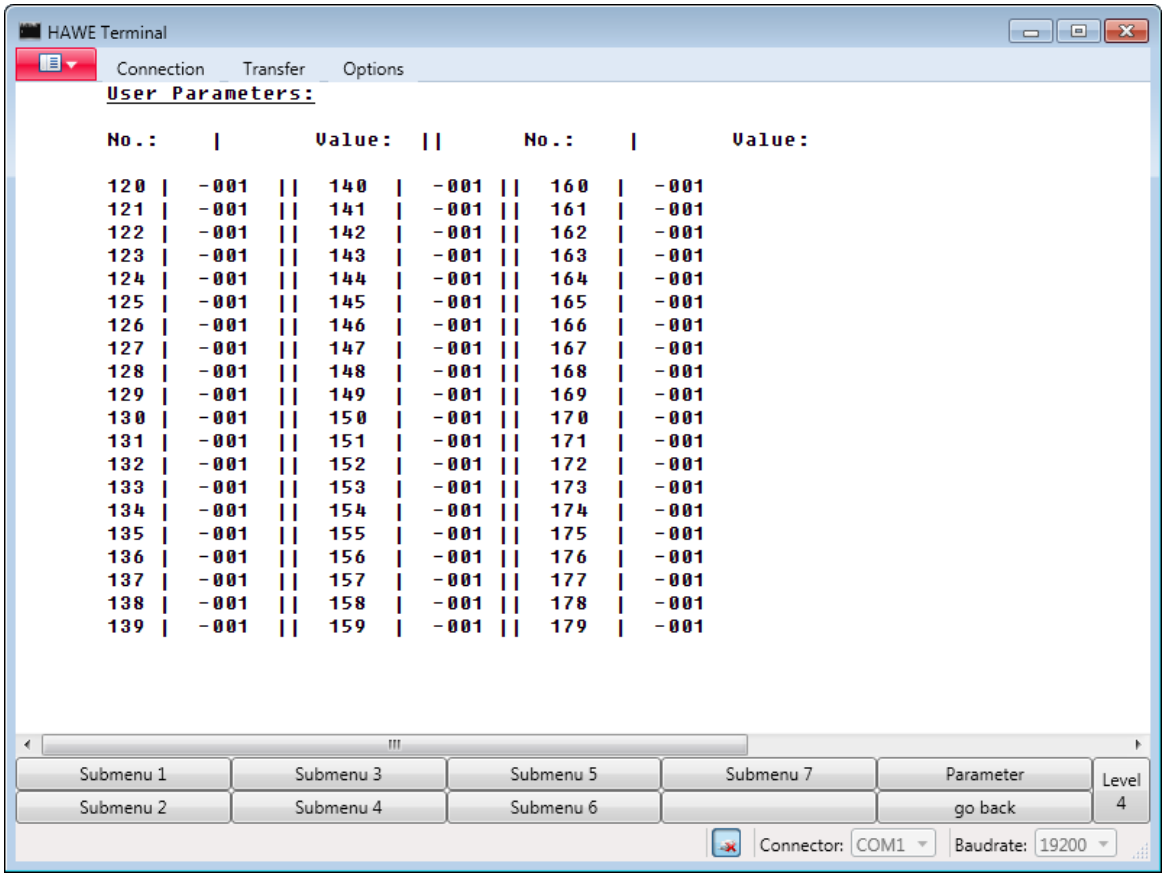


Figure 11.40.: User Parameters

As a default, the text behind the parameter just says 0.th parameter, ...

These texts can be modified (here: max. speed %) via OpenPCS, via function block DISP_TXT, channels 100ff.

Toggle between the buttons **Up** and **Down** to access the various lines. The activated line is highlighted.

Use the buttons **Value +** and **Value ++** to increase the highlighted value in increments of 1 or multiples. Use the buttons **Value -** and **Value –** to decrease the highlighted value in increments of 1 or multiples.

Basically 389 User Parameters can be set to the PLVC21 and PLVC41 with the Terminal. You have to watch out that all Parameters up to 127 are saved directly after having been entered. To get the same effect with Parameters 128 up to 389 you have to press additional the "Save to EE"-button. Alternatively you can write the User Parameters up to 127 with the function-block PUT_EE in OpenPCS. Thats not possible with the Parameters 128 and increasing.

In contrast to the PLVC21 and PLVC41 you can type in every User Parameter of the PLVC8 within the Terminal and they are saved directly. It is also possible to write all 389 User Parameters in OpenPCS via the function-block "PUT_EE".

User Parameters can cover a range from -32630 to 32600

NOTE

Watch out not to use the special-parameters 99, 103, 119 and 127 for they can cause unintended actions.

Click go back to return to the parameter menu from where you can return to the basic menu by clicking go back a second time.

11.9.2 Communication

Preset parameters for communication in **Submenu 4** (figure 11.41).

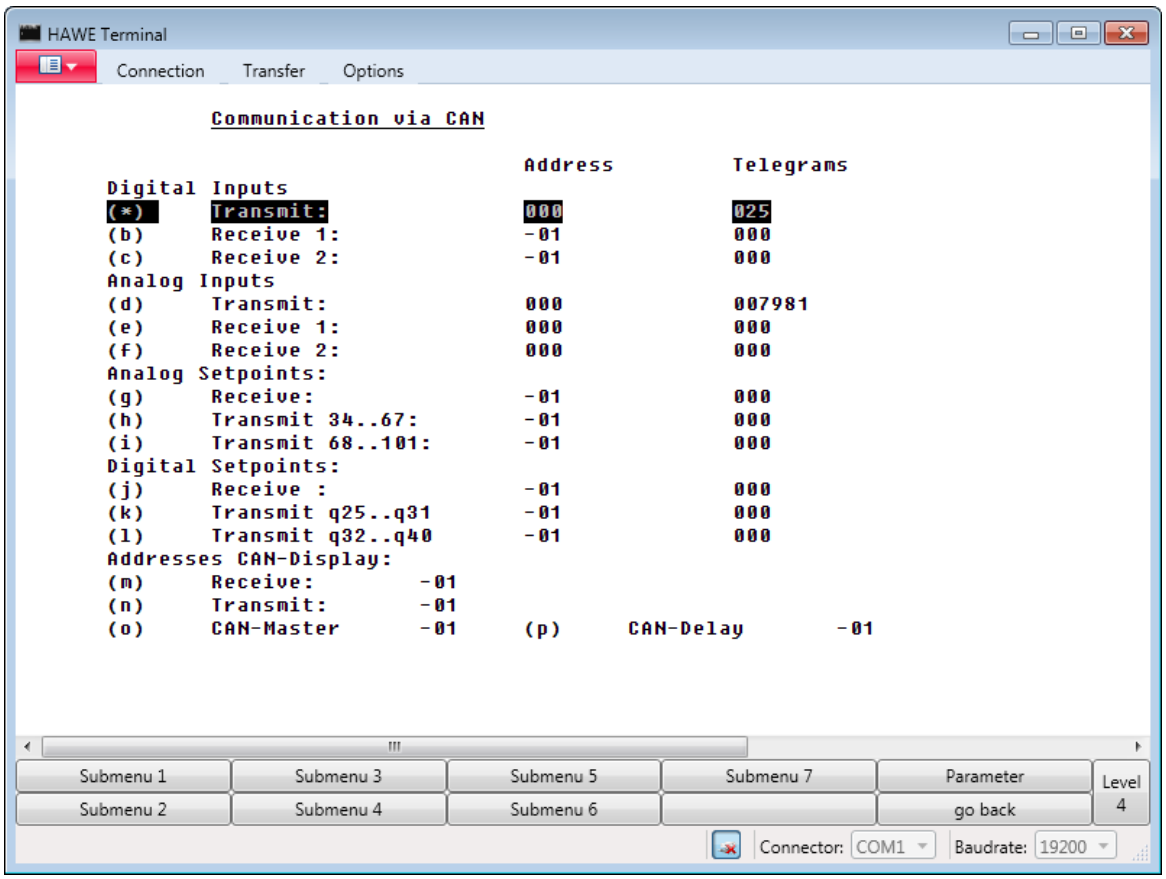


Figure 11.41.: Communication

If you prefer to mouse-click through the menu, click the button **Level:** twice.

This menu allows you the preset whether and what your device transmits and receives.

There are different sections: digital inputs, analog inputs, analog setpoints, digital setpoints and addresses CAN display.

In each of these sections you can allocate different addresses to the device. The addresses allocated to the different sections can also be identical without interfering with one another. Values set to -1

mean that their corresponding function is turned off and no telegrams are transmitted and/or received. The main CAN address is the first value in the section “Digital Inputs” / “Transmit”. The column “Telegrams” indicates whether telegrams were transmitted or received via bus.

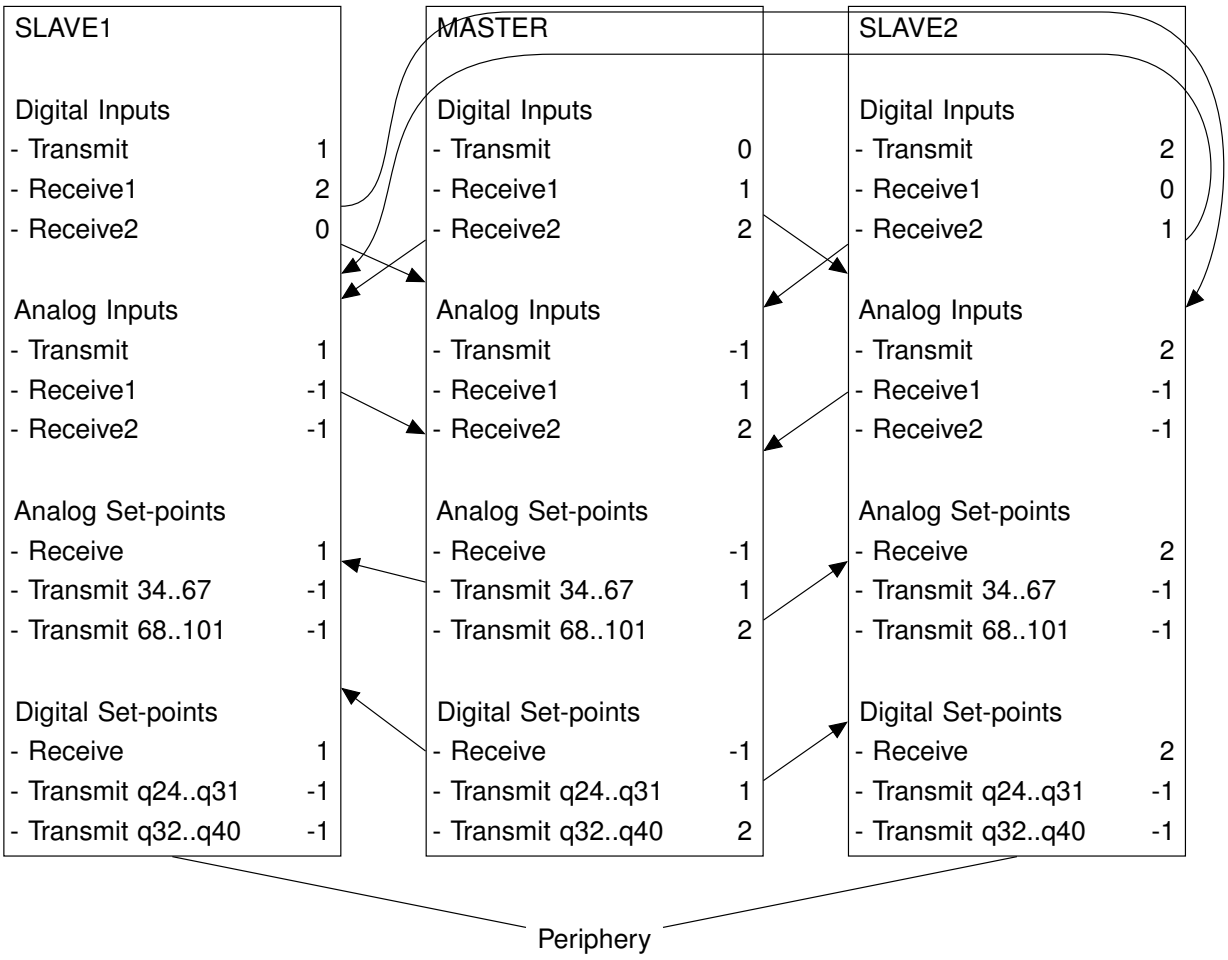


Figure 11.42.: Communication

Figure 11.42 above shows one possible combination of three PLVCs. This combination has one PLVC as master controlling the other two devices.

A maximum of four “slaves” is possible.

Figure 11.41 corresponds to the configuration of the MASTER.

These settings determine the data each device transmits and receives.

Never configure more than one device as master “1” to administer telegrams for CANopen on the bus. The other devices must be configured to “-1”.

The “Addresses CAN Display” are allocated to the large CAN display. They should correspond with the addresses configured in the respective display; i.e. transmission from the display should be configured to the PLVC’s “Receiving” address and vice versa.

Decrease the repeat rate for transmission via CAN delay to prevent overloading the bus with too many analog values.

Setting	Repeat rate
0	20ms
1	40ms
2	60ms

Table 11.20.: Repeat Rate CAN Signals

11.9.3 Global Parameter Values

Change of global values (figure 11.43):

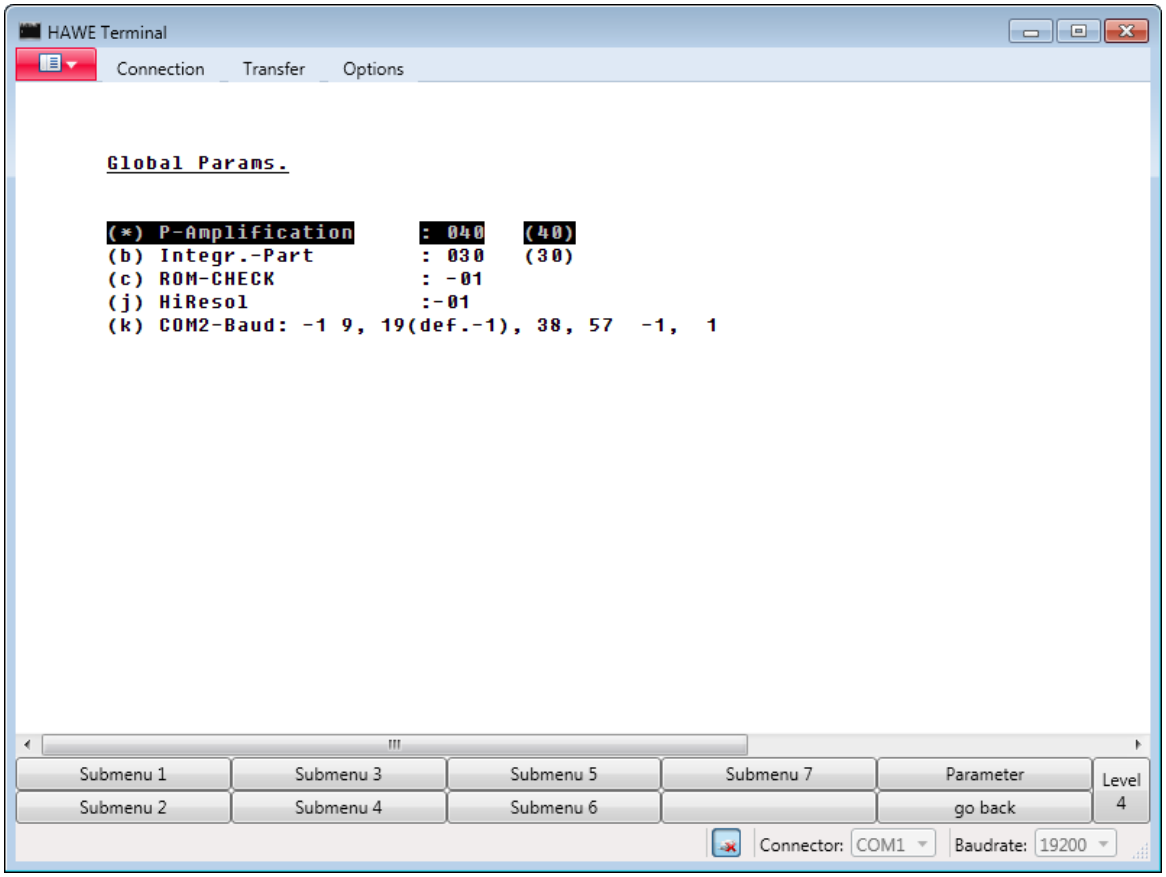


Figure 11.43.: Global Parameter Values

The following values are provided:

1. P-Amplification: Proportional amplification for current control (default 40)
2. Integr.-Part: Integral part (tracking time) for current control (default 30)
3. ROM Check has been implemented for safe applications.

- -1 means: no check
- 1 means: check of flash-program is done at startup (and lasts about 3 sec.)
- 0 or 2 means: check is done, and in case of error, no outputs are set any more. (severe error).

4. MODE:
   - -1 means no special check in current control
   - 1 means special check in current control: current, supply-voltage, resistance, and PWM must be in a special relation, otherwise an error is given to OpenPCS.
5. Hi Resolution:
   - -1 means: normal behaviour of analog inputs 40-43
   - 20 means: special behaviour of analog inputs 40-43: oversampling with 20 samples in 20ms is done, therefore resolution of analog input becomes more than 1000 steps (only possible with analog inputs 40-43)
6. Flash Speed: when set to 50, PLVC will become faster, as no more waitstates are used to access flash.

Amplification and integral part are related to closed loop of current control and normally do not have to be modified.

11.9.4 Proportional Valves (only on PWM)

Preset the current outputs to **Prop. Valves** → **Submenu 3**. Controlled is default.

In case uncontrolled current outputs is set, milli-ampere are interpreted as PWM-ppt, so only a voltage is produced, not a current. No errors will be detected anymore. This configuration can be used to use these outputs for lamps (where only 20mA will flow).

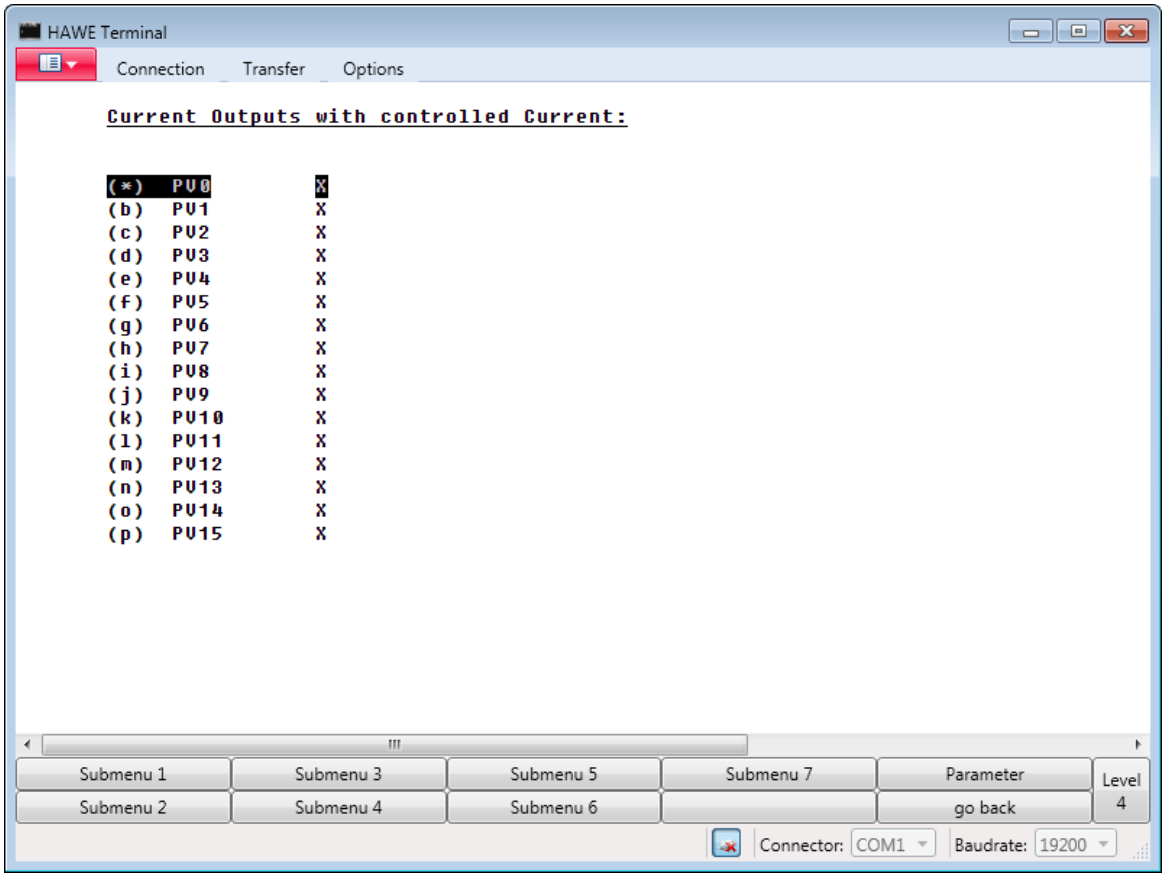


Figure 11.44.: Proportional Valves (only on PWM)

11.9.5 Special Parameters

You can access this menu (figure 11.45) only by clicking the keyboard-key “7”! See special parameters (table 11.21) below. Any changes to this menu should be applied very deliberately and carefully!

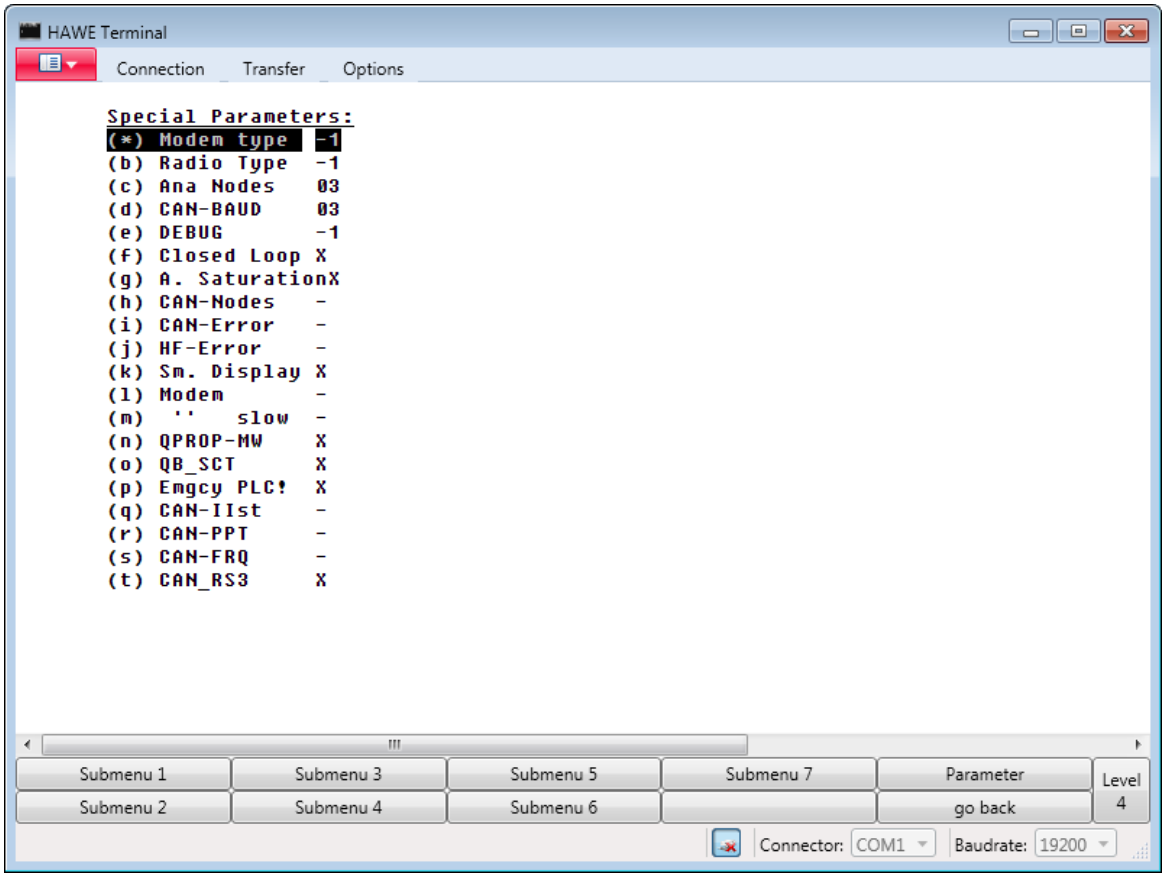


Figure 11.45.: Special Parameters

It is a mixture of different types of parameters. For easy navigation with program keys click repeatedly on **Level: X** until **Level 2** displays. Parameters with value “-1” or “-” are generally switched-off.

The following parameters are provided (table 11.21):

Modem type:	Type of cell-phone that is supported: 3=Nokia; 1=Siemens S35
Radio Type:	Type of radio that is supported: 1,2,3 = NBB, 4=Hetronic, 5=HBC
Ana Nodes:	Types and numbers of analog nodes supported 1=1; 3= 2; 7=3;
CAN-BAUD:	Baud-rate of CAN bus: 0=50kBd; 1=100kBd; 2=125kBd; 3=250kBd
DEBUG:	Triggers additional messages that usually are only necessary for errorhandling
Closed Loop:	Triggers computation of position- and quantity-regulation. This manual does not cover this item. PLVCs automatically regulates spool- position with this item.
A-Saturation:	Activates Anti-Saturation of PLVC
CAN-Nodes:	Activates support for CAN nodes, i.e. CAN BC, CAN POWER..
CAN-Error:	Activates/suppresses error message for CAN
HF-Error:	Activates/suppresses error message for Radio
Sm. Display:	Activates support for 5 keys of CAN BC
Modem:	Activates support for Modem (AT-commands)

Continued on the next page. . .

. . . Continued from previous Page

Modem slow:	Activates support for slow modem (cell phone)
QPROP-MW:	Average value computation for PWM-outputs
QB-SCT:	Switch-off on detection of short circuit at PWM-outputs
Emgcy PLC!:	Emergency-off can be revised from PLC
CAN-Ist:	Transmits actual flows via CAN instead of setpoints of proportional valves
CAN-PRM:	Transmits flows of proportional valves via CAN ppt instead mA
CAN-FRQ:	Releases transmission of actual values for frequency inputs
CAN_RS3:	For more than two PLVCs in the CAN network, an X must be here, so that an external login is possible via RS232

11.9.6 Special CAN Analog

You can access this menu only by clicking on keyboard-key “8”! This screen (figure 11.46) will display the special parameters for CAN bus.

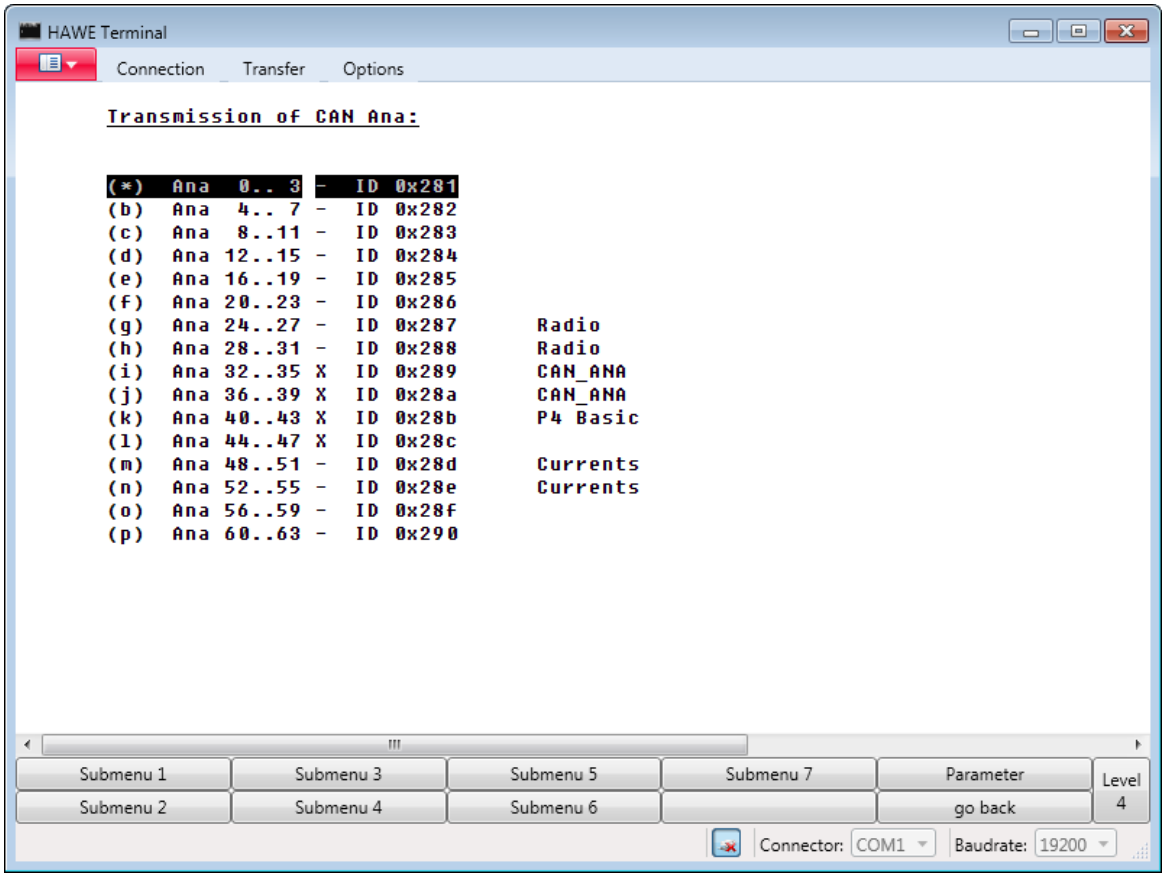


Figure 11.46.: Special CAN Analog

For easy mouse-navigation click repeatedly on **Level: X**, until **Level 2** displays.

Use this menu to control analog inputs to be transmitted and thereby avoid overloading the bus. Channel 0-47 correspond with analog inputs (as given) as well as with the values received for analog nodes. Channels 48-63 are allocated to the currents, which are re-read by the proportional valves.

11.9.7 Menus for Hardware Tests on PLVC

Make sure to deactivate any existing PLC-program when testing outputs to prevent continuous toggling of these outputs through the program. Go to the menu **Parameters** → **Parameter** and reset the user parameter from “99” to “4711”. Restart the device afterwards.

For renewed activation the PLVC requires the user parameter to be set back to 99 before restart.

11.9.8 Programmable Voltage Output

The programmable voltage output of PLVC (Pin 18 of PLVC41 basic device) has a max. load of 100mA.

In OpenPCS it is controlled via ACT_VALVE (`channel:=32`, `setpoint:=0.1000`, `override:=1000`);

This will produce an output of 0-10V.

As there is no ramping and scaling for this output, the PLVC uses this feature of the (not physically available) analog input no 46.

Configure ANA-In 46 as shown in figure 11.47 to get the above mentioned results:

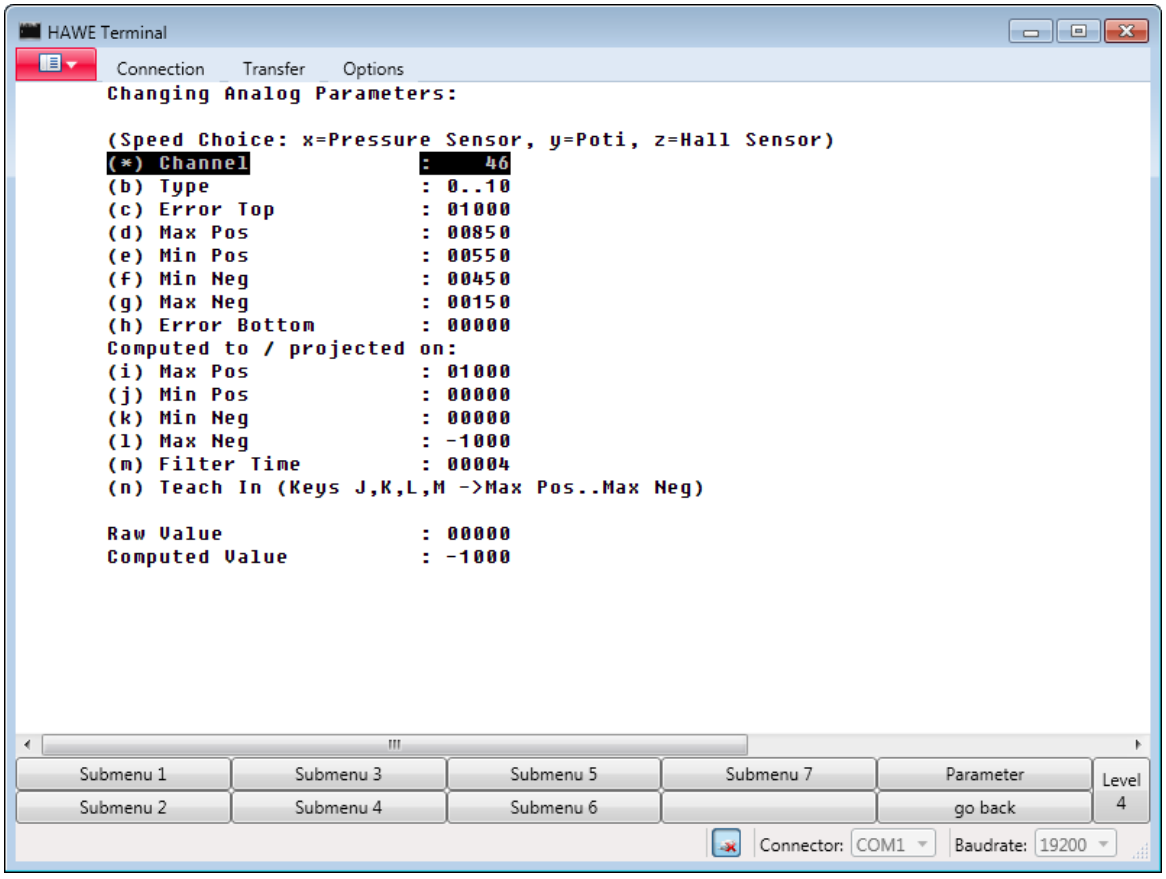


Figure 11.47.: Programmable Voltage Output

The above mentioned setpoint will be seen as “raw-value”, the computed value will be the same in this configuration. Of course it is now possible to adjust the output results via modification of parameter “j” for example:

Setting j) to 100 and i) to 900 will result in an output range between 1-9V.

Setting parameter m) will smooth abrupt changes of the setpoints.

Via RAMP no. 62 parameter PU and PD the output can be ramped.

11.10 Advanced User Guide

The following user guidelines will provide keyboard navigation instructions, which are suitable mainly for users familiar with using the terminal program. Navigating the menus via the keyboard will always display the same line of buttons at the bottom of the screen. Should you require other buttons, mouse-click the button **Level** repeatedly until the required buttons are displayed.

11.10.1 General

- Press the key “DEL” to access the menu below. Log off from basic menu by pressing “DEL” 3 times.
- Select and insert values:
  Lines that are prefixed by a letter can be accessed by pressing that letter on the keyboard. Switching the double spool will automatically toggle the value and the cursor will return automatically back into the first line.
  Other values can be inserted directly into the number sequence. For negative number just press the key “v”.
- Submenus: Submenus can be accessed directly by pressing their number (e.g. submenu 2 is activated by pressing keyboard-key “2”).
- Parameter settings: Go to the selection menu for inputs or outputs and press “Shift + P” to access the parameter settings.

11.10.2 Login

For login simply enter “111”.

11.10.3 In Basic Menu

Please note: the basic menu does not distinguish between upper-case and lower-case letters!

Key	Menu
A	Analog inputs
I	Digital inputs
T	Diagnosis (test)
p	Proportional valves
R	Ramps
D	Digital outputs
P	Parameter
S S	Save parameter

Table 11.22.: Advanced User Guide: Basic Menu

11.11 FAQ - Frequently Asked Questions

11.11.1 How Do You Install New Software (OS/Firmware)?

Software Bootstrap Loader

Software can be downloaded via an exe-file or via visual tool.

Figure 11.48.: Install New Software (OS/Firmware)

PLVC AutoFirmware shows the version of the firmware. After hitting the start button you must not interrupt the connection.

11.11.2 How Do You Install Operating System After Aborted Download?

Download of the operating system can be incomplete in case voltage supply for the PLVC was interrupted. In this case the PLVC will have an incomplete and non-functional operating system. When re-installing the operating system it is important to inform the PLVC of this new installation (this is not necessary if an old and functional operating system is replaced by a new operating system).

Generally this is done by connecting metal pins with a metal object (screwdriver et al.) while PLVCs are turned on (Connect metal pins → Start-up → Disconnect).

The two pins are located in different places of the respective PLVCs:

PLVC2: Remove metal lid. The two pins are located directly behind the 9-pole profibus switch (as seen from power supply line) to the right of the circuit board.

PLVC41: Opening on the left side of the casing (as seen from power supply line), through which both metal pins can be seen. The opening is wide enough to connect the two pins with a screwdriver et. al.

PLVC8: The pin G2 should be set to plus.

11.11.3 How Do You Save Parameters to a File?

To save all parameters to a file, you have to be logged out.

Select **Receive Text file** (figure 11.49).

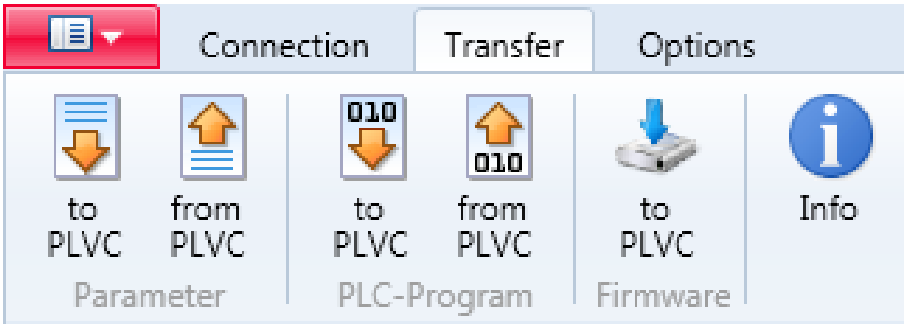


Figure 11.49.: Transfer

Insert the name of the file, ending with ".txt" (max length is 8 characters).

Then hit the button **SAVE to File** in the terminal program.

The parameters can be seen being sent by the PLVC. If they stop, click the stop-key in the left lower corner.

11.11.4 How Do You Copy Parameters to Next PLVC?

To send parameters back, select **Send Text File**.

Insert the name of the parameter file.

Each parameter sent is represented by an asterisk.

In the end you see how many parameters have been changed.

Do not forget to click the **SAVE TO EEprom** key!!

11.11.5 How Do You Download OPENPCS-Files via Terminal?

(bsl-version since May 2006)

Only binary files that are read via terminal can be downloaded!

Enable feature via setting user parameter 119 to 1199 (figure 11.50).

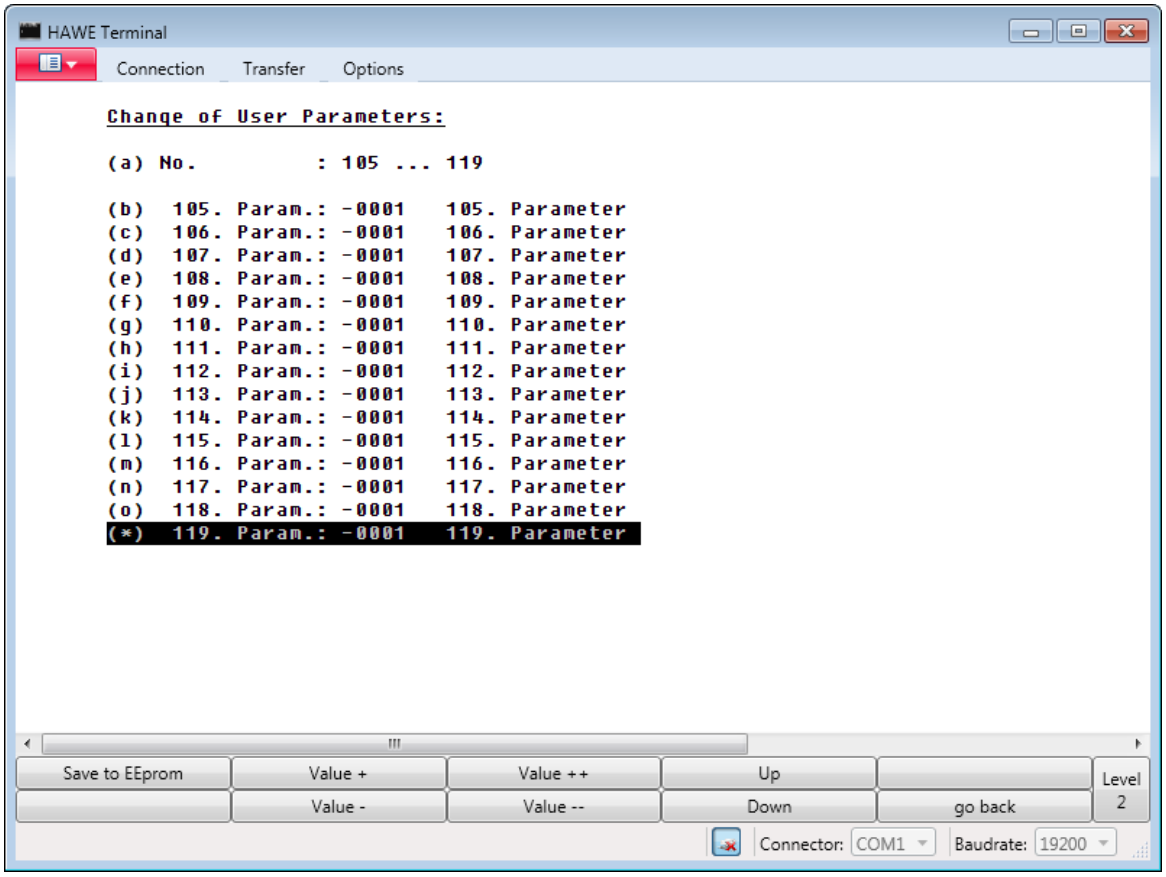


Figure 11.50.: Download OPENPCS-Files via Terminal

Log out, then go to **Transfer** → **Receive Binary File**, insert name of file, hit **OK**.

Transmission starts after some seconds, see bar at bottom.

Reception terminates automatically after 61 kB.

A file received like this can be downloaded to another PLVC:

Log out, hit 3 times the key “6”. PLVC will start to send CCC. Then you have 60 seconds time to select **Transfer** → **send binary file**. Insert a file name, and hit OK. PLVC will reset and restart with new OpenPCS-program.

11.11.6 How Do You Talk to HAWE-HMI?

Communication paramater settings for PLVC to communicate to HAWE-HMI.

HMI after reset always connects to the PLVC with address zero (**Parameters** → **Submenu 4: Communication** → **Digital Inputs a) Transmit**).

Also the baud rates must fit together: It can be set in HMI via **Terminal** → **Parameters** → **CAN_BAUD**.

11.11.7 How Does Remote Diagnostics for PLVC via Modem Work?

Always test before you sell to customer!!!!

The direct connection from computer to PLVC via RS232 and data-cable can be “extended” via two modems and a telephone line in between.

For this you need an analog modem on the side of the computer, connected to COM1 and an analog modem or cell-phone with built in analog modem on the PLVC side.

All these modems have to react on AT-commands and must support 19200 baud speed. With cell phone modems you have to make sure that the service supplier supports and enables data transmission!!

How to test whether a cell phone is capable of remote diagnostics:

1. Start terminal program
2. Connect cell-phone via data cable to computer
3. Type “AT” + ENTER
4. Cell phone should respond “OK”
5. Start second computer with terminal program
6. Connect analog modem to computer
7. Type “AT” + ENTER
8. Modem should respond “OK”
9. Select → **settings** → **phone number** and insert the cellphone number
10. Select → **phone** → **dial**
11. In cell phone display something like “incoming data” should appear
12. In cell phone terminal program “RING” should appear
13. In cell phone terminal program type in “ATA” + enter
14. After 30 seconds or so, communication message shoud appear and you should be able to type messages from one terminal program to the other.

15. Select → **dial** → **hangup**

After this test was successfull, connect the adapter cable of cell phone to PLVC. As this is a female connector you need an additional adapter, where

- Pin 2 goes to pin 1 of PLVC
- Pin 3 goes to pin 2 of PLVC
- Pin 5 goes to pin 3 of PLVC

Once more call the cell phone. After second ring, PLVC should take the connection, and LOGIN via terminal should be possible.

If PLVC does not take the connection, in spite of successful teste above, this can have the following reasons:

- Data cable/adapter not correct
- Baud rate not detected: once more re-power PLVC, set → **parameter** → **special parameter** → **modem** to 3
- 3wire connection is not ok for this kind of modem: Pins 4/7 of SUB-D connector have to be connected to 5V then.

“Tested” cell phones:

- Siemens S25, S35, S45, S55, S65 with original data cable.
- Nokia 6210, 6310. Unfortunately latest Nokias do not have analog modem with Rs232 output any more, but use USB.

12 Programming OpenPCS

OpenPCS is a programming software from infoTeam based on IEC 61131 for programmable logic controller. Basically, the customer can program his own control. For programming the software OpenPCS is needed, which is available at HAWE.

Together with the programming interface HAWE supplies specific, to the PLVC coordinated components (e.g. control of the proportional outputs, reading frequencies and more. . .).

12.1 Overview

The OpenPCS chapter consists of individual chapters:

Introduction: A short introduction into the most important tools of OpenPCS, using a short sample.

The project manager: Manual for the tool “Project Browser”.

POU-Editor: Manual for the tool “POU-Editor” for programming languages IL and ST.

Programming of PLVCs: Description of function blocks

Further sample programs: Some programs to show you the use of function blocks.

12.2 More Information

You can’t find what you’re looking for? There are other sources you can use:

- OpenPCS provides a “Help” function. In each tool, there is a menu item “Help”. It is easier to search for items in the electronic help than in this printed manual.
- Training on IEC 61131 is available from different sources. To get training covering the special features of PLVC and OpenPCS, HAWE Hydraulik offers in-house training seminars on this.
- Assistance with individual problems related to the use of OpenPCS is offered by HAWE.

12.3 Introduction

12.3.1 Basics

This part of the manual will give you an introductory overview over OpenPCS. With a simple example, you will learn how to create own projects, own programs, down to the very test with your controller.

A basic knowledge of Microsoft Windows and PLC programming is assumed.

12.3.2 Styles and Symbols

The notation “Project → New” is used to denote the selection of item *New* in menu *Project*. The arrow is used to mark individual steps you should follow.

12.3.3 Programming Example

When you have a new installation of OpenPCS you will get a sample program at startup simply by clicking **Open last project**. The same program will be used here. This chapter will describe the problem and the solution. The rest of this chapter will implement the solution with OpenPCS.

Problem:	A cylinder should be driven via joystick. There is an “off”-switch, which should disable the joystick and a “slow”-switch, which slows down the speed. Additional there is a horn switched by a switch.
Solution:	The solution might look like:

```
VAR †
  joy  AT %IW64.0:  int; ‡   (* Joystick analog input *)
  setp: int;
  prop: ACT_VALVE;  (* FB *)
  speed:  int;  (* The maximum speed *)
  enable_di AT %IB0.0:  bool; (* Enable switch *)
  slow_di AT %IB0.2:  bool; (* Digital input for slow down *)
  ini:  bool;
  horn_di AT %IB1.0:  bool;
  horn_do AT %QB3.2:  bool;
END_VAR

(* The following is only executed once at reset *)
if not ini then
  ini := true;
  (* Nothing to do *)
end_if;
```

```
19 (* Reset variables *)
   setp := 0;
21 speed := 1000;

23 if slow_di then
     speed := 200;  (* Means 20% of full speed *) §
25 end_if;

27 if enable_di then setp := joy; end_if;

29 (* Assignement of horn_di to horn_do. This will switch the horn trough horn_di*)
   horn_do := horn_di;
31
   (* Set the valves according to the variables *)
33 prop( CHANNEL := 10, SETPOINT := setp, OVERRIDE := speed );
```

† In contrast to traditional PLC programming languages, IEC 61131 requires that you declare all variables that you use.

‡ This line declares a variable of name joy of data type INT, to be mapped to hardware address %IW64.0.

§ Almost everywhere in IEC 61131 you can use comments to describe your programs.

Note: The available hardware addresses are strongly dependent on the control used.

12.3.4 Installing OpenPCS

Insert the HAWE-CD into your CD-ROM and read the readme.txt. Follow the instructions.

12.3.5 Starting OpenPCS

Start Windows and choose **Start** → **Programs** → **OpenPCS** → **OpenPCS** in the start-menu; the window project browser will be opened: If you select **Open last project**, one project was pre-selected, by selecting **Create new project** you see an empty window. Choose **Open last project** to load the HAWE sample program, which you can direct compile and run. You will see the screen shown in figure 12.1.

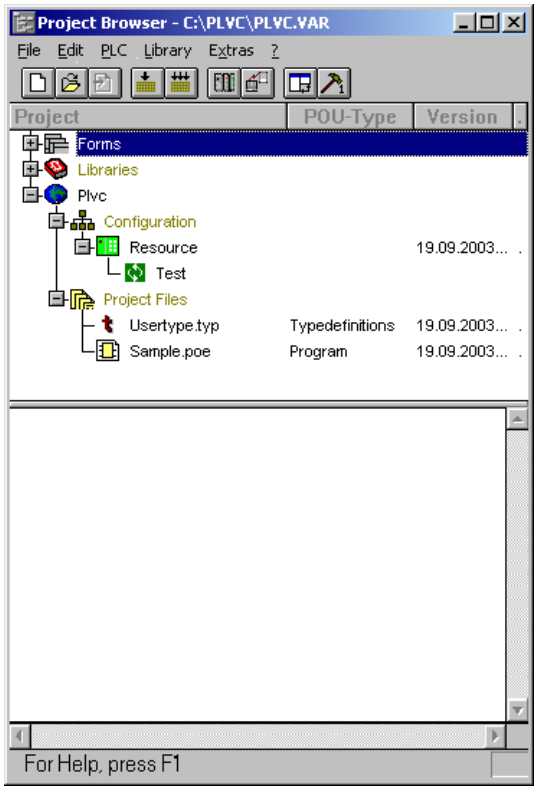


Figure 12.1.: OpenPCS Project Browser

With OpenPCS, you will structure your work in projects. It is very easy to reuse software within one project, and this is one of the great advantages of OpenPCS. Be sure to structure your work in a way to take best advantage of this feature:

- Avoid to use more than one project for related work. One machine, one network of CANopen nodes or even one plant might be one project.
- Use separate projects for unrelated work; this increases your overview and prevents you from mistakenly modifying wrong code.

The project browser is the file manager of OpenPCS. When you create files within an OpenPCS project, internal information is kept on these files. You cannot replace the project browser with a standard Windows file manager (like the Windows Explorer).

Double click on **Resource** in this box and answer the two dialogs with yes. If everything is correctly wired, you will see the “Test and Commissioning-Window” (figure 12.2).

Attention:

You must have the terminal closed, due to both programs use the same ressource!

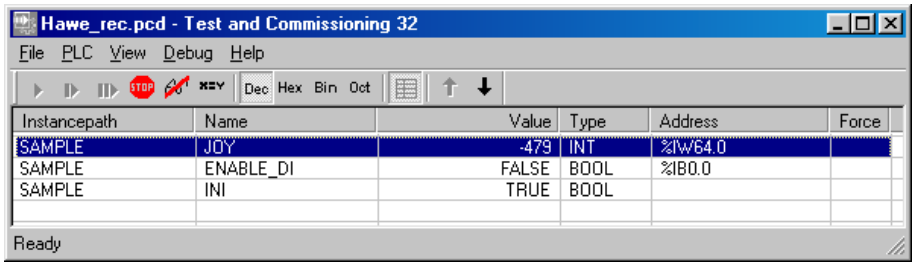


Figure 12.2.: OpenPCS Test and Commissioning

Starting and Stopping the Program

You may remote-control the program running in the PLC with the following commands (table 12.1):

	PLC STOP	Select *PLC STOP* to immediately stop the program.
	PLC Coldstart	Select *PLC Coldstart* to perform a cold start. All variables will be initialised to the initial value as programmed.
	PLC Warmstart	Select *PLC Warmstart* to initialise all normal variables, but keep the values of all variables programmed as RETAIN.
	PLC Hotstart	Select *PLC Hotstart* to continue execution where it stopped, not initialising any variable.

Table 12.1.: Remote-Control Commands for the Running Program

It is recommended to use *Coldstart*.

Variable Status

Watch Variable

To display and periodically update the value of selected variables, use the watch window in “Test and Commissioning”:

- Open the branches resource and sample of the T+C’s instance tree in the project browser. There you can find the variables which you want to monitor.
- Mark the variable which you want to monitor and double click it.

Choose “SPS → watch variable” or click right on a variable and choose “watch variable”. The choosen variabes will be shown in the “Test und Commissioning”-window.

Set Variable

For changing variables double click on the variable. In the new window (figure 12.3) you can enter the new value into the input field “Value”.

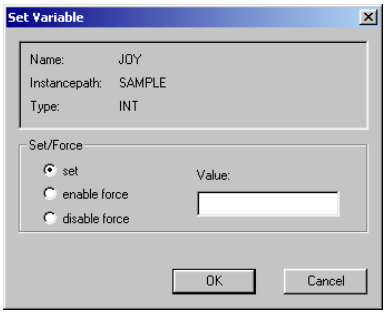


Figure 12.3.: OpenPCS - Set Variable

You cannot change the values of “IW” variables, since to they are updated by the inputs.

In the project browser select **Online** → **Go Offline** to leave the online-mode.

12.4 Project Manager

12.4.1 Introduction Project Manager

The tool “Project manager” is used for the management of projects and the files belonging to projects. It is similar to the well-known Windows Explorer, but has some additional functionality required by OpenPCS. This is why you should not try to replace the project manager by another file management tool; internal links are generated by the project manager automatically which are absolutely necessary for a correct work with OpenPCS. With the project manager you can structure your work into projects. A great advantage of OpenPCS is the reuse of blocks in other projects. Please note the following hint:

- Avoid to create several OpenPCS-projects for related work. One machine, one network of CANopen nodes or even one plant might be one project.
- Use separate projects for unrelated work; this increases your overview and prevents you from mistakenly modifying wrong code.

The project management window (see figure 12.4) consists of three parts:

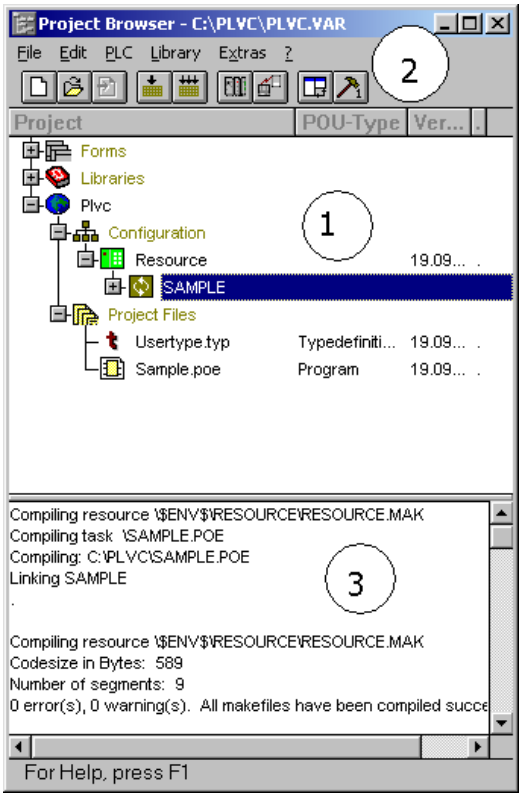


Figure 12.4.: Project Manager

1. **The project tree**
   All parts of a project are combined in a tree here.

2. **The menu panel and the toolbar** (table 12.2)

	Create new project
	Open an existing project
	Generate executable code
	Add file
	Edit resource
	Edit the task properties
	Add task to the selected resource
	Separate the desktop for applications

Continued on the next page. . .

. . . Continued from previous Page

Table 12.2.: Project Manager Panel/Toolbar

3. **The output window**
   Outputs of the compilation or the online operation are shown here.

12.5 Programming of PLVCs

12.5.1 Declarations

Datatypes

IEC 61131 defines a set of elementary datatypes, plus means to define derived datatypes. The keywords for datatypes may be written in lower or upper case.

Elementary data types are data types that are predefined and identified by keywords. The initialization value of a data type is set by the assignment operator ":=". If no initialization value is assigned, the value 0 is defined as default value for numeric data types.

Table 12.3 lists all elementary datatypes of IEC 61131 with their keyword and default initial value. Any variable defined to be of a given type will have this default initial value unless it is declared as being different for this individual variable:

Keyword	Datatype	Bits	Default
BOOL	Boolean value, can be 1 or 0, which is the same as "TRUE" and "FALSE".	1	0
SINT	Short Integer; values range from -128 to +127	8	0
INT	Integer; values range from -32768 to +32767	16	0
DINT	Double integer; values range from -2147483648 to +2147483647	32	0
USINT	Unsigned short integer; values range from 0 to 255	8	0
UINT	Unsigned integer; values range from 0 to 65 535	16	0
UDINT	Unsigned double integer; values range from 0 to 4294967295	32	0

Continued on the next page. . .

. . . Continued from previous Page

REAL	Real number	32	0.0
TIME	Time duration value	-	T#0s
STRING	Character string of variable length (for maximum length of strings, see capter 10.7.1)		“ ” (Empty string)
BYTE	Bitstring of 8 bits	8	0
WORD	Bitstring of 16 bits	16	0
DWORD	Bitstring of 32 bits	32	0

Table 12.3.: Elementary Datatypes of IEC 61131

Variables

Variables are names to represent memory locations which can hold different data values. There are two groups of variables:

- Variables mapped to physical (or logical) inputs, outputs or markers of a PLC, and
- Variables used only internally within the program to hold intermediate results

Each variable has a name, starting with a letter (a-z) or an underscore. The variable name may contain digits, upper and lowercase letters and underscores, but not space. The length of a variable name should not exceed 64 characters.

Directly represented variables

Directly represented variables are those variables, that under the control of the programmer are mapped to a specific input, output or memory address. Keyword AT is used to declare this, and the address is specified in a string starting with a percent sign (%).

Example:

Directly represented variables.

Declaration of a directly represented variable with a symbolic name.

```
1 VAR
    Input1 AT %IB0.0 : BOOL;
3 END_VAR
```

It is strongly recommended to use symbolic names for directly represented variables, as this eases rewiring to different addresses. The declaration cannot be saved either.

Directly represented variables are only allowed in a POU of type PROGRAM!

Syntax of the %...

The syntax of the string starting with the percent sign is:

% <Location-Mnemonic> <Size-Mnemonic> <Number>.<Number>

where **Location-Mnemonic** is one of

I	Input location
Q	Output location
M	Memory location

Table 12.4.: Location-Mnemonic

and **Size-Mnemonic** is one of

X	Single bit size
<none>	Single bit size
B	Byte (8bits) size
W	Word (16bits) size
D	Double word (32bits) size
L	Long word (64bits) size

Table 12.5.: Size-Mnemonic

Example

The Expression “%IB3.4” accesses the Input byte 4 of the third “module”. Having a look at the table Pin Description List PLVC41-G / PLVC41-4-G(7.8) the Expression uses the Digital Input IB3.4 of a PLVC4 Basic Device.

Variable types

All variables that you want to use in the instruction part of your POU have to be declared in the declaration part of this POU. There are different types of variables, depending on the keyword used to introduce the declaration section. Table 12.6 gives an overview:

Keywords	Usage
VAR	“Local Variable”, only accessible in the defining POU.
VAR_GLOBAL	Global Variable, accessible from all parts of a program. Any other POU wishing to access this variable has to declare this as external.
VAR_EXTERNAL	Declaration for a variable defined as global elsewhere.
VAR_INPUT	Input variable, only to be read from within the defining POU, and to be written by other POUs (typically the calling POU).
VAR_OUTPUT	Output variable, to be written by the defining POU and read by others (typically the calling POU)

Table 12.6.: Variable Types

Each declaration section starts with the respective keyword and ends with END_VAR or END_TYPE. Local and global variable declarations can be attributed with the following attributes:

Keywords	Usage
CONSTANT	Use CONSTANT to flag variables that should not be written by the program. Using this keyword makes your program more self-documenting and helps you spot programming errors, if unintentionally you tried to write this variable.
AT	Use AT to map variables to physical addresses

Table 12.8 lists which section of declarations is allowed in which type of POU (1=Allowed):

Section	FUNCTION	FUNCTION BLOCK	PROGRAM
VAR_INPUT	Allowed	Allowed	
VAR_OUTPUT		Allowed	
VAR_GLOBAL			Allowed
VAR_EXTERNAL		Allowed	
VAR	Allowed	Allowed	Allowed

Table 12.8.: Allowed Declarations by Type of POU

Instantiating Function Blocks

With IEC 61131, function blocks cannot be called; only instances of function blocks can be called. Before, the instance has to be declared like any other variable. To declare an instance, declare it like any other variable by using the function block name as the datatype.

```
1 Timer : TON;
```

This declares an instance “Timer” of function block TON. Inputs and outputs are parameterised with the call to an instance.

12.5.2 Instruction Part of a POU

Constants

With OpenPCS, there are two ways to use constant values:

1. declare a variable with attribute CONSTANT, and assign the constant value to this value as an initial value in the declaration,
2. write the constant as a literal constant directly in the instruction list.

Example:

Variable val1 shall be incremented by a constant value of 115, and stored into variable val2:

```
1 Val2 := val1 + 115;
```

Within a literal constant, underscores are allowed to increase readability. Such underscores have no meaning regarding the value of a constant.

Literal constants for some datatypes require a special prefix (table 12.9):

Constant Datatype	Example	Meaning
INT	-13 45165 or 45_165 +125	Integer -13 Integer 45165 (both) Integer 125
REAL	-13.12 123.45 0.123 -1.23E-3	Real -13,12 Real 123,45 Real 0,123 Real -0,00123
Dual number	2#0111_1110 or 126	126
Hexadecimal number	16#123 or 291	291
BOOL	0 and 1 TRUE and FALSE	Boolean TRUE and FALSE values
STRING	'ABC'	Character string ABC
TIME	T#12.3ms or TIME#12.3ms	Time duration of 12,3 milliseconds
	T#12h34m or TIME#12h34m	Time duration of 12 hours and 34 minutes
	T#-4m	Negative time duration of 4 minutes

Table 12.9.: Special Prefixes for Literal Constants Datatypes

Literal constants of datatypes `TIME` uses keywords (`TIME` or `T`) plus a hash sign "#".

Functions

A function provides one means of re-usability. A function may have more than one input, but it always has exactly one output value.

A function is always accessible to all POUs of a project, there is no declaration needed to be able to call a function. There are no instances of functions. To call a function, the function's name is used as an operator, the current result is always used for the first parameter, more parameters are passed to the function as operands after the function name, separated by commas.

The function returns its value by storing it into a variable having the name of the function (as PASCAL does). After the function call, the value returned by the function will be the current result.

Function cannot store values over a call, give the same input parameters, a function shall always return the same output value.

Example:

Function "Sum"

Sample function "Sum" (figure 12.5) will compute the sum of three arguments:

```
1 FUNCTION SUM : INT
  VAR_INPUT
3   A : INT;
    B : INT;        (* Declarations *)
5   C : INT;
  END_VAR
7   sum := a+b+c;
  END_FUNCTION
```

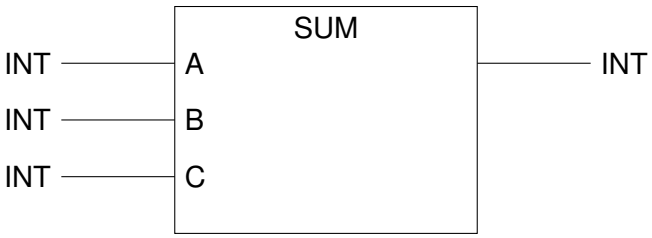


Figure 12.5.: Function SUM

Example:

Program “POU2”

Calling function SUM from above

```
VAR
  SUMMAND1 : INT := 1;
  SUMMAND2 : INT := 2;
  SUMMAND3 : INT := 3;
  RESULT : INT;
END_VAR
  result := sum(summand1, summand2, summand3);
```

A function may call other functions, but it may not instantiated nor call instances of function blocks, and function blocks are not allowed as parameters to functions. Besides defining your own userdefined functions, you can use many standard functions that come already with OpenPCS.

12.5.3 Function Blocks

Besides functions, function blocks are the second way of re-usability in IEC 61131 and OpenPCS. Other than functions, a function block may have more than one output and it may store values from one call to the next.

A function block may call other functions, it may instantiate other function blocks and invoke instances of function blocks.

The parameters of the function block are accessed like this:

```
FB.Param
```

A function block input may be declared as being edge detective.

IEC 61131 defines different ways to pass parameters to function blocks. The following example will demonstrate these:

Example:

Function block “average”

A simple function block to compute the average of two numbers, and its invocation from a program POU3 (figure 12.6):

```
1 FUNCTION_BLOCK average
  VAR_INPUT
3   Val1 : USINT;
    Val2 : USINT;
5 END_VAR
  VAR_OUTPUT
7   AvgVal : USINT;
  END_VAR
9   AvgVal := (val1+val2) /2
  END_FUNCTION_BLOCK
```

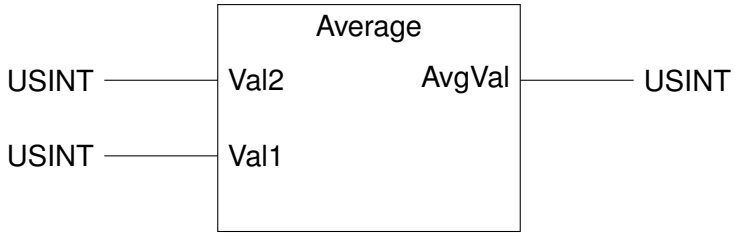


Figure 12.6.: Function Average

Example:

Program POU3

Instantiate function block “Average”, and two different invocations:

```
VAR
  Instance1_Name : Average;
  Instance2_Name : Average;
  MessVal1 AT %IB0.0 : USINT;
  MessVal2 AT %IB1.0 : USINT;
  Result1 AT %QB0.0 : USINT;
  Result2 AT %QB1.0 : USINT;
END_VAR
(* 1. Method of FB call *)
Instance1_Name (Val1:= MessVal1, Val2:= MessVal2);
result1 := Instance1_Name.AvgVal;

(* 2. Method of FB call *)
Instance2_Name.Val1 := MessVal1;
Instance2_Name.Val2 := MessVal2;
Instance2_Name ( );
Result2 := Instance2_Name.AvgVal;
```

Function block “Average” exists only once, but multiple instances of it may be declared with different names (here: “Instance1_Name” and “Instance2_Name”). Although not shown in this example, it is definitely possible to call one instance more than once.

Each instance will receive its own copy of all data associated with function block “Average”. Parameters are passed to the function block

- as arguments to the instruction, enclosed in parentheses.
- before the call by storing some value into the proper member variable (e.g. `Instance2\_Name.Val1 := Val1`).
- not at all, in which case the input will be left to the value it was assigned last, or the initial value.

12.5.4 Other Programming Languages

Besides Instruction List, OpenPCS offers other languages. On the one hand, these are the languages as defined by IEC 61131:

- Function block diagram (FBD)
- Structured text (ST)

HAWE doesn’t recommend the use of languages other than ST.

12.5.5 Functions (IEC 61131)

OpenPCS comes with a variety of standard functions and standard function blocks according to IEC 61131. Table 12.10 provides an overview:

Function	Purpose
*_TO_**	Type conversion
TRUNC	INT part of a real number
ABS	Absolute value
SQRT	Square root of a real number
SIN	Sinus of a real number
COS	Cosinus of a real number
TAN	Tangens of a real number
ASIN	Arcus sine of a real number
ACOS	Arcus cosine of a real number
ATAN	Arcus tangent of a real number
SHL	Shift left
SHR	Shift right
ROL	Rotate left
ROR	Rotate right

Table 12.10.: Functions (IEC 61131)

The following description of some standard functions shows the prototype of each function in graphical format, with datatypes and symbolic names of inputs and outputs as appropriate.

Example:

Prototype

The following textual prototype definition for standard function SHR...

```
FUNCTION SHR : ANY_BIT
  VAR_INPUT
    IN : ANY_BIT;
    N : UINT;
  END_VAR
END_FUNCTION
```

...corresponds to the graphical prototype in figure 12.7:

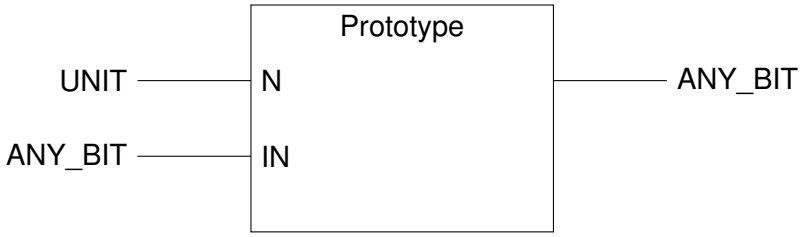


Figure 12.7.: Function Prototype

Overloaded Functions

Some functions, like SHR shown above, can be applied not only to one datatype of operands, but to a variety of datatypes. For function “SHR”, this is shown by giving the generic datatype ANY_BIT as the datatype for input argument “IN”.

Only manufacturer defined functions may be overloaded, this is not possible for userdefined functions.

12.5.6 ABS - Absolute Value

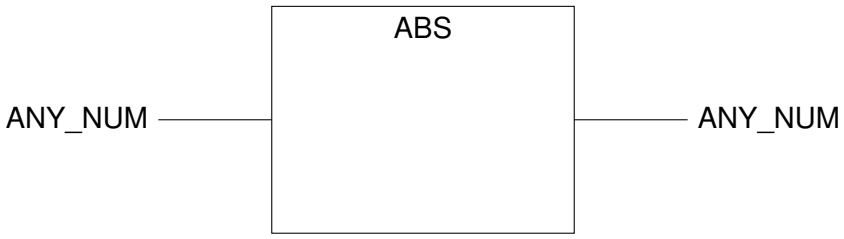


Figure 12.8.: Function ABS

ABS will compute the absolute value of the input.

Example:

```
(* joystick smaller than +/- 10% *)
if (abs(joystick) <100) then
  ...
end_if;
```

12.5.7 Trigonometric Functions (ACOS, ASIN, ATAN, COS, SIN)

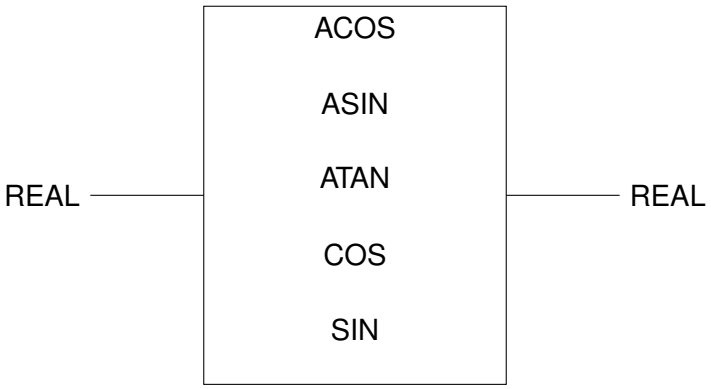


Figure 12.9.: Trigonimetric Function

Description

- ACOS will compute the arcus cosine of the input.
- ASIN will compute the arcus sine of the input.
- ATAN will compute the arcus tangent of the input.
- COS will compute the cosine of the input.
- SIN will compute the sine of the input.

Example:

You need the radius of a crane where you have the angle and the length as analog inputs.

```
R := real_to_int (int_to_real (l)*cos(int_to_real(alph)*2*PI/3600));
```

12.5.8 MAX

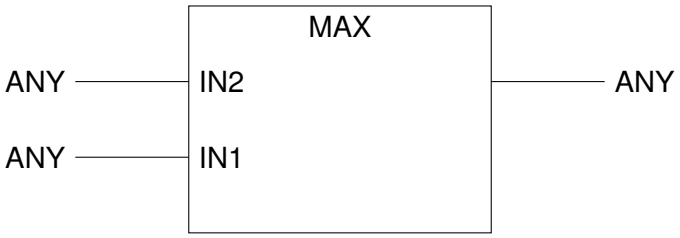


Figure 12.10.: Function MAX

Inputs:

- **IN1** First input, provided by current result
- **IN2** Second input

All inputs have to have the same datatype, and the output will have the same datatype as all inputs.

Description

MAX will deliver the greatest input-value to current result.

Example:

```
1 Result := max(100,200);   (* result will be 200 *)
```

12.5.9 MIN

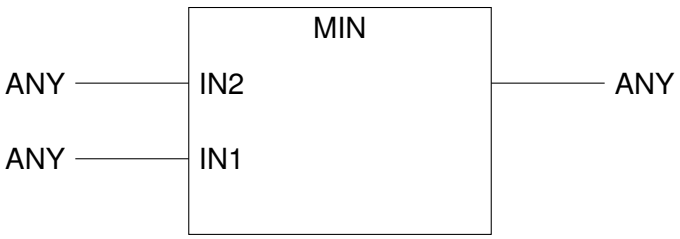


Figure 12.11.: Function MIN

Inputs:

- **IN1** First input, provided by current result
- **IN2** Second input

All inputs have to have the same datatype, and the output will have the same datatype as all inputs.

Description

MIN will deliver the smallest input-value to current result.

12.5.10 MOD

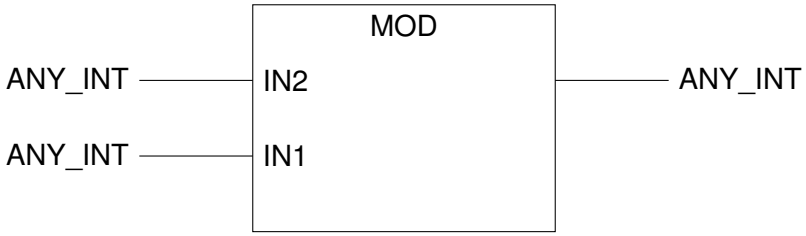


Figure 12.12.: Function MOD

Inputs:

- **IN1** Dividend, provided by current result
- **IN2** Divisor

All inputs have to have the same datatype, and the output will have the same datatype as all inputs.

Description

MOD will deliver the remainder of the division of IN1 by IN2.

12.6 HAWE Function Blocks

12.6.1 Frequently Used Function Blocks

ACT_VALVE

ACT_VALVE act_valve(CHANNEL:=,SETPOINT:=,OVERRIDE:=);	
Inputs	
INT	CHANNEL *Channel to activate. Possible values:* *00-15: Local proportional outputs* *16-31: Local PWM outputs* *32-33: Programmable 0-10V outputs* *34-50: CAN connected outputs*
INT	SETPOINT *Possible values: -1000 to +1000 (-100% to +100%)*
INT	OVERRIDE *Possible values: 0 to 1000 (0 to 100%)*

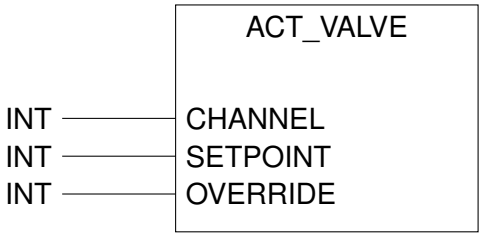


Figure 12.13.: Function ACT_VALVE

Parameters:

- *CHANNEL* is the appropriate output
- *SETPOINT* is the adjusted setpoint
- *OVERRIDE* indicates a possible decrease of the setpoint

Description

- The function block “ACT_VALVE” generates a current-value by using the incoming values and the valve-parameters. This current-value is then given to the declarated CHANNEL.
- The valve-parameters can be found in the Terminal programm in category “Prop. Valves”.
- The formula for calculating is:
  $\frac{SETPOINT}{1000} \cdot \frac{OVERRIDE}{1000} \cdot (IAMX - IAMN) + IAMN$ for $SETPOINT > 0$, and
  $\frac{-SETPOINT}{1000} \cdot \frac{OVERRIDE}{1000} \cdot (IBMX - IBMN) + IBMN$ for $SETPOINT < 0$.
- The second coil is supplied with standby current IPR.
- Independent of the given formulas both coils get supplied with IPR if $SETPOINT = 0$.

Example: Adressing proportionalvalve 3 with Joystick. Precise controlling for small deflection, IAMX/IBMX for maximum positive/negative deflection.

Variable declaration:

```
VAR
  prop: ACT_VALVE;
  joy AT %IW104.0: int;
END_VAR
```

Program:

```
PROP( CHANNEL:=2,
      SETPOINT:=joy,
      OVERRIDE:=joy
  );
```

Depicting the deflection of the Yoystick in direction of the x-axis and the proportionalvalve-current in direction of the y-axis you get, referring to the example, a progressive curve.
That causes the effect, where you can control the proportionalvalve very precisely within smaller current-ranges but achieve IAMX with the maximum deflection.

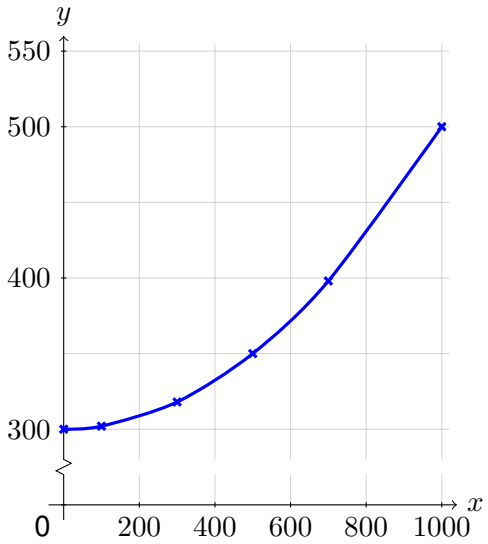


Figure 12.14.: Curve for IAMX=500, IAMN=300;

GET_EE

Loading a persistent parameter from EEPROM.

GET_EE get_ee(CHANNEL:= | :=EE_VAL);	
Inputs	
INT	CHANNEL *Channel to read. Possible values:* *00-99: Serial EE used for user parameters* *100-149: Parameter CAN display 0-49* *150: Active picture CAN display 0-49* *151: Active notice CAN display 0.49* *500-599: hidden internal parameters* *2000-2240: (only PLVC2) Profibus Byte 0-240 as SINT* *4000-4240: (only PLVC2) Profibus Word 0-240 as INT* *3000: Menu small Display, main branch* *3001: Menu small Display, 2nd branch* *3002: Menu small Display, 3rd branch* *3003: actual branch in the smaller display's tree* *10000-10383: User Parameter 0-383 for devices with more than 100 User parameters.*
Outputs	
INT	EE_VAL *is the read value*

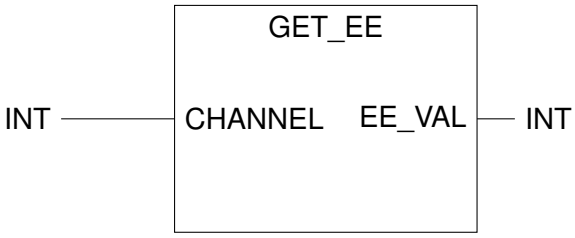


Figure 12.15.: Function GET_EE

Parameters:

- *CHANNEL* is the channel you want to read.
- *EE_VAL* is the read value.

Description:

You can store up to 100 user parameter within the PLVC. These user parameters will retain even if the PLVC is turned off.

These parameters can be accessed via GET_EE (read) and PUT_EE (write).

Attention:

Keep in mind that the lifetime of a EEPROM is 100000 to 1 million write cycles.

Example: Read user parameter 10

Variable declaration:

```
VAR
  para: GET_EE;
  value:  INT;
END_VAR
```

Program:

```
para(CHANNEL :=10);
value:= para.ee_val;
```

PUT_EE

Saving a persistent parameter in EEPROM.

PUT_EE put_ee(CHANNEL:=,EE_VAL:=);	
Inputs	
INT	CHANNEL *number of the channel to write* *00-99: Serial EE used for user parameters* *100-149: Internal variables of a graf-syteco-display* *500-599: Hidden internal parameter* *2000-2240: (Only PLVC2) profibus byte 0-240 as SINT* *4000-4240: (Only PLVC'2) profibus word 0-240 as INT*
INT	EE_VAL *Value to save*

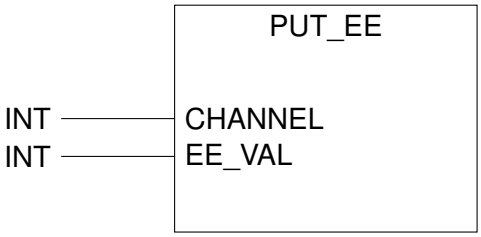


Figure 12.16.: Function PUT_EE

Parameters:

- *CHANNEL* is the number of the channel you want to write.
- *EE_VAL* is the value you save.

Description:

The PLVC does not support “Retain Variables” in the strict sense. The programmer has however 100 so called user parameters which will be saved and can be read after hardware reset.

Attention:

Note: The writability is limited to 100000 cycles!! Therefore also see 12.6.1 .

PUT_EE writes the value of EE_VAL to the EEPROM in user parameter CHANNEL.

Example: Save value 33 to user parameter 10

Variable declaration:

```
VAR
  save: PUT_EE;
  value:  INT;
END_VAR
```

Program:

```
value:= 33;
save (CHANNEL :=10 , EE_VAL := value);
```

12.6.2 Initialize

The following functionblocks common run once after the SPS was started, to define specific configurations.

I_INI

Debounce Digital Inputs

I_INI i_ini(CHANNEL:=,UP:=,DN:=);	
Inputs	
INT	CHANNEL *number of the digital input* *0-63 (%IB0.0-%IB7.7)*
INT	UP (positive)
INT	DN (positive)

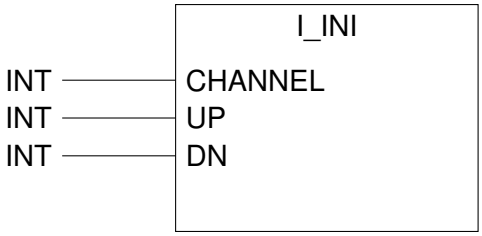


Figure 12.17.: Function I_INI

Description of the parameters:

- *CHANNEL* is the number of the digital input.
- *UP* is the elapsed time (in 10ms) until a rising edge is passed to the PLC.
- *DN* is the elapsed time (in 10ms) until a falling edge is passed to the PLC.

Example: Debounce input %IB5.1 for a rising edge of 100ms

Variable declaration:

```
VAR
  dig_ini : I_INI;
END_VAR
```

Program:

```
dig_ini (CHANNEL:=41, UP := 10, DN := 0); (* Channel 41 = IB5.1 *)
```

Q_INI

Configuring Digital Outputs

Attention:

Generally digital outputs should only be configured with the terminal program/visual tool. There you will find all settings you can alter by Q_INI. Compared to the function block configuring with the terminal program/visual tool requires no additional computing time.

One special feature of the digital outputs of the PWM module is the possibility to drive the outputs with a PWM signal. From the programmer's point of view the output remains as a digital output. Its behaviour can be changed with the Q_INI function block.

Q_INI (CHANNEL:=,TV1:=,TV2:=,TIM1:=,TIM2:=,DFQ:= | :=OK);	
Inputs	
INT	CHANNEL *Number of the digital output. Possible values:* *0-15*
INT	TV1 *Possible values: 0,5,10,...,95,100*
INT	TV2 *Possible values: 0,5,10,...,95,100*
INT	TIM1 *Possible values: 0-255*
INT	TIM2 *Possible values: 0-255*
INT	DFQ *Possible values: 0,1,2*
Outputs	
INT	OK (if parameter within valid range)

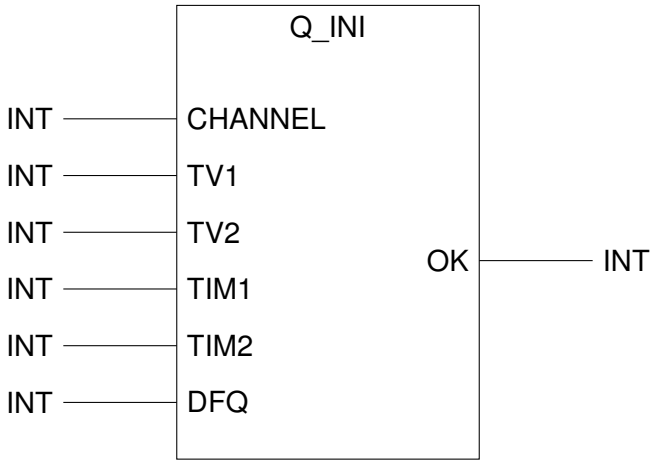


Figure 12.18.: Function Q_INI

Description of the parameters:

- *CHANNEL* Number of the digital output.
- *TV1* = Ratio 1.
- *TV2* = Ratio 2.
- *TIM1* Time in s/100 to get from TV1 to TV2.
- *TIM2* Time in s/100 to get from TV2 to TV1.
- *DFQ* Dither frequency of the PWM signal 0=50Hz, 1=100Hz, 2=200Hz.

The behaviour of the output is configured by the adjustment of parameter TV1 and TV2.

Mode 1: TV1 > TV2: The output works like an on/off output with energy-saving mode. First TV1 will be reached. After time TIM1 the output will be switched to a level configured with TV2. By resetting the output it will be disabled immediately.

Mode 2: TV2 > TV1: The output works like a proportional output („QuasiProp“). After enabling the output, the level will be TV1. Within time TIM1 it will ramp to level TV2. By resetting the output it will ramp to level TV1 within time TIM2.

Example: A “common” digital output at %QB0.6

Variable declaration:

```
VAR
  out_ini : Q_INI;
END_VAR
```

Program:

```
out_ini (CHANNEL :=6, TV1 :=100, TV2 :=100, TIM1 :=0, TIM2 :=0, DFQ :=0);
```

Example: Digital output at %QB1.0 with energy-saving mode after 1 sec., 100 Hz

Variable declaration:

```
VAR
  out_ini : Q_INI;
END_VAR
```

Program:

```
out_ini (CHANNEL :=8, TV1 :=100, TV2 :=50, TIM1 :=10, TIM2 :=0, DFQ :=1);
```

Example: Digital output at %QB1.1 “QuasiProp”, minimum PWM 40%, maximmum PWM 95% within 2 sec., drop to 40% within 0.5 sec., then switch off, 100 Hz:

Variable declaration:

```
VAR
  out_ini : Q_INI;
END_VAR
```

Program:

```
out_ini (CHANNEL :=9, TV1 :=40, TV2 :=95, TIM1 :=20, TIM2 :=5, DFQ :=1);
```

ANA_INI

Configuring Analog Inputs

Attention:

Generally analog inputs outputs should only be configured with the terminal program/visual tool. There you will find all settings you can alter by ANA_INI. Compared to the function block configuring with the terminal program/visual tool requires no additional computing time.

Purpose of configuring an analog input is to get a filtered, ramped, converted and normalized value during run-time. Additionally the input can be checked for cable break and short circuit.

The most comprehensive adjustment is needed to configure a joystick. The joystick can be configured by the following parameters:

- Error detection over 9.5V and under 0.5V
- 0 to 100% between 5.5V and 9.5V

- Death band between 4.5V and 5.5V
- 0 to -100% between 4.5V and 0.5V

Analog inputs configured as poti, joystick or angle should be powered with the 5V output of the PLVC41. The measured value is calculated proportional to this output.

All other types are interpreted as absolut values.

Furthermore all analog values can be filtered and converted to other values (units). The values ±1000 and ±100% are of special importance.

ANA_INI ana_ini(CHANNEL:= , TYP:= , MXXP:= , MXP:= , MNP:= , MNN:= , MXN:= , MXXN:= , PXP:= , PNP:= , NNP:= , NXP:= , FILT:= | :=OK)	
Inputs	
INT	CHANNEL *Select the channel you want to initialize. Possible values:* *0-23: analog values of the PLVC* *24-32: analog values of the radio control*
INT	TYP *Type of the input (poti, joystick, 4-20mA, 0-10V or angle)*
INT	MXXP *Value for cablebreak detection positive side*
INT	MXP *value for 100%*
INT	MNP *Value for minimum at positive side*

Continued on the next page. . .

. . .continued from previous page

INT	MNN *Value for minimum at negative side*
INT	MXN *Value for -100%*
INT	MXXN *Value for cablebreak detection negative side*
INT	PXP *Feedback of MXP*
INT	PNP *Feedback on MNP*
INT	NNP *Feedback on MNN*
INT	NXP *Feedback on MXN*
INT	FILT *Time constant of the digital filter*
Outputs	
INT	OK *all parameters are in a valid range*

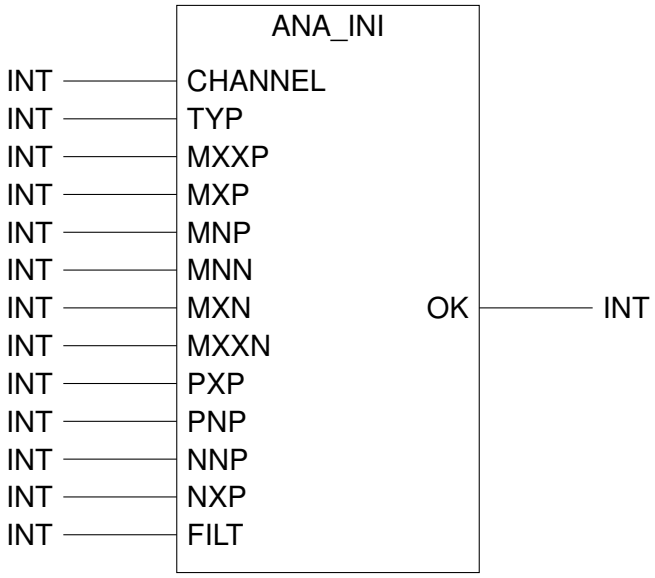


Figure 12.19.: Function ANA_INI

Description of the parameters:

- *CHANNEL* Number of the analog input (channel 0-2 = analog values of the PLVC, channel 24-32 analog values of the radio control)
- *TYP* Type of the input (poti, joystick, 4-20mA, 0-10V or angle)
- *MXXP* Value for cablebrak detection positive side
- *MXP* Value for 100 percent
- *MNP* Value for minimum at positve side
- *MNN* Value for minimum at negative side
- *MXN* Value for -100 percent
- *MXXN* Value for cablebrak detection negative side
- *PXP* Feedback of MXP
- *PNP* Feedback of MNP
- *NNP* Feedback of MNN
- *NXP* Feedback of MXN
- *FILT* Time constant of the digialt filter in 10ms steps. (Even FILT = filtered value is ramped, otherwise the ramped value is filtered!)

Note:

The modes poti and joystick are evaluated ratiometric (proportional to the supply voltage of the potentiometer output). The other modes are evaluated absolute.

General value of the parameters: MXXN $<$ MXN $<$ MNN $<$ MNP $<$ MXP $<$ MXXP

Only for mode joystick all parameters are needed. By using poti or 4-20mA, MNN and MXN are not used.

By setting MXXP to 1000 and MNX to 0 cable break detection is disabled. Parameter PXP, NXP respectively PNP and NNP are per default set to 1000 resprectively 0. It is possible to get “physical” values by choosing other values.

Example: Rotation speed with poti

Variable declaration:

```
VAR
  ana: ANA_INI;
END_VAR
```

Program:

```
ana(CHANNEL :=10,
  TYP :=0,
  MXXP :=950,
  MXP :=900,
  MNP :=100,
  MNN :=0,        (* Not relevant *)
  MXN :=0,        (* Not relevant *)
  MXXN :=50,
  PXP :=2000,     (* 2000Hz *)
  PNP :=0,        (* 0Hz *)
  NXP :=0,        (* Not relevant *)
  NNP :=0,        (* Not relevant *)
  FILT :=4);
```

Example: Pressure transducer DT2-2 (4-20mA)

Variable declaration:

```
VAR
  ana: ANA_INI;
END_VAR
```

Program:

```
ana(CHANNEL :=10,
  TYP :=2,
  MXXP :=950,
  MXP :=880,      (* Equates to 20mA *)
  MNP :=176,      (* Equates to 4mA *)
  MNN :=0,        (* Not relevant *)
  MXN :=0,        (* Not relevant *)
  MXXN :=90,
  PXP :=250,      (* 250 Bar *)
  PNP :=0,        (* 0 Bar *)
  NXP :=0,        (* Not relevant *)
  NNP :=0,        (* Not relevant *)
  FILT :=4);
```

RAMP_INI

Configuring Ramps

Attention:

Generally ramps should only be configured with the terminal program/visual tool. There you will find all settings you can alter by RAMP_INI. Compared to the function block configuring with the terminal program/visual tool requires no additional computing time.

RAMP_INI ramp_ini(CHANNEL:=,PU:=,PD:=,NU:=,ND:=,DIST:=,NR:=);	
Inputs	
INT	CHANNEL *Select the channel you want to initialize* *0-31: Analog inputs* *32-47: Current outputs*
INT	PU *Ramp time in 1/10 sec if value increases in positive range*
INT	PD *Ramp time in 1/10 sec if value decreases in positive range*
INT	NU *Ramp time in 1/10 sec if value increases in negative range*
INT	ND *Ramp time in 1/10 sec if value decreases in negative range*
INT	DIST *Factor of ramp length (default 1000, lower values result in longer ramp times, higher values result in shorter ramp times)*
INT	NR *Number of the parameter set that should be used (can only be adjusted with the terminal program or visual tool)*

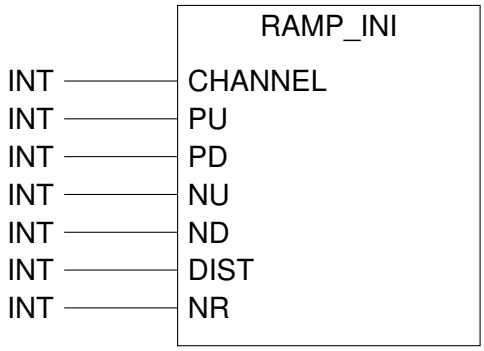


Figure 12.20.: Function RAMP_INI

Description of the parameters:

- *PU* Ramp time in 1/10 sec if value increases in positive range
- *PD* Ramp time in 1/10 sec if value decreases in positive range
- *NU* Ramp time in 1/10 sec if value increases in negative range
- *ND* Ramp time in 1/10 sec if value decreases in negative range
- *DIST* Factor of ramp length (default 1000, lower values result in longer ramp times, higher values result in shorter ramp times)
- *NR* Number of the parameter set that should be used (can only be adjusted with the terminal program or visual tool)

Note:

- The maximum ramp time value is 32000 (equates to 5 minutes and 20 seconds, DIST set to 1000).
- For an analog input set to poti mode (and all other inputs with only positives values) parameters NU and ND are not needed.
- All analog In- and Outputs have adjustable ramps, each of them with two selectable parameter sets. You can switch between the two parameter sets “hitchless” during run-time. For reasons of simplification the second parameter set for analog values is chosen by “digital outputs”:

  %QB17.0 . . . %QB17.7 for ramps **%IW24.0** to **%IW38.0** (Basic device)

  %QB18.0 . . . %QB19.7 for ramps **%IW40.0** to **%IW70.0** (Extension)

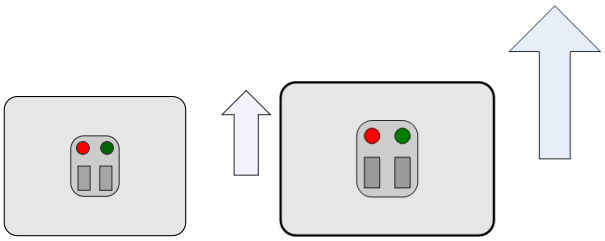
  %QB20.0 . . . %QB20.7 for ramps **%IW72.0** to **%IW86.0** (Radio)

  %QB21.0 . . . %QB22.7 for ramps **32 . . . 47** (current outputs 0 . . . 15)

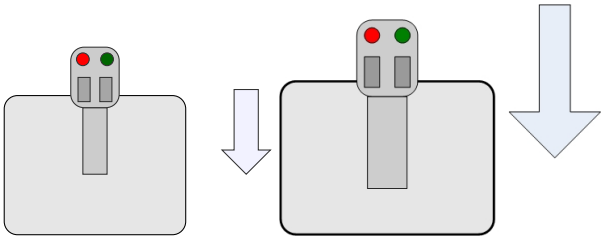
  %QB23.0 . . . %QB24.7 for ramps **%IW88.0** to **%IW118.0** (analog CAN nodes)

- If DIST is set to 100 all ramp times must be multiplied with 10.
- To clarify parameters PU, PD, NU and ND look at the shematic diagram of a joystick. The arrow shows the direction of movement.

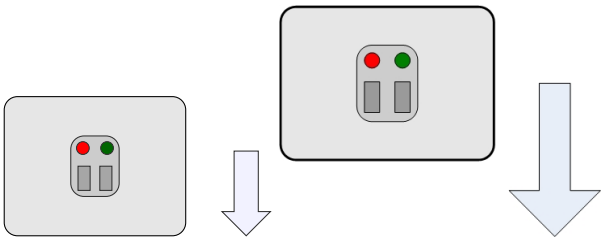
By **moving from zero position forward** ramp time set in **parameter PU** is valid.



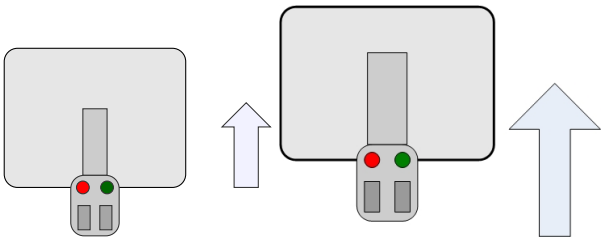
By **moving from ahead to zero postione** ramp time set in **parameter PD** is valid.



By **moving from zero position backwards** ramp time set in **parameter NU** is valid.



By **moving from back to zero position** ramp time set in **parameter ND** is valid.



FQ_INI

Initializing the frequency inputs.

The PLVC consists of three accurate frequency inputs. The resolution of these inputs can be adjusted.

NOTE

Each channel used must be initialized with FQ_INI.

FQ_INI fq_ini(CHANNEL:=,EXPONENT:=,PPU:=,ABT:=,);	
Inputs	
INT	CHANNEL *Select the channel you want to use. Possible values: 0-3*
INT	EXPONENT *Adjust the resolution (see FQ_READ). Possible values: -10 to +10*
INT	PPU *Pulses per Turn (for RPM). Possible values: 1-30000*
	ABT *not used*

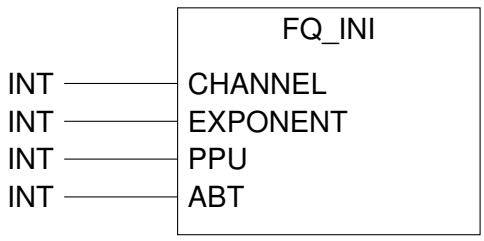


Figure 12.21.: Function FQ_INI

Description of the parameters:

- *CHANNEL* Choosen frequency channel
- *EXPONENT* Set resolution, following the rule $[Hz] \cdot 2^{EXPONENT}$. Value of 0 provides the result in Hertz, -1 provides half Hertz, -2 provides quarter Hertz, +1 double Hertz, and so on.

- *PPU* Factor pulses per turn (see also output UPM in FQ_READ (12.6.4))
- *ABT* Currently not used

Note:

- Frequency Inputs are generally controlled by a rotating disc which gives several pulses per each turn.
- The number of pulses can be entered in functionblock FQ_INI as PPU
- Via the number of pulses per rotation and the period from one to another pulse the rotational speed can be calculated. The functionblock FQ_READ displays that speed in turns per minute.

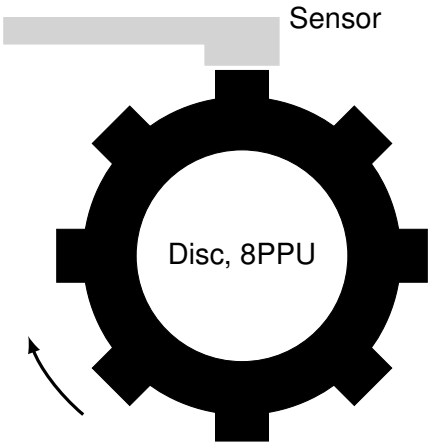


Figure 12.22.: Simplified sketch of a disc giving pulses to a sensor

An example to this functionblock is given with the functionblock FQ_READ (12.6.4).

POS_INI

Configure Frequency Inputs as Counter

Frequency inputs can be configured as a counter to represent a certain position.

POS_INI pos_ini(CHANNEL:=,FAK1:=,FAK2:=,OFFS:=,MODE:=);	
Inputs	
INT	CHANNEL *frequency channel you want to configure. Possible values: 0-2*
INT	FAK1 *Multiplier for POS1; 10000 ≙ 1.0*
INT	FAK2 *Multiplier for POS2; 10000 ≙ 1.0*
DINT	OFFS *Adjust offset for the counter*
INT	MODE *Bit 0: Counter active* *BIT 1: Up/down, else only up (for channel 0)* *Bit 3: Automatic zero setting*

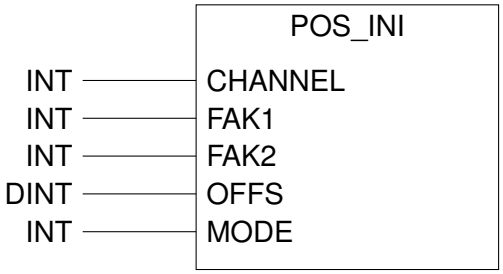


Figure 12.23.: Function POS_INI

Description of the parameters:

- *CHANNEL* Choosen frequency channel

- *FAK1* Multiplier for POS1 (10000⇒1.0)
- *FAK2* Multiplier for POS2 (10000⇒1.0)
- *OFFS* Adjust offset for the counter
- *MODE* Set mode

Example: Configure counter

Assume you get 10000 pulses per 360 degrees and you need the result after a range of time not in pulses but the angle of rotation, the disc covered. Therefore you have to generate a conversion factor. We will now generate a factor, which computes an angle of rotation in $\frac{1}{10}^\circ$. $\frac{x}{10000}$ is the number of rotations. This value multiplied by 3600 gives the rotation in $\frac{1}{10}^\circ$

$$\frac{x}{10000} \cdot 3600 = 0,36x.$$

That means that the measured number of pulses has to be multiplied by 0,36. Accordingly FAK1 has to be set to 3600. If you now call POS_READ.pos1 you see $\frac{1}{10}^\circ$. When you leave FAK2 at 10000 you can read the pulses (raw value) over POS_READ.pos2.

Variable declaration:

```
VAR
  posi: POS_INI;
END_VAR
```

Program:

```
posi( CHANNEL :=0,
      FAK1 :=3600,
      FAK2 :=10000,
      OFFS :=0,
      MODE :=1);
```

In chapter POS_READ(12.6.4) this example is extended by another functionblock, namely POS_READ.

VALVE_INI

Configuring Proportional Output

Attention:

Generally valves should only be configured with the terminal program/visual tool. There you will find all settings you can alter by VALVE_INI. Compared to the function block configuring with the terminal program/visual tool requires no additional computing time.

Depending on the configuration of the PLVC you get up to 16 proportional outputs which can also be used to control twin coils.

VALVE_INI valve_ini(CHANNEL:=,IAMX:=,IAMN:=,IBMX:=,IBMN:=,DFRQ:=,DAMP:=,MODE:=,R20:=,DOUB:=,IPR:= | :=OK);	
Inputs	
INT	CHANNEL *Chosen channel, Possible values:* *0-15: local proportional valve outputs* *16-31: local PWM (quasi prop)* *32-33: local 0-10V outputs* *34-67: only with external PLVC2* *68-101: only with external PLVC2*
INT	IAMX *maximum current A. Possible values: 0-2200mA*
INT	IAMN

Continued on the next page. . .

. . .continued from previous page

	minimum current A. Possible values: 0-1200mA
INT	IBMX *maximum current B. Possible values: 0-2200mA*
INT	IBMN *minimum current B. Possible values: 0-2000mA*
INT	DFRQ *dither frequency. Possible values: 25Hz-200Hz*
INT	DAMP *dither amplitude. Possible values: 0-500 permil*
INT	MODE *Reserved*
INT	R20 *cold resistance. Possible values: 2Ω-35Ω*
INT	DOUB *twin coil. Possible values: 1=yes, 0=no*
INT	IPR *standby current. Possible values: 0-2000mA*
Outputs	
INT	OK

Continued on the next page. . .

. . .continued from previous page

	if parameters are in a valid range

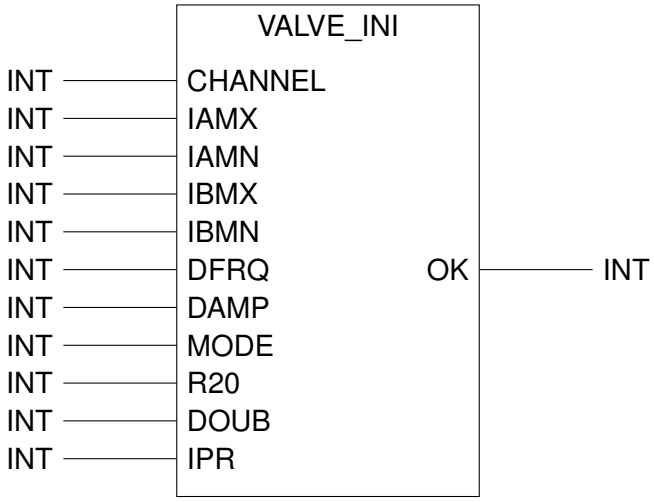


Figure 12.24.: Function VALVE_INI

Description of the parameters:

- *CHANNEL* Choosen channel
- *IAMX* Maximum current from side A in mA
- *IAMN* Minimum current from side A in mA
- *IBMX* Maximum current from side A in mA
- *IBMN* Minimum current from side A in mA
- *DFRQ* Dither frequency in Hz
- *DAMP* Dither amplitude in $\%_o$
- *MODE* (Currently not used)
- *R20* Resistance of the solenoid in Ω
- *DOUB* Single coil or twin coil
- *IPR* Standby current in mA

Note:

Generally all parameters should be adjusted using the terminal program of the visual tool.

If you want to configure a twin coil use an even channel and set DOUB to 1. You dont have to configure the next uneven channel.

If you configure a single coil parameters IBMX and IBMN are irrelevant.

Check if: IAMX $\geq$ IAMN $>$ IPR; IBMX $\geq$ IBMN $>$ IPR

Special configuration:

Set DOUB to 0, IAMN to 0 and IPR< 0. Current control reacts with a setpoint>0 like an on/off solenoid (100% PWM).

After (-IPR*10ms) the current will be reduced to IAMX.

Example: Configure proportional output 2 and 3 as a twin coil

Variable declaration:

```
VAR
  prop_ini: VALVE_INI;
END_VAR
```

Program:

```
prop_ini( CHANNEL :=2,  (* Prop. valve 2 *)
      IAMX :=580,   (* Imax A side 580mA *)
      IAMN :=280,   (* Imin A side 280mA *)
      IBMX :=610,   (* Imax B side (Prop. valve 3) 610mA *)
      IBMN :=310,   (* Imin B side (Prop. valve 3) 310mA *)
      DFRQ :=50,    (* Dither frequency 50Hz *)
      DAMP :=480,   (* Dither amplitude 48% *)
      MODE :=0,
      R20 :=27,   (* Resistance 27 Ohm *)
      DOUB :=1,   (* Valve 2 and 3 as twin coil *)
      IPR :=0);   (* Iprep (standby current) 0mA *)
```

12.6.3 Further Function Blocks

ANZ_7SEG

ANZ_7_SEG anz_7_seg(status:=);	
Inputs	
USINT	status

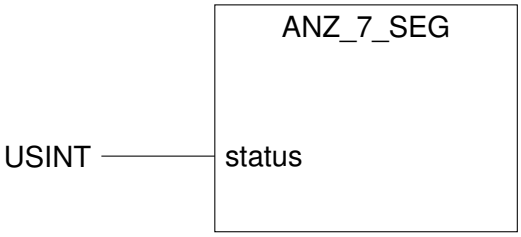


Figure 12.25.: Function ANZ_7_SEG

AVERAGE

Calculates the average of the last inputs.

AVERAGE average(INPUT:=,LENG:= | :=AVR);	
Inputs	
INT	INPUT *The input value*
INT	LENG *The number of included values. Possible values: 0-32*
Outputs	
INT	AVR *Average of the last inputs*

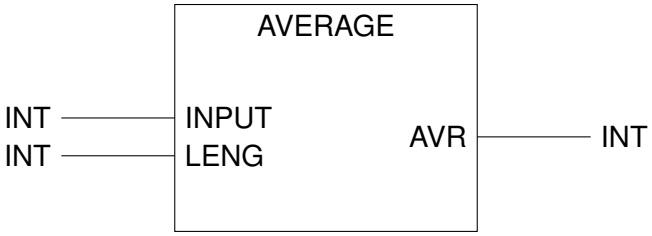


Figure 12.26.: Function AVERAGE

Inputs:

INPUT	The input value
LENG	1-32: The number of included values

Table 12.14.: AVERAGE Inputs

Example: AVERAGE called 5 times with INPUT:=100 and 5 times with INPUT:=200

Variable declaration:

```
VAR
avrg  : AVERAGE;
mean:  INT;
END_VAR
```

Program:

```
avrg (INPUT:=100);
avrg (INPUT:=100);
avrg (INPUT:=100);
avrg (INPUT:=100);
avrg (INPUT:=100);
avrg (INPUT:=200);
avrg (INPUT:=200);
avrg (INPUT:=200);
avrg (INPUT:=200);
avrg(INPUT:=200, LENG:=10);
mean:= avrg.AVR;
```

If AVERAGE has been called 5 times with INPUT:=100 and 5 times with 200, AVR will be 150. After another four calls with 200 it will be 190.

CAN_VALVE

PSL_CAN

Continued on the next page. . .

. . .continued from previous page

psl_can(CHANNEL:=, SETP:=, OVERR:=, RAMP1_LIM:=, RAMP_PAR_OFFS:= | :=AVAIL, :=FLOW, :=MELD, :=ERR_INT, :=ERR_SET, :=COIL_ERR, :=TEMP_ERR, :=ZERO_ERR, :=ERR_ALARM, :=FLOW_HIGH);

Inputs

INT	CHANNEL *Node-ID as input-variable* *possible values: 0≙NodeID 32, 2≙NodeID 34,. . . ,30≙NodeID 62*
INT	SETP *Setpoint, possible values: -1000 . . . 1000*
INT	OVERR *Override to reduce Setpoint: 0=stop, 1000=100%* *possible values:0 - 1000*
INT	RAMP1_LIM *setpoint, above which first ramp is used (abs value)* *0 → always first ramp*
INT	RAMP_PAR_OFFS *0: no ramps are sent to PSL_CAN* *1,...,96: the userparameters 1,...,4 bzw 96,...,99 are sent as ramps to PSL_CAN in the following order 1: positiv raising, 2: positiv falling, 3: negativ rising (B-side), 4: negativ falling (B-side)* *100,...,1048: the parameters of the ramps 0,...48 of the PLVC are sent to PSL_CAN;*

Continued on the next page. . .

. . .continued from previous page

	e.g.: *100 → parameters of the ramp from analog input 0* *132 → parameters of the ramp from proportional valve 0* *140 → parameters of the ramp from proportional valve 8* *141 → parameters of the ramp from proportional valve 9* *Note: Double valves do not make use of the odd-numbered parameters of the ramps 32 to 47 which makes those parameters eaven more useful*
Outputs	
BOOL	AVAIL *device responding/alive*
INT	FLOW *spool position, possible values: 0-1000 ≙ 0-100%*
BYTE	MELD *bitfield for errors, including errors below*
BOOL	ERR_INT *internal error*
BOOL	ERR_SET *setpoint out of range*
BOOL	COIL_ERR *coil resistance*

Continued on the next page. . .

. . .continued from previous page

BOOL	TEMP_ERR *temperature too high*
BOOL	ZERO_ERR *setpoint-not-zero error*
BOOL	ERR_ALARM *or-gated errors*
BOOT	FLOW_HIGH *spool too far out*

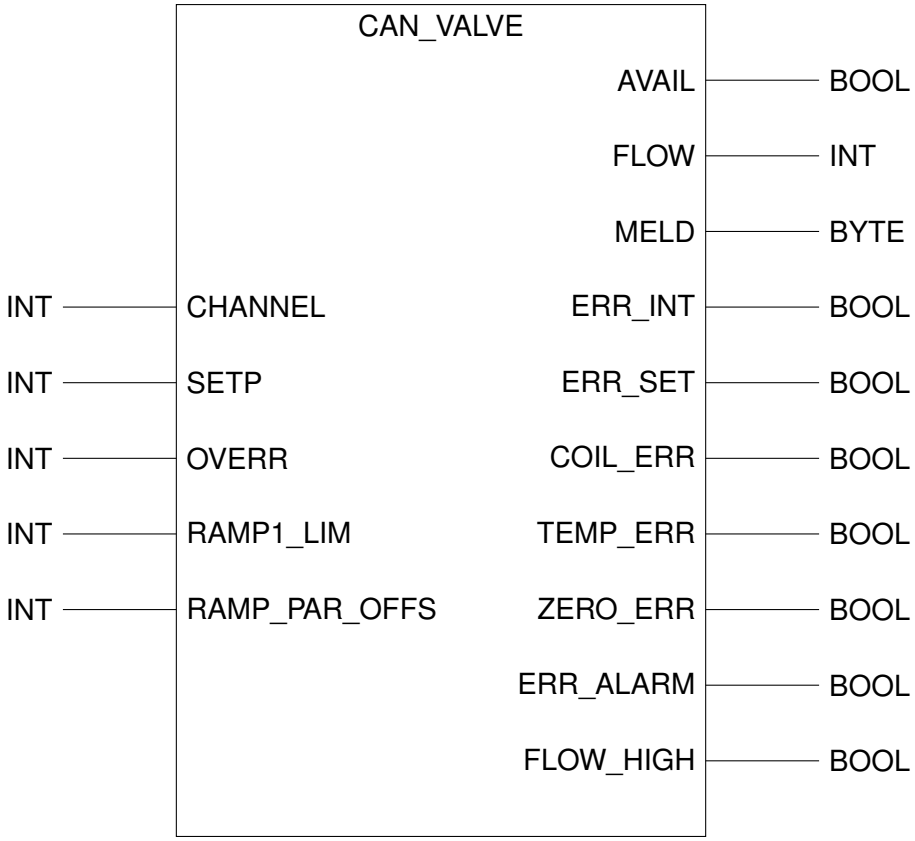


Figure 12.27.: Functionblock CAN_VALVE

Bit 0: ≙ ERR_INT	self-test-errors which avoid regular use. ERRMASK_STARTUP_SFT ERRMASK_EEPROM_CHECKSUM ERRMASK_FLASH_CHECKSUM
Bit 1: ≙ ERR_SET	Timeout error bus commands ERRMASK_SETP_TIMEOUT ERRMASK_GUARD_TIMEOUT ERRMASK_SETPOINT
Bit 2: ≙ Coil_ERR	errors avoiding ragular use, reset + reboot by voltageinterruption necessary ERRMASK_COIL_RES_HIGH ERRMASK_COIL_RES_LOW ERRMASK_CURRENT_CONTROL
Bit 3: ≙ TEMP_ERR	limited automatic mode (e.g. overheating) ERRMASK_T_LIMIT_HIGH
Bit 4: ≙ FLOW_HIGH	Lag error plus ERRMASK_POS_PLUS
Bit 5:	Lag error minus ERRMASK_POS_MINUS
Bit 6: ≙ ZERO_ERR	Setpoint not 0 when switched on ERRMASK_SETP_NEQU_NEUTRAL
Bit 7: ≙ ERR_ALARM	self resetting errorstatus caused by surrounding conditions ERRMASK_VOL_SUPPLY_LOW ERRMASK_VOL_SUPPLY_HIGH
Bit 8:	Electronic temperature too low ERRMASK_TEMP_LOW
Bit 9:	Electronic temperature too high

Continued on next page. . .

. . .continued from previous page

	ERRMASK_TEMP_HIGH

Display

Besides the graphic display that is programmed with an own software, PLVC supports the use of a 2*16 digit text-display. Standard menus for diagnosis are available there, supplied by the OS. Additional user-programmed texts can be shown.

The PLC-programmer can program a set of texts, two lines each (beginning with channel 2). Depending on external events he can activate one of these sets (see PUT_PAR, CHANNEL=4,(12.6.3)). Within this actual texts, he can add the actual value of the variables via DISP_VAL.

The 5 keys of the display are available on the %IB15:

Up-Key	%IB15.0 (partly reserved by OS - use with caution, because the user may get in a OS-menu when key is pressed)
Down-Key	%IB15.1
Right-Key	%IB15.2
Left-Key	%IB15.3
Enter-Key	%IB15.4

Table 12.17.: Keys Available on the Display of the %IB15

The up and down keys may bring you to the OS-menues. But you can disable them with PUT_PAR (see PUT_PAR (12.6.3) for more description).

DISP_TXT

Puts a textstring into one line of the display.

DISP_TXT disp_txt(CHANNEL:=,OFFSET:=,PUT_INFO:=);	
Inputs	
INT	CHANNEL *Line number. Possible values:* *2,3: set0* *4,5: set1* ⋮ *30,31: set14*
INT	OFFSET *Possible values: 0-15*
TEXT-STRING	PUT_INFO *String to be written*

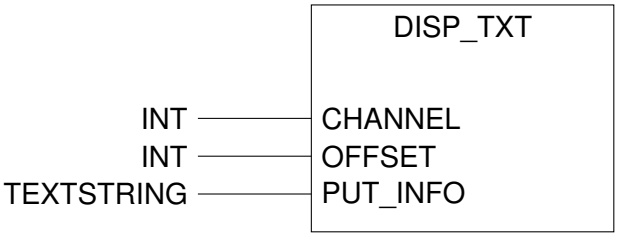


Figure 12.28.: Function DISP_TXT

Inputs:

CHANNEL	Line number 2-31 (2,3 = set 0, 4,5 = set 1 etc.) From channel 100 the texts of the parameters can be overwritten.
OFFSET	Column (0-15)
PUT_INFO	The string to be written.

Table 12.18.: DISP_TXT Inputs

The operator “OFFSET” defines the cell, the first character is written in. If OFFSET=0, the information starts in cell 0, the first cell. The information then can be 16 characters. If OFFSET=15, only the last cell is written and the information can only be one charakter. Put OFFSET=3 and the textstring “HELLOWORLD” is centered.

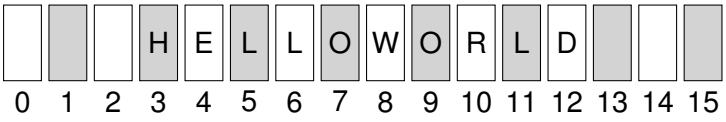


Figure 12.29.: One line of the display

Description:

See above and in chapter: Sample with the Display (DISP_TXT, DISP_VAL) (12.7.2).

DISP_VAL / DISP_VAL2

Puts a value of a variable on the display.

This functionblock can be used together with DISP_TXT. Via OFFSET or space characters, blanks in the DISP_TXT textstring are kept, which are filled by variable contents out of DISP_VAR.

DISP_VAL / DISP_VAL2 disp_val(CHANNEL:=,OFFSET:=,LENG:=,VAL:=);	
Inputs	
INT	CHANNEL *Line number. Possible values:* *2,3: set0* *4,5: set1* ⋮ *30,31: set14*
INT	OFFSET *Possible values: 0-15*
INT	LENG *Number of digits*
INT/DINT	VAL *Variable value*

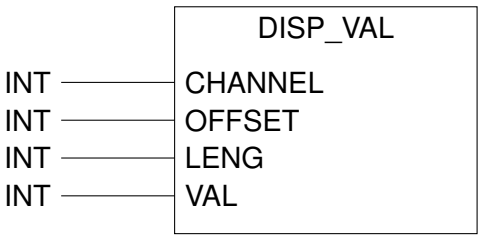


Figure 12.30.: Function DISP_VAL

Inputs:

CHANNEL	Line number 2-31 (2,3 = set 0), (4,5 = set 1) etc. From channel 32 the texts of the parameters can be overwritten.
OFFSET	Column (0-15)
LENG	Number of digits
VAL	Variable Value

Table 12.19.: DISP_VAL Inputs

Description:

See above and in chapter: Sample with the Display (DISP_TXT, DISP_VAL) (12.7.2).

EE_SAVE

If you need a counter which cannot be erased by a reset, HAWE offers the EE-Save-Option which can be ordered with a PLVC41. With this option you can switch the PLVC off via software, after saving. This is the schematic plan of this product:

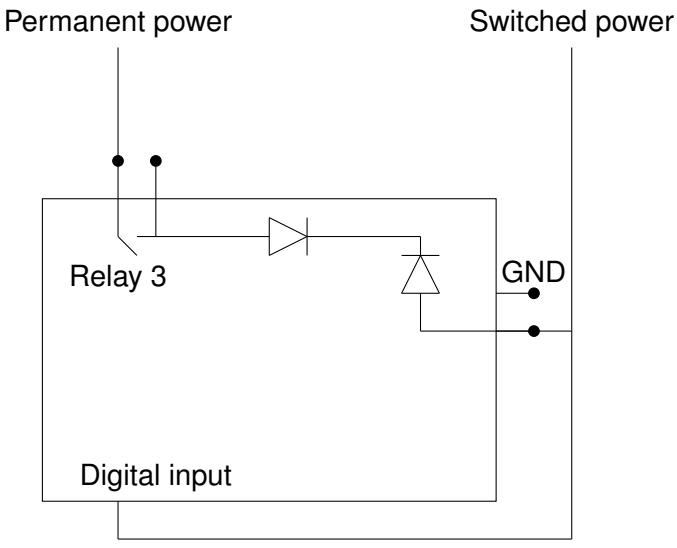


Figure 12.31.: Function EE_SAVE

The programming is easy: You have to check the input. If you detect a falling edge, you have to write all important variables with PUT_PAR. Then you start a timer of 500ms, and when the timer is running out, you switch the relay 3 off. By this, the whole PLVC is powered off with delay.

After restart, the values are read by GET_EE.

F_TRIG - Detect Falling Edge

F_TRIG f_trig(CLK:= | :=Q);	
Inputs	
BOOL	CLK Possible values: TRUE, FALSE
Outputs	
BOOL	Q Possible values: TRUE, FALSE

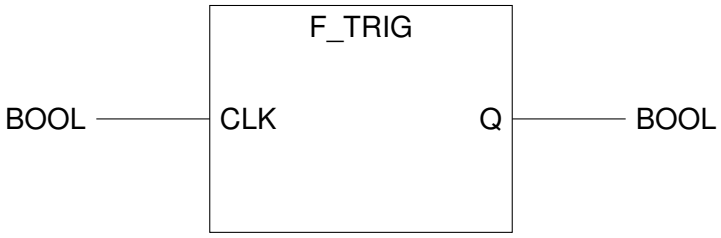


Figure 12.32.: Function TRIG

Inputs:

CLK	Signal to detect falling edge from
Q	Status: TRUE on a falling edge of CLK

Table 12.20.: F_TRIG Inputs

Description:

F_TRIG monitors its input CLK and delivers a TRUE on output Q anytime that a transition from 1 to 0 is detected on input CLK.

GET_COS

Fast cosine function.

GET_COS get_cos(IN:= | :=ZAHL);	
Inputs	
INT	IN *Angle in* $\frac{1}{10}^\circ$
Outputs	
INT	ZAHL *Calculated value in* $\frac{1}{10000}$

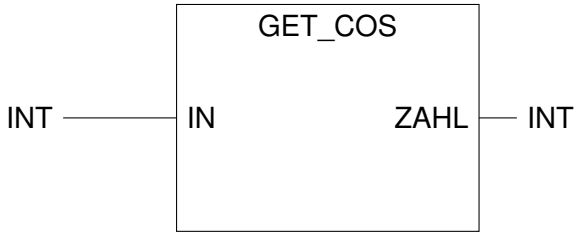


Figure 12.33.: Function GET_COS

Description:

GET_COS calculates the cosine of the value of IN in 10ths per thousand($\frac{1}{10000}$. You have to devide this value by 10000 to get the real value of the cosine.
Watch out to enter the angle in $\frac{1}{10}^\circ$: e.g. 120° equates to IN:=1200

NOTE

The domain of definition of GET_COS.IN is 0,...,1800 ($\widehat{=}\{0^\circ,\ldots,180^\circ\}$)

Example: Calculates cos(180°) and multiplies this value by 5

Variable declaration:

```
VAR
value : INT :=5;
result: INT;
cosinus: GET_COS;
END_VAR
```

Program:

```
cosinus(IN:=1800);  (* cos(180) *)
result:= mul_div(value, cosinus.zahl, 10000);
```

Since $cos(180°) = -1$, the variable “result” has to be -5.

GET_EE_DW

Reads internal variables of a Graf-Syteco-Display with the length of 4 byte.

GET_EE_DW get_ee_dw(CHANNEL:= | :=EE_VAL);	
Inputs	
INT	CHANNEL *possible values: 0-49*
Outputs	
DINT	EE_VAL *read value*

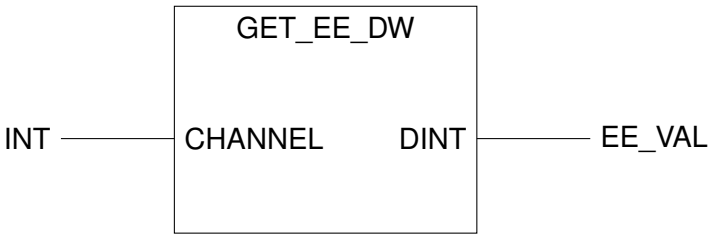


Figure 12.34.: Function GET_EE_DW

GETTIME

Get current system time.

GETTIME gettime(TT:= | :=ET)	
Inputs	
TIME	TT *previous time*
Outputs	
TIME	ET *current system time*

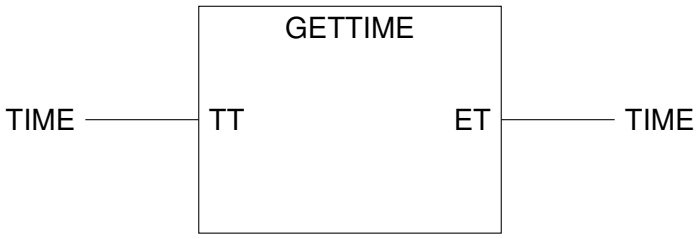


Figure 12.35.: Function GETTIME

Inputs:

TT	Previous time

Table 12.21.: GETTIME Inputs

Description:

GETTIME will retrieve the time elapsed since the controller has last been switched on, less the time value is supplied as an input. This can be used to easily measure time spans.

Example:

Stop watch

```
PROGRAM StopW
VAR
  begin, result : TIME;
  timer : GETTIME;
END_VAR
 (* start *)
  timer (TT := t#0ms);
  begin := timer.ET;
   ...

 (*  stop *)
  timer (TT := begin);
  result := timer.ET;
END_PROGRAM
```

LED_CODE

This function-block is used to control the device-LED of a PLVC 8. It is available since firmware-version published on the 27th of April 2015.

LED_CODE led_code(CODE:=,TL:=,TR1:=,TR2:=,TR3:=);	
Inputs	
INT	CODE *Flashing code. possible values:* *1: continuous flashing of the red LED* *10: continuous flashing of the green LED* *100 to 199: continuous light of red LED*

Continued on the next page. . .

. . .continued from previous page

	200 . . . : continuous light of the green LED *random double-digit numbers: 1st digit number of green LED flashing; 2nd digit number of red LED flashing*
INT	TL *Duration of LED active while flashing in $\frac{1}{10}$sec*
INT	TR1 *Duration of the LED being inactive while flashing in $\frac{1}{10}$sec*
INT	TR2 *Duration of the timeout between switching from red (green) to green (red) signal in $\frac{1}{10}$sec*
INT	TR3 *Duration of the timeout before repeating a flashing code in $\frac{1}{10}$sec*

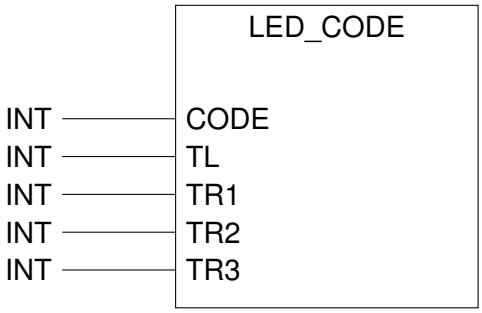


Figure 12.36.: Function ACT_VALVE

Beschreibung:

- The functionblock has to be called cyclically
- All paramters except CODE are optional. If those parameters are not set explicit, following default-values are used: TL:=3, TR1:=3, TR2:=7, TR3:=14

Example: green LED flashing two times and red LED flashing five times

Variable declaration:

```
VAR
  LEDanzeige : LED_CODE
END_VAR
```

Program:

```
LEDanzeige (CODE := 25, TL := 3, TR1 := 3, TR2 := 7, TR3 := 14);
(*  LEDanzeige (CODE := 25);
  is identically because of TL, TR1, TR2 and TR3 were called with
  default-values.*)
```

MUL_DIV

Multiplies two values and divides them by another.

MUL_DIV mul_div(MUL1:=,MUL2:=,DIVI:= | :=);	
Inputs	
INT	MUL1 *first multiplicant*
INT	MUL2 *second multiplicant*
INT	DIVI *divisor*
Outputs	
INT	*calculated value*

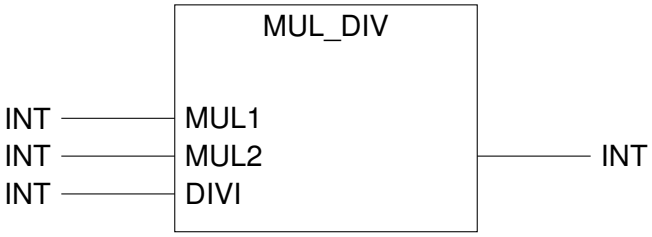


Figure 12.37.: Function MUL_DIV

Inputs:

MUL1	The first multiplicant
MUL2	The second multiplicant
DIVI	The divisor

Table 12.23.: MUL_DIV Inputs

Description:

This function calculates $MUL1 * MUL2/DIVI$. The difference to the “normal” calculating is that this function internally works with 32 bit values.

Example:

You want to multiply a variable var1 with 0.656, so you write:

```
1 Var1 := MUL_DIV(Var1, 656, 1000);
```

With this function and the ACT_VALVE.override you should be able to work without floating point numbers. These are extremly timeconsuming!

Because of MUL_DIV being a function, not a functionblock you can use it without defining anything in the first place.

PUT_CHAR

PUT_CHAR put_char(NR:=, CHAR:=);	
Inputs	
USINT	NR
USINT	CHAR

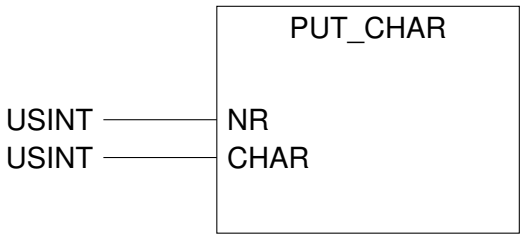


Figure 12.38.: Function PUT_CHAR

PUT_PAR

Writes special parameters and allows to write any analog values to an not used analog input as raw value. In this way You can use the parameters of this analog input included filter and ramps.

Not used analog inputs are:

- AnalogInput 24 to 31 - if no radio remote control ist used - (see figure 12.39)
- AnalogInput 32 to 39 - for this parameter/submenu7 parameter (c) must be set to 3 - (see figure 12.40)

Because internal a conversion is used (necessary for real electrical signals) the shown value will be 2.4% lower than the writen number. For example: put_par para 1024 => analog raw value 1000.

Note: Value must be positive, normally from 0..500 (respectively ..512), or from 0..1000 (resp. 1024), or 0..10000 (resp. ..10240) if you nee high resolution. Negative values will be interpreted as not-connected!

We recommend to use MUL_DIV to convert the value before using PUT_PAR like this:

```
1 value_for_putpar := mul_div(original_value,1024,1000);
```

Example: write a value to analog input 32

Variable declaration:

```
VAR
  write_parameter: PUT_PAR;
  value_x: INT;
END_VAR
```

Program:

```
  value_x := 1000;
  write_parameter.PARA := mul_div(value_x,1024,1000); (* increase by 2,4% *)
  write_parameter(CHANNEL := 32);
```

```
Special Parameters:
(a) Modem type   -1
(*) Radio Type   -1
(c) Ana Nodes    03
(d) CAN-BAUD     03
(e) DEBUG        -1
(f) Closed Loop  -
(g) A. SaturationX
(h) CAN-Nodes    X
(i) CAN-Error    -
```

Figure 12.39.: radio control disabled

```
Special Parameters:
(a) Modem type   -1
(b) Radio Type   -1
(*) Ana Nodes    03
(d) CAN-BAUD     03
(e) DEBUG        -1
(f) Closed Loop  -
(g) A. SaturationX
(h) CAN-Nodes    X
(i) CAN-Error    -
```

Figure 12.40.: CAN HMI aktivated

Structure of function block:

PUT_PAR put_par(CHANNEL:=,PARA:=);	
Inputs	
INT	CHANNEL *selected parameter, see table 12.24*
INT	PARA *value to be written*

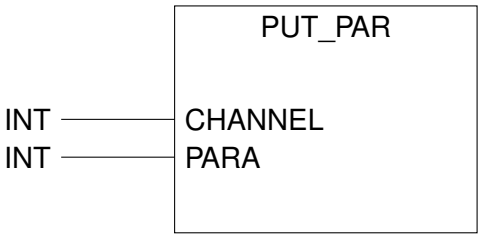


Figure 12.41.: Function PUT_PAR

The meaning depends on CHANNEL:

CHANNEL	Selected parameter
0	Set message no. PARA in graphic display
1	Reset message no. PARA in graphic display
2	Set screen no. PARA in graphic display
3	Reset screen no. PARA in graphic display
4	Small display: Setting lines, selects which lines are shown
5	Number of changable parameters in small display
6	Number of enabled menus in small display
7	Choose shown OS menu
8	Change to desired column of the OS menu
9	Change to display menu column 2
10	Send simulated navigation key of small display: 1=up, 2=down, 4=left, 8=right, 16=enter
99	Bit word: 1 = current controller only PWM
100	Ramp value for internal flow regulator
200 + number of AnalogInput	writes an value as raw value to this analog input
1000-1049	External variable for display on graphic display

Table 12.24.: PUT_PAR Values for CHANNEL

4: The lines are combined to sets. Line 2+3 are set 0, line 4+5 set 1 and so on. With this function you can choose the “set”. See description of function block DISP_TXT in chapter 12.6.3.

5: This limits the changeable user parameters in the small display.

6: This limits the enabled menus in the CAN BC. Which menus are on which position you can see the CAN BC description. If you write a -1 there, no menus are visible; at 0 you get to the lowest menu.

PUT_PAR2

Writes spezial parameter as DINT.

PUT_PAR2 put_par2(CHANNEL:=,PARA:=);	
Inputs	
INT	CHANNEL *selected parameter, see table 12.26*
DINT	PARA *value to be written*

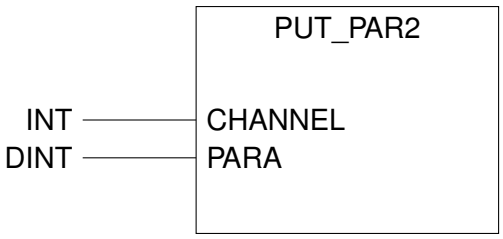


Figure 12.42.: Function PUT_PAR2

Inputs:

CHANNEL	Selected parameter
PARA	The value to be written

Table 12.25.: PUT_PAR2 Inputs

Description:

Writes specific parameters. The meaning depends on CHANNEL:

CHANNEL	Selected parameter
100-149	Internal variable for display on graphic display
1000-1049	External variable for display on graphic display

Table 12.26.: PUT_PAR2 Values for CHANNEL

RAMPS

The function block RAMPS can be used if you need a ramped value. As both inputs and outputs of PLVC already have a ramp, it will not be needed frequently.
If however there is a digital Input to generate a ramped setpoint, this function block can be used.

RAMPS ramps(STEP_UP:=,STEP_DN:=,TARGET:= | :=OUT);	
Inputs	
INT	STEP_UP *max. steps per second for rising ramps*
INT	STEP_DN *max. steps per second for falling ramps*
INT	TARGET *where to ramp*
Outputs	
INT	OUT *returns the ramped value*

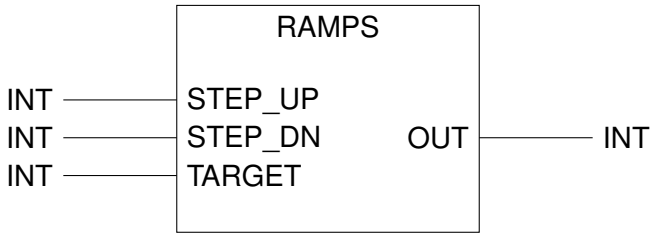


Figure 12.43.: Function RAMPS

Parameter

- *STEP_UP:* gives the max. steps of the FB per second for rising ramps.
- *STEP_DN:* gives the max steps of the FB per second for falling ramps.
- *TARGET:* tells where to ramp, usually the non-ramped setpoint.
- *OUT:* returns the ramped value.

Description

The duration for the ramp is calculated as followed

$$\frac{1000}{STEP\_UP\frac{1}{s}}$$

for rising values and

$$\frac{1000}{STEP\_DN\frac{1}{s}}$$

for falling values.

STEP_UP:=1000	1 second duration for rising ramp
STEP_UP:=100	10 seconds for rising ramp
STEP_UP:=10000	0,1 seconds for rising ramp
STEP_DN:=1000	1 second for falling ramp
STEP_DN:=100	10 seconds for falling ramp
STEP_DN:=10000	0,1 seconds for falling ramp

Example: Configure proportional output 2 and 3 as a twin coil

Variable declaration:

```
VAR
  ramp1:RAMPS;
  setpoint: INT;
  dig_input AT %IB3.4: BOOL;
END_VAR
```

Program:

```
if dig_input then setpoint:= 1000;
else setpoint := 0;
end_if;

ramp1(STEP_UP:=1000, STEP_DN:=2000, TARGET:=setpoint); (*ramp 1s rising,
    500ms falling*)
setpoint := ramp1.OUT;
```

REG_PI

REG_PI

reg_pi(IST_WERT:=, SOLL:=, MN:=, MX:=, PF:=, TN:=, | :=OUT);

Inputs

INT	IST_WERT

Continued on the next page. . .

. . .continued from previous page

INT	SOLL
INT	MN
INT	MX
INT	PF
INT	TN
DINT	I
INT	E
DINT	Q0
DINT	Q1

Continued on the next page. . .

. . .continued from previous page

INT	PFA
INT	TNA
INT	RES1
INT	RES2
INT	RES3
INT	RES4
Outputs	
INT	OUT

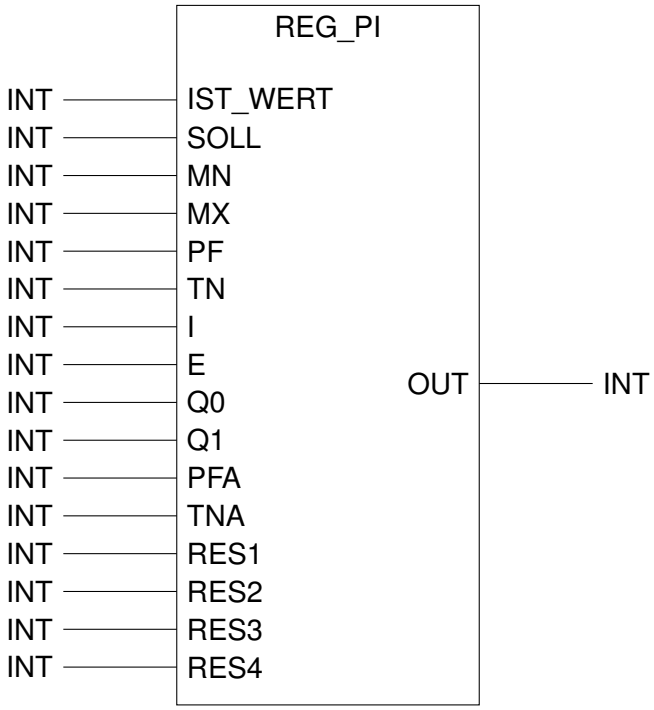


Figure 12.44.: Function REG_PI

SPLINE

Linear interpolation between several pairs of values with limit of range.

SPLINE spline(IN:=,X0:=,Y0:=,X1:=,X2:=,...,X15:=,Y15:= | :=OUT);	
Inputs	
INT	IN *possible values: between X0 and X15*
INT	X0 *X-value of first pair*
INT	Y0 *Y-value of first pair*

Continued on the next page. . .

. . .continued from previous page

⋮	
INT	X15 *X-value of 15th pair*
INT	Y15 *Y-value of 15th pair*
Outputs	
INT	OUT *computed value*

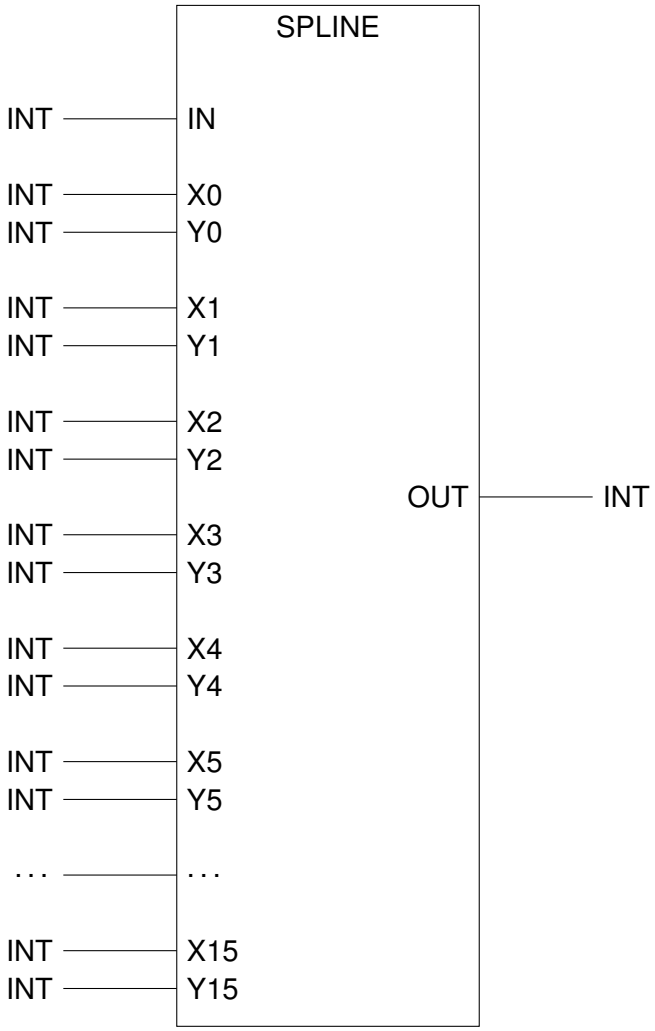


Figure 12.45.: Function SPLINE

Inputs:

X	Input
X1	First pair
Y1	First pair
X2..X15	Next pair
Y2..Y15	Next pair
MAX_Y	Maximum
MIN_Y	Minimum

Table 12.29.: SPLINE Inputs

Description:

A linear interpoltation between 15 pairs of points is computed and given back in value OUT.

For an accurately reproduktion of a more complex curve, SPLINE can be used. The X values of the points have to be in ascending order, but not all 16 must be used. The value range of the run-time parameter IN must be between X0 and X15.

SPLINE2

Interpolation in a two-dimensional stretched grid (of curves).

SPLINE2 spline2(IN:=,IN2:=,X0:=,...,X9:=0,X2_0:=,...X2_9:=,Y0_0:=...Y9_9:= | :=OUT);	
Inputs	
INT	IN *possible values: between X0 and X15*
INT	IN2 *possible values: between X2_0 and X2_15*
INT	X0 *1st X-value*
⋮	
INT	X9 *9th X-value*
INT	X2_0 *1st X2-value*

Continued on the next page. . .

. . .continued from previous page

⋮	
INT	X2_9 *9th X2-value*
INT	Y0_0 *Y-value referring to X0,X2_0 combination*
⋮	
INT	Y7_3 *Y-value referring to X7,X2_3 combination*
⋮	
INT	Y9_9 *Y-value referring to X9,X2_9 combination*
Outputs	
INT	OUT *computed value*

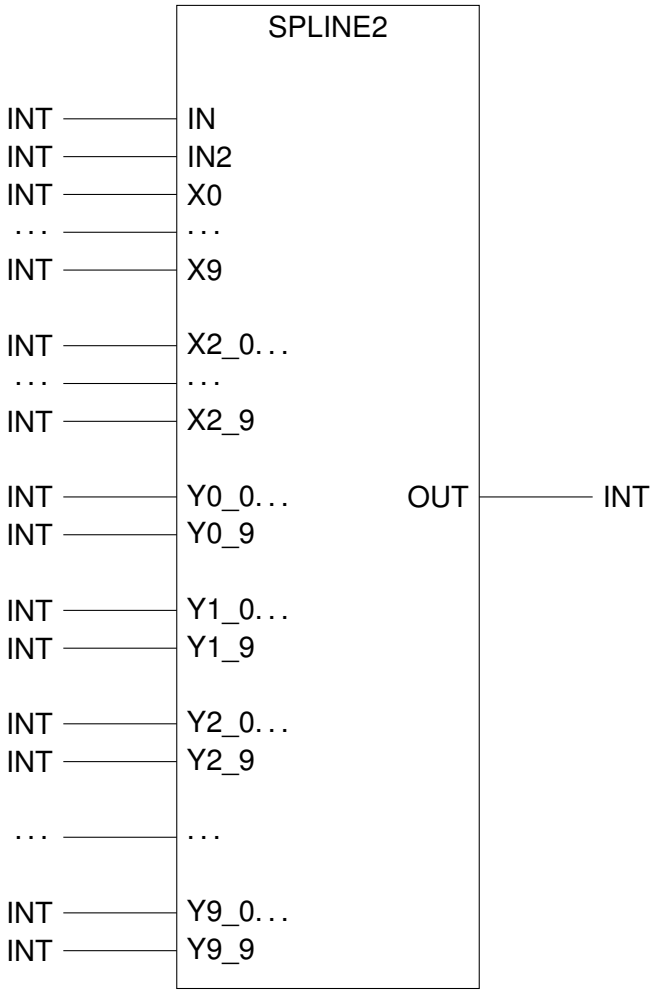


Figure 12.46.: Function SPLINE2

Description:

If a complex surface should be modeled as a function of two variables, SPLINE2 can be used. The X- and X2- values of the points must be in ascending order, but all 9 don't have to be used. The range of the term IN parameter must be between X0 and X15. The range of the duration parameter IN2 must be between X2_0 and X2_15.

Example: max. load moment of a crane, depending on the angle and the length of the main arm: I.e. for 10 different angles from -10 to 90 degrees and the 10 different lengths of the telescope, the values are entered. At runtime, the approximate values for all the intermediate points are calculated by interpolation.

TOF - Timed Off Delay

TOF tof(PT:=,IN:= | :=ET,:=Q);	
Inputs	
TIME	PT *time to delay the switching off*
BOOL	IN *incoming signal getting delayed*
Outputs	
TIME	ET *time passed since IN:=FALSE (reset, as soon as Q:=0)*
BOOL	Q *off-delayed output signal*

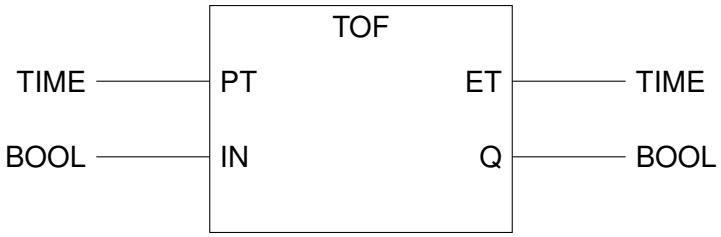


Figure 12.47.: Function TOF

Inputs:

IN	Start condition
PT	Preset time
Q	Status
ET	Elapsed time

Table 12.31.: TOF Inputs

Description:

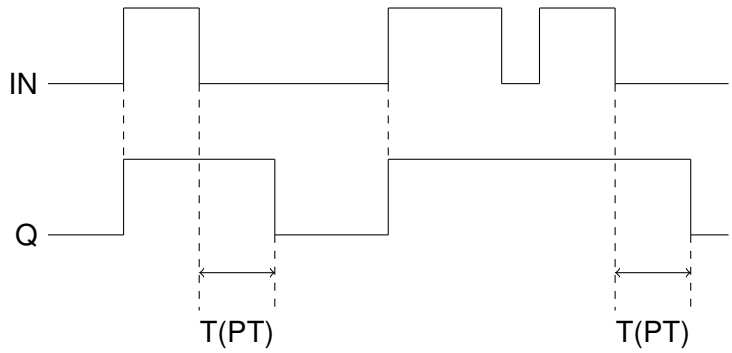


Figure 12.48.: Time Diagram TOF

If input IN is TRUE, output Q will immediately be set to TRUE as well. With the falling edge of input IN, the timer will start and output Q will be kept to TRUE for the time given by input PT. Input PT will be sampled only on the rising edge of input IN, later changes have no effect. Output ET gives the time elapsed since the falling edge of input IN, but it will not go negative.

Example:

Delay for 125 milliseconds

```
VAR
  Timer3 : TOF;
  Start AT %I0.0 : BOOL;
  Duration : TIME := T#125ms;
  Output AT %Q0.0 : BOOL;
  ActTime : TIME;
END_VAR
  Timer3(IN := Start, PT := Duration);
  Output := Timer3.Q;
  ActTime := Timer3.ET;
```

TON - Timed On Delay

TON ton(PT:=,IN:= | :=ET,:=Q);	
Inputs	
TIME	PT *time to delay the switching on*
BOOL	IN *incoming signal getting delayed*
Outputs	
TIME	ET *time passed since IN:=TRUE (reset, as soon as Q:=TRUE)*
BOOL	Q *on-delayed output signal*

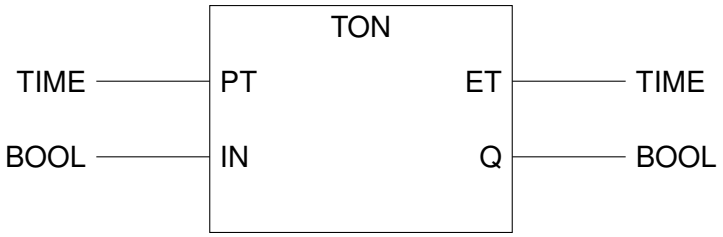


Figure 12.49.: Function TON

Inputs:

IN	Start condition
PT	Preset time
Q	Status
ET	Elapsed time

Table 12.32.: F_TRIG Inputs

Description:

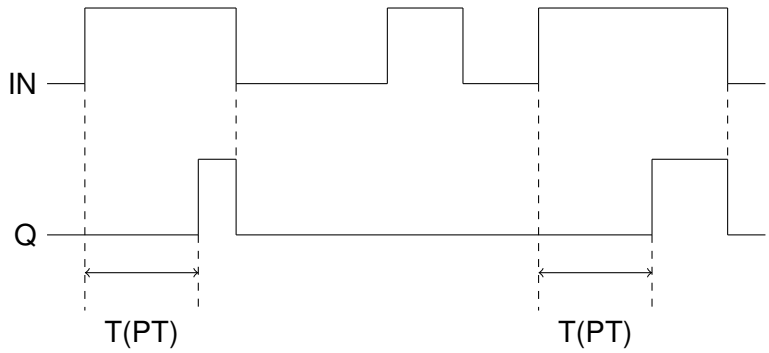


Figure 12.50.: Time Diagram TON

The rising edge of input IN will start the timer. Output Q will go to TRUE after the time given by PT has elapsed after the rising edge of input IN. If IN is FALSE, Q will always be set to FALSE. PT is only sampled on the rising edge of input IN, later changes will have no effect. Output ET delivers the time elapsed since the rising edge of IN, but will not go higher than PT. If IN is FALSE, ET will be 0.

Example:

Off-delay by 12 milliseconds

```
VAR
  Timer2 : TON;
  Start AT %IB0.0 : BOOL;
  Duration : TIME := T#12ms;
  Output AT %QB0.0 : BOOL;
  ActTime : TIME;
END_VAR
  Timer2 ( IN := Start, PT := Duration);
  Output := Timer2.Q;
  ActTime := Timer2.ET;
```

TP - Timed Pulse

TP tp(PT:=,IN:= | :=ET,:=Q);	
Inputs	
TIME	PT *duration of the pulse*
BOOL	IN *signal, causing the pulse*
Outputs	
TIME	ET *time passed since IN:=TRUE (reset, as soon as Q:=0)*
BOOL	Q *pulsed output signal*

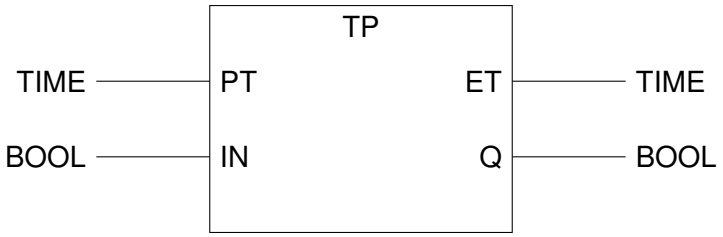


Figure 12.51.: Function TP

Inputs:

IN	Start condition
PT	Preset time
Q	Status
ET	Elapsed time

Table 12.33.: F_TRIG Inputs

Description:

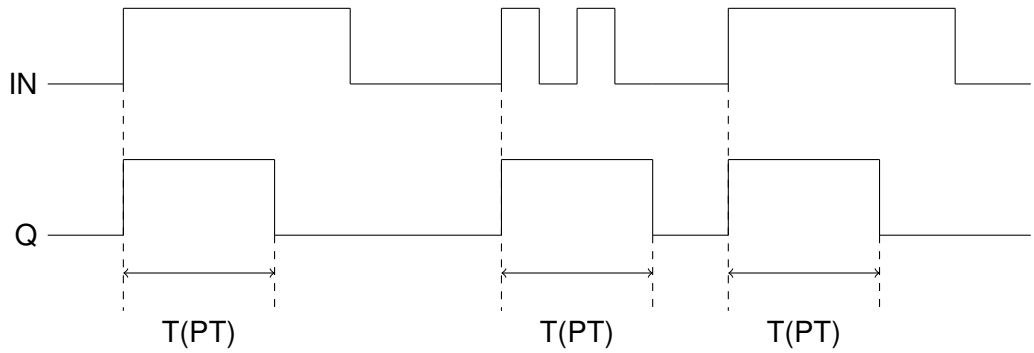


Figure 12.52.: Time Diagram TP

The rising edge of input IN will start the timer, but only if it is idle. Output Q will be set to TRUE for the time given by input PT. A falling edge on IN will not stop the timer nor change the output Q while the timer is running, neither will another rising edge during this time restart the timer. Input PT samples only on the rising edge of IN starting the timer, later changes will have no effect.

Example:

125ms pulse

```
VAR
  Timer1 : TP;
  Start AT %IB0.0 : BOOL;
  PulseDuration: TIME := T#125ms;
  OutputPulse: BOOL;
  ActTime : TIME;
END_VAR
  Timer1(IN := Start, PT := PulseDuration);
  OutputPulse := Timer1.Q;
  ActTime := Timer1.ET
```

12.6.4 Reading Signals

GET_ANA

Reading Analog Inputs

Values from all analog inputs may direktly be assigned at the deklaration of variables.
For example:
Analog Input 40

```
VAR
  preasure AT\% IW104.0:INT;
END_VAR
```

Get the adress (IW104.0) from the Pinning.

Nevertheless you have the possibility to get these values of analog inputs and also the current of outputs, the voltage of battery and the values of CAN PDOs with function block **GET_ANA**.

GET_ANA get_ana(CHANNEL:= | :=ANA_VAL,:=OK);	
Inputs	
INT	CHANNEL Channel you want to read: *Possible values* 0-47*: Calculated Value of Input 0 to 47* 48-63*: current of the outputs 0-15 in mA* 1000*: battery voltage* **With firmware since jear 2013 and later:** 64-223*: value from CAN Id 281$_{hex}$ to 2A8$_{hex}$ (PDO2)* *per channel 2 Bytes as integer* 10224-10383*: CAN Id 2A9$_{hex}$ to 2D0$_{hex}$ (PDO2)* 10384-10543*: CAN Id 2D1$_{hex}$ to 2F8$_{hex}$ (PDO2)* 10544-10703*: CAN Id 181$_{hex}$ to 1A8$_{hex}$ (PDO1)* 10704-10863*: CAN Id 1A9$_{hex}$ to 1D0$_{hex}$ (PDO1)* 10864-11023*: CAN Id 1D1$_{hex}$ to 1F8$_{hex}$ (PDO1)*
Outputs	
INT	ANA_VAL *read value*
INT	OK *possible values: 0 →not OK, 1→OK*

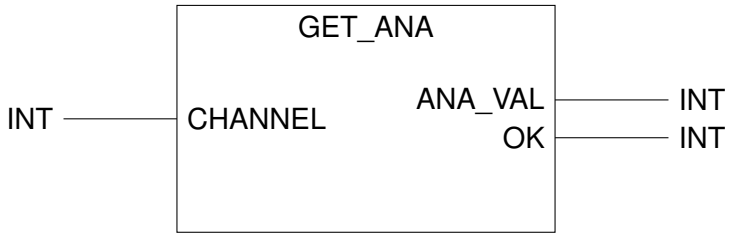


Figure 12.53.: Function GET_ANA

Description of the parameters:

- *CHANNEL* Channel you want to read
- *ANA_VAL* Calculated value of CHANNEL

Note:

Feedback of channel 48 to 63 is a signed current in mA.
The programmer is responsible to check cable break. Return value of ANA_VAL is zero if anything is wrong.
The values for ANA_VAL result from the actual analog value and the values set in ANA_INI (or directly via terminal programm, visual tool).

Example: Read current from prop.-channel 0

Variable declaration:

```
VAR
  read_ana: GET_ANA;
  current_prop0: INT;
END_VAR
```

Program:

```
read_ana(CHANNEL :=48);
current_prop0 := read_ana.ana_val;
```

Example: Read analog input 5

Variable declaration:

```
VAR
  read_ana: GET_ANA;
  ana5: INT;
END_VAR
```

Program:

```
read_ana(CHANNEL :=5);
ana5 := read_ana.ana_val;
```

If the contents of CAN PDOs (AnalogInputs Submenu 7 to C) should be evaluated byte by byte, this can be done in the OpenPCS like this:

Example: Byte 0 und 1 einzeln ohne Vorzeichen

Variable declaration:

```
VAR
  read_ana: GET_ANA;
  value0: USINT; (* unsigned small integer *)
  value1: USINT;
  byte0:  BYTE;  (* byte *)
  byte1:  BYTE;
END_VAR
```

Program:

```
getanalog(channel := 64); (* values from byte 0 and 1 from CAN ID 281h *)
value0 := INT_TO_USINT(getanalog.ANA_VAL); (* value from byte 0 *)
value1 := UINT_TO_USINT(INT_TO_UINT(getanalog.ANA_VAL)/256); (* value from
    byte 1 *)
  (*
     alternatively to evaluate bit by bit later - values as byte
                                                                  *)
byte0 := INT_TO_BYTE(getanalog.ANA_VAL);
byte1 := UINT_TO_BYTE(INT_TO_UINT(getanalog.ANA_VAL)/256);
```

Example: Value from byte 0 and 1 as single signed values

Variable declaration:

```
VAR
  read_ana: GET_ANA;
  value0: SINT; (* signed small integer *)
  value1: SINT;
END_VAR
```

Program:

```
read_ana(CHANNEL := 64);
value0 := INT_TO_SINT(read_ana.ana_val);
value1 := UINT_TO_SINT(INT_TO_UINT(read_ana.ana_val)/256);
```

Example: to combine 2 integer values to one INT32

Variable declaration:

```
VAR
  read_ana: GET_ANA;
  ana1:    INT;
  ana2:    INT;
  valueUnsigned:  UDINT;
  value32:    DINT; (* signed double integer *)
END_VAR
```

Program:

```
read_ana(CHANNEL := 64 | ana1 := ana_val);
read_ana(CHANNEL := 65 | ana2 := ana_val);

valueUnsigned := UINT_TO_UDINT(INT_TO_UINT(ana1));
value32 := UDINT_TO_DINT(valueUnsigned);
valueUnsigned := 16#10000 * UINT_TO_UDINT((INT_TO_UINT(ana2))
value32 := value32 + UDINT_TO_DINT(valueUnsigned);
```

FQ_READ

Reads frequency inputs.

FQ_READ fq_read(CHANNEL:= | :=Upm,:=Hz);	
Inputs	
INT	CHANNEL *Select the channel you want to read. Possible values: 0-3*
Outputs	
INT	HZ *Integer value in $Hz/(2^{EXPONENT})$*
INT	UPM *Integer value in RPM*

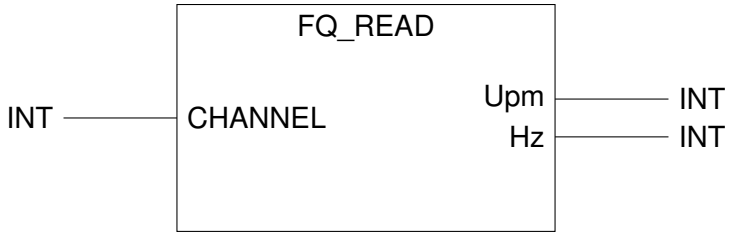


Figure 12.54.: Function FQ_READ

Description:

- After initialisation you can read the channel with this function. There are two values: Hz and UPM.
- $Hz = [Hz] \cdot 2^{EXPONENT} \Rightarrow [Hz] = Hz/2^{EXPONENT}$ (EXPONENT is given with the FQ_INI).
- UPM is the rounds per minute. This value can be computed with the formula

$$\frac{\frac{PULSE}{\Delta t}}{PPU} = \frac{PULSE}{\Delta t \cdot PPU},$$

with Δt being the range of time in which the pulses where counted.

Example: Read frequency channel 0

Variable declaration:

```
VAR
  freq_ini: FQ_INI;
  freq_read : FQ_READ;
  upm_0 : INT;
  hz_0: INT;
END_VAR
```

Program:

```
freq_ini(CHANNEL :=0,
    EXPONENT:= 0,
    PPU:= 1000);
freq_read(CHANNEL :=0);
upm_0 := freq_read.upm;
hz_0 := freq_read.Hz;
```

NOTE

Every frequency channel has to be initialized with FQ_INI.

POS_READ

Read Frequencies

POS_READ pos_read(CHANNEL:=,ZERO_SET:= | :=POS1,:=POS2);	
Inputs	
INT	CHANNEL *choosen channel, possible values: 0-2*
BOOL	ZERO_SET *zero positioning, possible values: TRUE, FALSE*
Outputs	
DINT	POS1 *outputs an value charakterized by FAK1 out of functionblock POS_INI*
DINT	POS2 *outputs an value charakterized by FAK2 out of functionblock POS_INI*

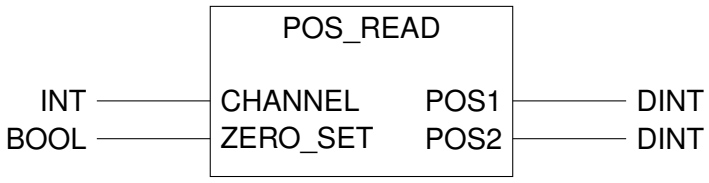


Figure 12.55.: Function POS_READ

Description of the parameters:

- *CHANNEL* Choosen frequency channel
- *ZERO_SET* Switch zero position setting on/off

Note:

- Every counting channel has to be initialized with POS_INI.
- Channel 0-2 equates frequency channels FIN1-FIN3.

- The counter will be set to zero if ZERO_SET is not set to 0.
- If you want to use POS1,2 as INTEGER, you have to limit the range with FAK1,2 and use function DINT_TO_INT.

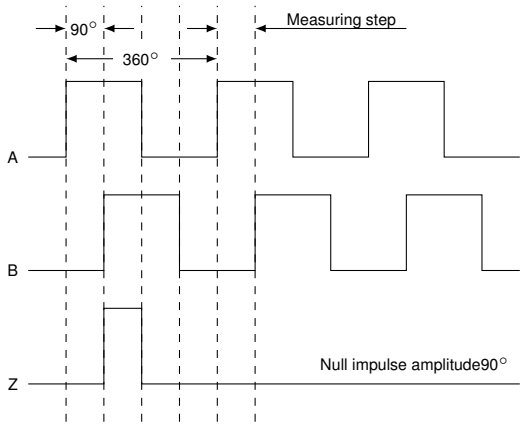


Figure 12.56.: Configure an Incremental Encoder

Continuation of example POS_INI(12.6.2).

Example: Use frequency input 0 as position measurement

Measures the number of degrees the disc with 10000 pulses covers in $\frac{1}{10}^{\circ}$. This value is displayed within wert_pos1. The number of pulses is given to wert_pos2.

Variable declaration:

```
VAR
  posi: POS_INI;
  get_pos: POS_READ;
  wert_pos1: DINT;
  wert_pos2: DINT;
  fqin at %IB3.0 : BOOL;
END_VAR
```

Program:

```
posi( CHANNEL :=0,
    FAK1 :=3600,
    FAK2 :=10000,
    OFFS :=0,
    MODE :=1);

get_pos(CHANNEL :=0,
    ZERO_SET :=0);
wert_pos1 := get_pos.pos1;
wert_pos2 := get_pos.pos2;
```

12.6.5 Control Systems

MENGE_INI

Flow Control

With flow control you can easily limit the power output. Furthermore you can avoid undersupply of single functions. The operating system then controls the whole oil distribution!

MENGE_INI menge_ini(CHANNEL:=,LITER:=,GRUPPE:=,LITER_MAX:=);	
Inputs	
INT	CHANNEL *proportional channnel* *possible values: 0-15 (Prop), 16-31 (Quasiprop)*
INT	LITER *0-1000*
INT	GRUPPE *possible values: 0-3 (0=highest priority)*
INT	LITER_MAX *possible values 0-30000*

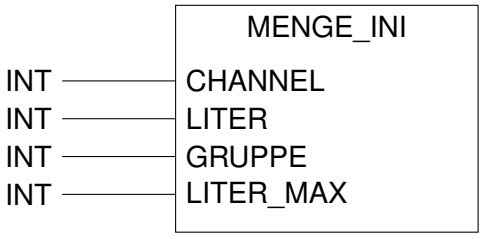


Figure 12.57.: Function MENGE_INI

Description of the parameters:

- *CHANNEL* Proportional channel
- *LITER* Flow rate of the valve (e.g. 40 liters)
- *GRUPPE* Group of the valve
- *LITER_MAX* Max. flow rate of all valves

Note:

For each proportional output (CHANNEL) it is specified how many liters of oil flow at maximum deflection of the valve, and which group the output belongs to.

Assign the flow rate and group of every single valve.

Is the sum of the needed liters greater than LITER_MAX, all consumers are reduced linear to reach the max. flow rate.

Consumers of a low priority group will then get no current at all.

LITER_MAX is the sum of the flow rate of all valves.

Liter can also mean milli- or deciliter. The sum of all liters must not be greater than 32000.

Example: Flow control for three proportional outputs, max. 80 liters

Variable declaration:

```
VAR
  oil_volume: menge_ini;
END_VAR
```

Program:

```
oil_volume( CHANNEL :=0,
    LITER   :=40,
    GRUPPE  :=0,
    LITER_MAX :=80
);

oil_volume( CHANNEL :=2,
    LITER   :=20,
    GRUPPE  :=0,
    LITER_MAX :=80
);

oil_volume( CHANNEL :=8,
    LITER   :=60,
    GRUPPE  :=0,
    LITER_MAX :=80
);
```

BREMS_POS

Controlled Slowdown

In combination with a distance measurement you can manipulate the setpoint of a hydraulic function (e.g. slowdown at end position).

BREMS_POS

brems_pos(IST_WERT:=, POSITION:=, POS_MAX:=, POS_MXB:=, POS_MNB:=, POS_MIN:=, POS_HYST:= | :=BR_SOLL, :=RICHTG, :=WARN);

Inputs	
INT	IST_WERT *present value*
INT	POSITION *present position*
INT	POS_MAX *max. position*
INT	POS_MXB *start slow down at max.*
INT	POS_MNB *start slow down at min.*
INT	POS_MIN *min. position*
INT	POS_HYST *hysteresis at stop position*
Outputs	
INT	BR_SOLL

Continued on the next page. . .

. . .continued from previous page

	calculated value
USINT	RICHTG *possible values: 1=forward, 2=backward*
USINT	WARN *possible values: 0=OK, 1=reduced, 2=stop*

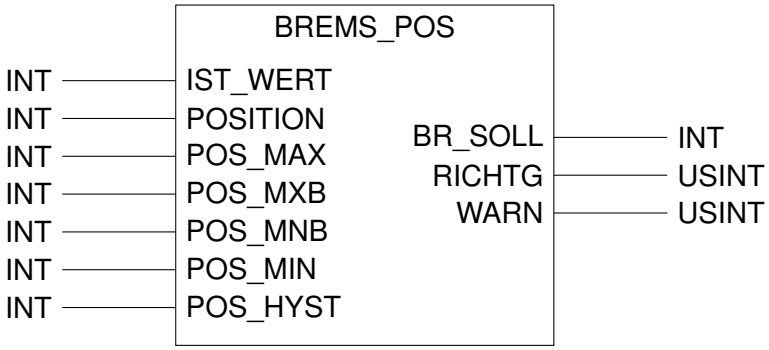


Figure 12.58.: Function BREMS_POS

Note:

Keep in mind: POS_MAX > POS_MXB > POS_MNB > POS_MIN

These are the feedback values for BR_SOLL:

- POSITION > POS_MAX and IST_WERT > 0 : 0
- POSITION < POS_MIN and IST_WERT < 0 : 0
- POSITION < POS_MAX and POSITION > POS_MXB und IST_WERT > 0: 0 to IST_WERT
- POSITION > POS_MIN and POSITION < POS_MNB und IST_WERT < 0 : 0 to IST_WERT
- Else: IST_WERT

AUTO_MOVE

If you have a sensor (e.g. encoders) for a hydraulically driven function, you can manipulate a maximum speed dependend on defined parts as well as a target position.

If a position sensor is available, you can determine the appropriate speed on the basis of your own position, the position of the target and some parameters with AUTO_MOVE. Note that the machine is moving in positive direction (POS_ACT increases) if the result is positive.

The two function blocks FAHR_POS and AUTO_MOVE have the same functionality. The name AUTO_MOVE should be used.

In conjunction with a position sensor you can calculate a setpoint based on the present position, the position of the destination and a few parameters.

AUTO_MOVE auto_move(VEL_LFT:= , VEL_RGT:= , POS_ACT:= , DESTIN:= , VEL_MIN_LFT:= , SLW_LFT:= , HYST_LFT:= , HYST_RGT:= , SLW_RGT:= , VEL_MIN_RGT:= , COR_LFT:= , COR_RGT:= | :=WARN, :=DIR, :=SETP);	
Inputs	
INT	VEL_LFT *max speed left of target, possible values: 0-1000*
INT	VEL_RGT *max speed right of target, possible values: 0-1000*
INT	POS_ACT *actual position*
INT	DESTIN *target position*
INT	VEL_MIN_LFT *minimum speed left*

Continued on the next page. . .

. . .continued from previous page

INT	SLW_LFT *distance left of target (POS_ACT < DESTIN), where setpoint reduction starts*
INT	HYST_LFT *hysteresis left*
INT	HYST_RGT *hysteresis right*
INT	SLW_RGT *distance right of target (POS_ACT > DESTIN), where setpoint reduction starts*
INT	VEL_MIN_RGT *minimum speed right*
INT	COR_LFT *correction left*
INT	COR_RGT *correction right*
Outputs	
INT	WARN *returnvalue for proportional output*
INT	DIR

Continued on the next page. . .

. . .continued from previous page

	direction, possible values: 1=forwards, 2=backwards
INT	SETP *indicates state of slowing down (POS_ACT is between SLW_LFT and SLW_RGT)*

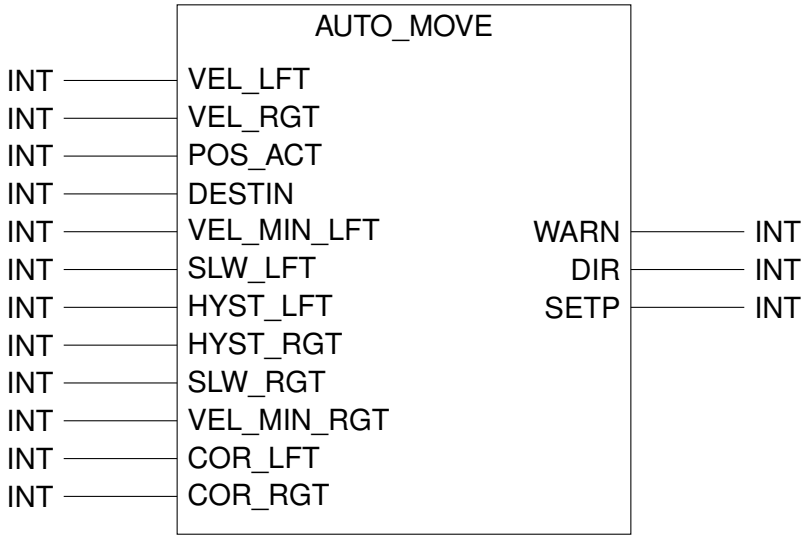


Figure 12.59.: Function AUTO_MOVE

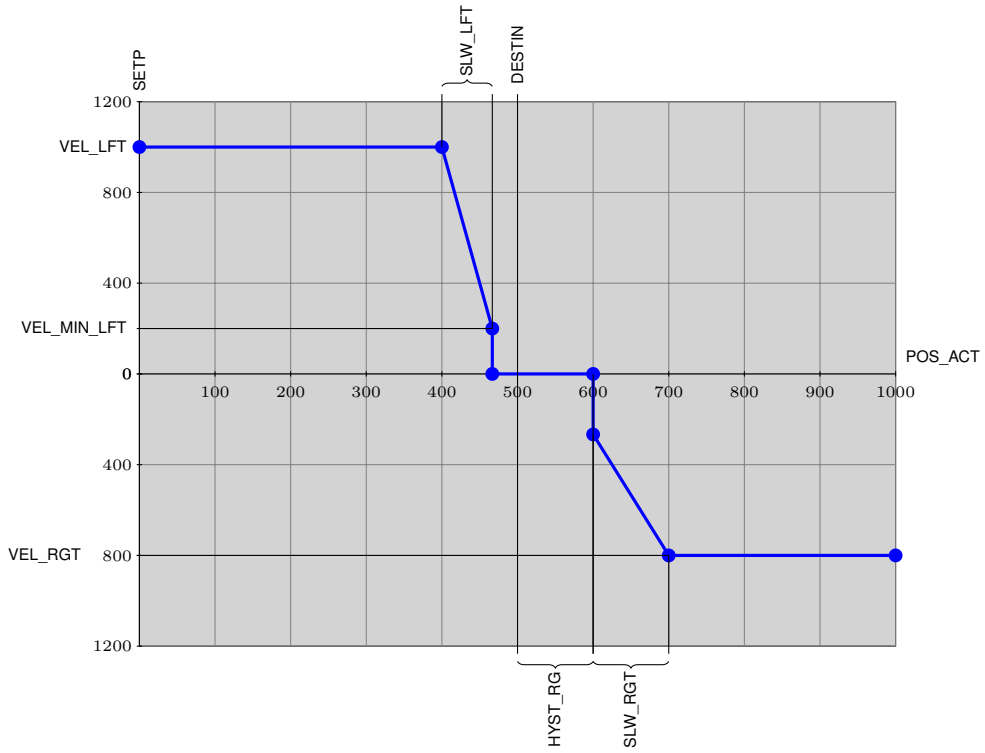


Figure 12.60.: AUTO_MOVE

Example: Move to angle 0

Variable declaration:

```
VAR
  auto_plat: AUTO_MOVE;
  ini: BOOL;
  plat_lsen AT %IW40.0: INT;
  auto_level_on AT %IB3.1:BOOL; (* choose auto levelling *)
  level_joy AT %IW41.0: INT;
  prop: ACT_VALVE;
  setp_level : INT := 500;
END_VAR
```

Program:

```
if ini = false then
  ini:= true;
  auto_plat (
    VEL_LFT := 1000,  VEL_RGT :=1000,
    VEL_MIN_LFT :=50,      VEL_MIN_RGT :=50,
    SLW_LFT :=40,       SLW_RGT :=40,
    HYST_LFT :=5,      HYST_RGT :=5,
    COR_LFT :=0,          COR_RGT :=0
  );
end_if;
if auto_level_on then
  auto_plat( POS_ACT := setp_level, DESTIN :=0 ); (* Destination ist angle 0*)
  setp_level := auto_plat.SETP; (* Activate autolevelling*)
else
  setp_level := level_joy;  (* Manual movement*)
end_if;
prop(CHANNEL :=34, SETPOINT := auto_plat.SETP, OVERRIDE := 1000);
```

FAHR_POS

The two function blocks FAHR_POS and AUTO_MOVE have the same functionality. The name AUTO_MOVE should be used.

12.6.6 GET_STATUS

For easier processing errors are delivered as bits.

GET_STATUS get_status(CHANNEL:= | :=PLVC_STAT);	
Inputs	
INT	CHANNEL *possible values:* *0: Prop. valve range error* *1: Prop. valve cable break* *2: Prop. valve short circuit* *3: Dig. output short circuit* *4: Dig. output misplaced* *5: Position control prop. moved to far* *6: Position control prop. moved to less* *7: See 5, but quasiprop* *8: See 6, but quasiprop* *9: Cable break/short circuit analog in 0-15*

Continued on the next page. . .

. . .continued from previous page

	10: Cable break/short circuit analog in 16-31 *11: Cable break/short circuit analog in 32 -47* *12: Internal errors: 1: CAN bus off, 2: CAN warn, 8 CRC flash* *20: Reads dig. inputs 0.0-1.7 at this position in program* *21: Reads dig. inputs 2.0-3.7 at this position in program* *22: Dig. inputs 4.0-5.7* *23: Dig. inputs 6.0-7.7* *24: Dig. inputs 8.0-9.7* *25: Dig. inputs 10.0-11.7* *26: Dig. inputs 12.0-13.7* *27: Dig. inputs 14.0-15.7*
Outputs	
WORD	PLVC_STAT *return value of type WORD*

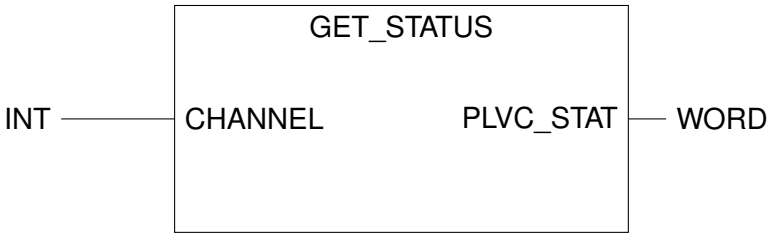


Figure 12.61.: Function GET_STATUS

Description:

- GET_STATUS reads the extended errors. Via CHANNEL you can select the range of channels you want to check.
- TYPE WORD has 16 bits. Each of them is for one channel.

  Note: Return value: PLVC_STAT of TYPE WORD:

  - Bit 0 → Error in channel 0
  - Bit 1→ Error in channel 1
  - and so on. . .
  - Bit 15→ Error in channel 15
- With this function you can get errors which you otherwise would not get. This is useful e.g. for anglesensors which produce a return value of -1000 to 1000 and return a zero on error. So an angle of “0” does not differ from an error. Another application would be the parallel moving of cylinders. If one cylinder does fail, you can get the error (probably an OPEN-error), and you can fast switch it off.

Example: Cabel brak detection for analog channels 40, 41, 42, 43

Variable declaration:

```
VAR
  error: GET_STATUS;
  kb40: BOOL;
  kb41: BOOL;
  kb42: BOOL;
  kb43: BOOL;
END_VAR
```

Program:

```
error( CHANNEL := 11 ); (* ana40 and following *)
kb40:=0;
kb41:=0;
kb42:=0;
kb43:=0;
if ((error.plvc_stat and 16#0100)>0) then kb40 := 1; end_if;
if ((error.plvc_stat and 16#0200)>0) then kb41 := 1; end_if;
if ((error.plvc_stat and 16#0400)>0) then kb42 := 1; end_if;
if ((error.plvc_stat and 16#0800)>0) then kb43 := 1; end_if;
```

Output of PLVC_STAT:
PLVC_STAT consists of 16 bits, that (in the exampe above) show the state of the cable break detection of an analog input.
To get the value of a single bit you have to combine PLVC_STAT with a bit word using the AND function. If one of the variables is “TRUE”, there is an error in this channel.

16#0100 means (binary): 0000 0001 0000 0000

PLVC_STAT																	
Analog input	Binary code																Hex code
Ana32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
Ana33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
Ana34	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0004
Ana35	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0008
Ana36	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0010
Ana37	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0020
Ana38	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0040
Ana39	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0080
Ana40	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0100
Ana41	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0200
Ana42	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0400
Ana43	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0800
Ana44	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000
Ana45	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
Ana46	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
Ana47	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000

12.6.7 Mathematical Function

MW_EX

Get Average Value

MW_EX mw_ex(IST_WERT:=,LENG:= | :=M_WERT);	
Inputs	
INT	IST_WERT *value to be filtered*
INT	LENG *number of the last incoming values to be filtered* *possible values: 1 (unfiltered) to 32 (most precise filtering)*
Outputs	
INT	M_WERT *fitered value*

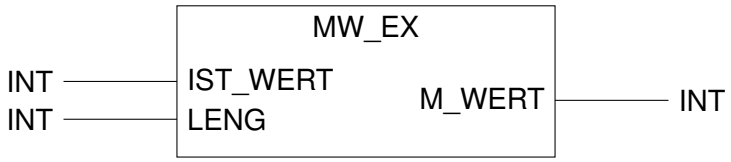


Figure 12.62.: Function MW_EX

Description of the parameters:

- *IST_WERT* Value that should be filtered
- *LENG* Amount of averaging
- *M_WERT* Filtered value

Note:

Return value is the average of the last LENG IST_WERTE in M_WERT.

Example: Filter analog input of angle of inclination

Variable declaration:

```
VAR
  inclination AT %IW50.0: INT;
  avg_cnt : int := 4;  (* Amount of averaging of the angle of
 inclination, 1 = unfiltered, max= 32 *)
  filter : MW_EX;
  incl_filtered : INT;
END_VAR
```

Program:

```
filter( IST_WERT :=inclination, LENG := avg_cnt );
incl_filtered := filter.M_WERT ;
```

AXB

Linear scaling with limit of range.

AXB

axb(X:=,X1:=,Y1:=,X2:=,Y2:=,MAX_Y:=,MIN_Y:= | :=Y);

Inputs

INT	X *input*

Continued on the next page. . .

. . .continued from previous page

INT	X1 *x-value of the first pair*
INT	Y1 *y-value of the first pair*
INT	X2 *x-value of the second pair*
INT	Y2 *y-value of the second pair*
INT	MAX_Y *maximum of Y*
INT	MIN_Y *minimum of Y*
Outputs	
INT	Y *calculated value*

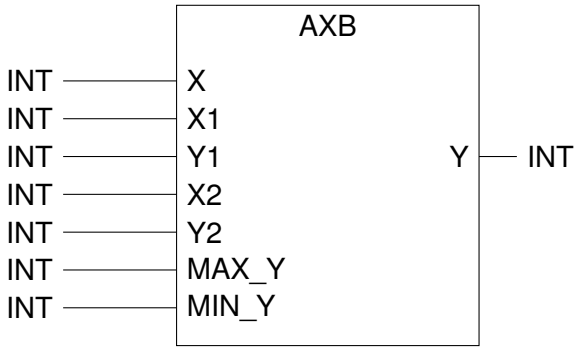


Figure 12.63.: Function AXB

Description of the parameters:

- *X:* Input value
- *X1* = X-value of the first pair of values
- *Y1* = Y-value of the first pair of values
- *X2* = X-value of the second pair of values
- *Y2* = Y-value of the second pair of values
- *MAX_Y:* Upper limit of Y
- *MIN_Y:* Lower limit of Y
- *Y:* Calculated value

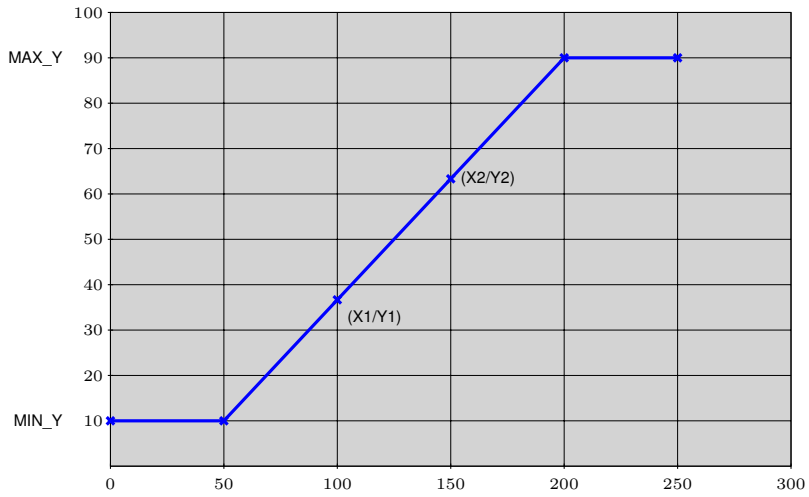


Figure 12.64.: Function of AXB

Note:

A linear scaling of type $Y = A \cdot X + B$ is computed and given back in value Y, where A and B are internally determined by the two pairs $X1, Y1$ and $X2, Y2$.

Example:

You need 1000 at 80 degrees and 0 at 110 degrees, linear mapping and a limitation between 0-1000.

Variable declaration:

```
VAR
  axb_tele: AXB;
  tele_len at %IW104.0: INT;
  speed : INT ;
END_VAR
```

Program:

```
axb_tele ( X :=tele_len,
  X1 :=80,
  Y1 :=1000, (*100%*)
  X2 :=110,
  Y2 :=0,    (*0%*)
  MAX_Y :=1000,
  MIN_Y :=0 );
speed := axb_tele.y;
```

ABK

Bent curve in the range from 0 to 1000

ABK abk(X:=, X1:=, Y1:= | :=y);	
Inputs	
INT	X *possible values: 0-1000*
INT	X1 *X-value of the pair of variates which defines the bend; possible values: 0-1000*
INT	Y1 *Y-value of the pair of variates which defines the bend; possible values: 0-1000*
Outputs	
INT	Y *Y-value to the entered X-value; possible values: 0-1000*

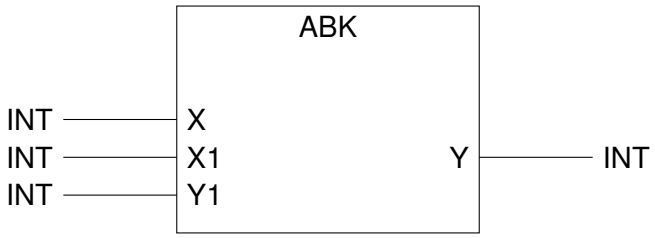


Figure 12.65.: Function ABK

Example: precision control for proportional output; onls 30% speet at 50joystick:

Variable declaration:

```
VAR
  abk_fein: ABK;
  y:INT;
END_VAR
```

Program:

```
abk_fein(X:=setpoint,
    X1:=500,
    Y1:=300 (*30%*)
    );
    setpoint_fein:=abk_fein.y;
```

Mapping of a range 0 to 1000 with bend, which is defined by the pair of values (X1,Y1). Within 50% deflection of the joystick, there is only 30% output-current. Thereby the output is more easy to be controlled in lower ranges.

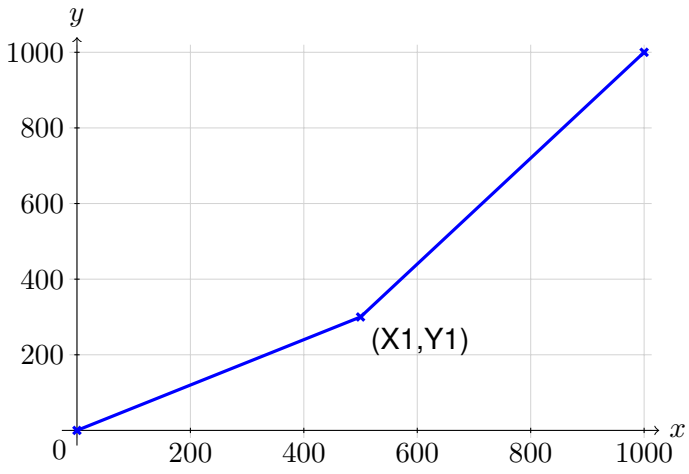


Figure 12.66.: Characteristic curve for values X1=500, Y1=300;

The pair of values (X1,Y1) should be written only one time at the beginning.

12.6.8 CAN Bus

CAN_WRITE

Write on the CAN Bus

CAN_WRITE can_write(ID:=,LENG:=,RTR:=,B0:=,B1:=,B2:=,B3:=,B4:=,B5:=,B6:=,B7:=);	
Inputs	
INT	ID *CAN-ID of the receiver, possible values: 0-2050*
USINT	LENG *Length of the telegram, possible values: 0-8*
USINT	RTR *Remote request, 0=CAN-Bus1, 2=CAN-Bus2 (PLVC8)*
USINT	B0-B7 *Data to be sent*

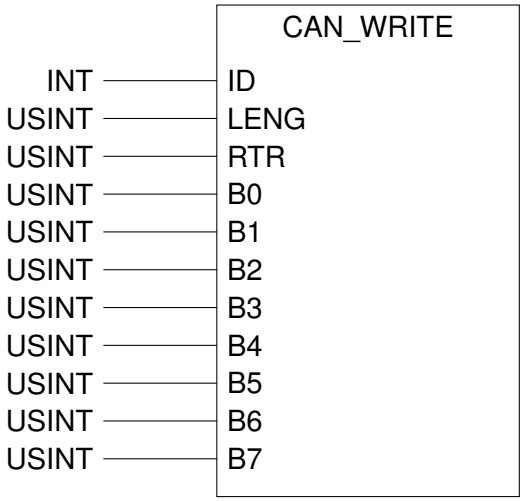


Figure 12.67.: Function CAN_WRITE

Example:

Variable declaration:

```
VAR
  can_w  : CAN_WRITE;
  data_b0  : USINT;
  data_b1  : USINT;
  data_b2  : USINT;
  data_b3  : USINT;
  data_b4  : USINT;
  data_b5  : USINT;
  data_b6  : USINT;
  data_b7  : USINT;
END_VAR
```

Program:

```
can_w( ID    := 16#680,
  LENG := 8,
  B0    := data_b0,
  B1    := data_b1,
  B2    := data_b2,
  B3    := data_b3,
  B4    := data_b4,
  B5    := data_b5,
  B6    := data_b6,
  B7    := data_b7
 );
```

CAN_WRITE_BYTE

Write a 11bit CAN message - BYTE values

CAN_WRITE_BYTE can_write_byte(ID:=, LENG:=, RTR:=, B0:=, B1:=, B:=, B3:=, B4:=, B5:=, B6:=, B7:=);	
Inputs	
INT	ID *CAN_ID of the receiver, possible values: 0-2050*
BYTE	LENG *Length of the telegram*
BYTE	RTR *Remote request, 0=CAN-Bus1, 2=CAN-Bus2 (PLVC8)*
BYTE	B0-B7 *Data to be sent*

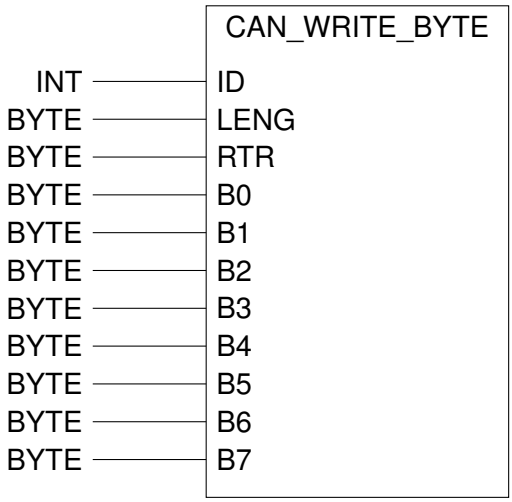


Figure 12.68.: Function CAN_WRITE_BYTE

CAN_WRITE_INT

Write on the CAN Bus (INT)

CAN_WRITE_INT can_write_int(ID:=,LENG:=,RTR:=,I0:=,I1:=,I2:=,I3:=);	
Inputs	
INT	ID *CAN-ID of the receiver, possible values: 0-2050*
BYTE	LENG *Length of the telegram, possible values: 0-8*
BYTE	RTR *Remote request, 0=CAN-Bus1, 2=CAN-Bus2 (PLVC8)*
INT	I0-I3 Data to be sent

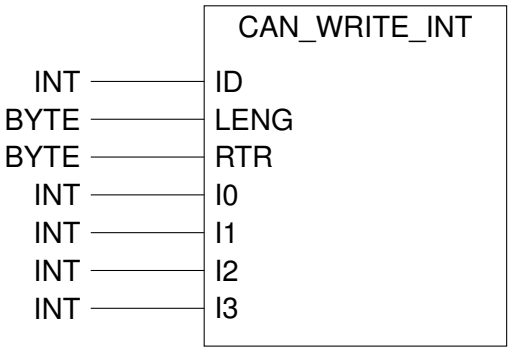


Figure 12.69.: Function CAN_WRITE_INT

Example: Send integer values over the CAN bus

Variable declaration:

```
VAR
  can_w_int : CAN_WRITE_INT;
  data_i0 : INT;
  data_i1 : INT;
  data_i2 : INT;
  data_i3 : INT;
END_VAR
```

Program:

```
can_w_int(  ID   := 16#680,
      LENG := 8,
      I0   := data_i0,
      I1   := data_i1,
      I2   := data_i2,
      I3   := data_i3);
```

CAN_WRITE_29

Write on the CAN Bus (29Bit)

CAN_WRITE_29 can_write_29(ID:=,ID2:=,LENG:=,RTR:=,B0:=,B1:=,B2:=,B3:=);	
Inputs	
WORD	ID *CAN-ID 1 of the receiver, possible values: 0-2050*
INT	ID2 *CAN-ID 2 of the receiver, possible values: 0-2050*
USINT	LENG *length of the telegram*
USINT	RTR *Remote request, 0=CAN-Bus1, 2=CAN-Bus2 (PLVC8)*
USINT	B0-B7 Data to be sent

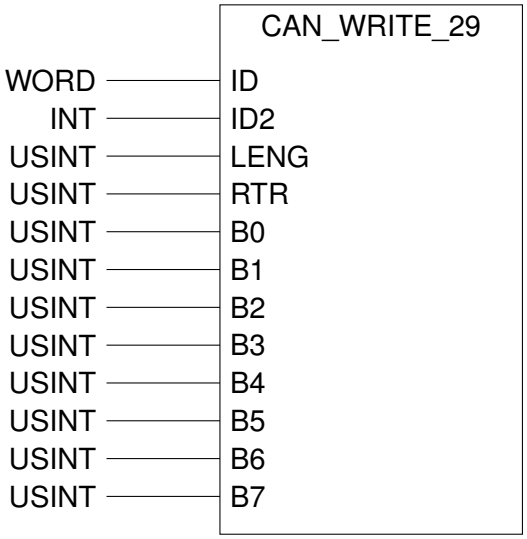


Figure 12.70.: Function CAN_WRITE_29

Example:

Variable declaration:

```
VAR
  can_w29  : CAN_WRITE;
  data_b0  : USINT;
  data_b1  : USINT;
  data_b2  : USINT;
  data_b3  : USINT;
  data_b4  : USINT;
  data_b5  : USINT;
  data_b6  : USINT;
  data_b7  : USINT;
END_VAR
```

Program:

```
can_w29( ID    := 16#680,
  ID2  := 16#10a0,
  LENG := 8,
  B0   := data_b0,
  B1   := data_b1,
  B2   := data_b2,
  B3   := data_b3,
  B4   := data_b4,
  B5   := data_b5,
  B6   := data_b6,
  B7   := data_b7
 );
```

CAN_REC_INI

Initialize CAN Bus

CAN_REC_INI can_rec_ini(CHANNEL:=,ID:=,_29:= | :=VALID);	
Inputs	
INT	CHANNEL *Select the channel you want to read. possible values: 0-24 for BUS1, 200-224 for BUS2*
WORD	ID *CAN-ID where you want to read from, possible values: 0-2050*
INT	_29 *Activate 29bit identifier. Possible values: 0 = 11bit, 1 = 29bit*
WORD	ID2 *Set CAN ID2. Only necessary for 29bit identifier.*
Outputs	
INT	VALID *Indicates successful initialization.*

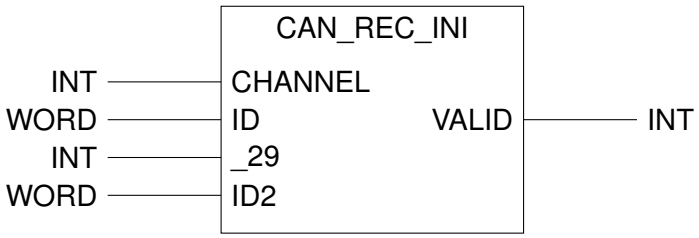


Figure 12.71.: Function CAN_REC_INI

Example: Initialize channel 1 and 2

Variable declaration:

```
VAR
    can_i : CAN_REC_INI;
    ini : BOOL;
END_VAR
```

Program:

```
IF NOT ini THEN
  ini := 1;
  can_i(CHANNEL:=1, ID:=16#295);
  can_i(CHANNEL:=2, ID:=16#296);
END_IF;
```

CAN_READ

Read from the CAN Bus

CAN_READ can_read(CHANNEL:= | :=ID,:=VALID,:=LENG,:=B0,:= B1,:= B2,:= B3);	
Inputs	
INT	CHANNEL *Select the channel you want to read.* *Possible values: 0-24 for BUS1, 200-224 for BUS2*
Outputs	
INT	ID *CAN ID of the CAN message, possible values: 0-2048*
USINT	VALID *Indicates a valid message reception.*
USINT	LENG *Length of the CAN message.*
USINT	B0-B7 *Data of the message.*

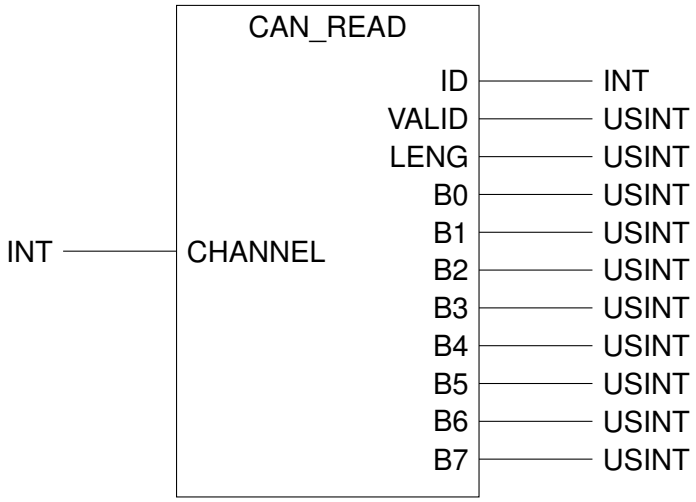


Figure 12.72.: Function CAN_READ

Note:

As soon as a CAN message arrives, VALID is set to 1, otherwise it is set to 0. Only if VALID is 1, the other values are valid:
ID = CAN identifier
LENG = Length in byte
B0-B7 Data bytes
Only messages initialized with CAN_REC_INI can be received.

Example:

Variable declaration:

```
VAR
  can_i : CAN_REC_INI;
  ini : BOOL;
  can_r : CAN_READ;
  value1 : USINT;
  value2 : INT;
  setpoint : INT;
  signals AT %QB3.0 : USINT;
  signal_0 AT %QB3.0 : BOOL;
  signal_1 AT %QB3.1 : BOOL;
  ...
  ...
END_VAR
```

Program:

```
IF NOT ini THEN
  ini := 1;
  can_i(CHANNEL:=1, ID:=16#295);
END_IF;

can_r(CHANNEL:=1);
IF (can_r.valid>0) THEN  (* CAN message valid *)
  value1 := can_r.b0;    (* value1 = USINT *)
  value2 := usint_to_int(can_r.b1); (* value2 = INT *)
(* Join two USINT to one INT *)
  setpoint := usint_to_int(can_r.b2);
  setpoint := setpoint*256; (* Shift 8bit left *)
(* Add second part *)
  setpoint := setpoint + word_to_int(usint_to_word(can_r.b3));
(* Read one bit from USINT *)
  signal_0 := can_r.b4;
END_IF;
```

CAN_READ_BYTE

Read Byte from CAN Bus

CAN_READ_BYTE CHANNEL:= | := ID,:= VALID,:= LENG,:= B0,:= B1,:= B2,:= B3,:= B4,:= B5,:= B6,:= B7);	
Inputs	
INT	CHANNEL *Select the channel you want to read.* *Possible values: 0-24 for BUS1, 200-224 for BUS2*
Outputs	
INT	ID *CAN ID of the CAN message, possible values: 0-2048*
USINT	VALID *indicates a valid message reception.*
USINT	LENG *Length of the CAN message.*
USINT	RTR *Remote Request, possible values: 0-1*
BYTE	B0-B7 *Data of the message.*

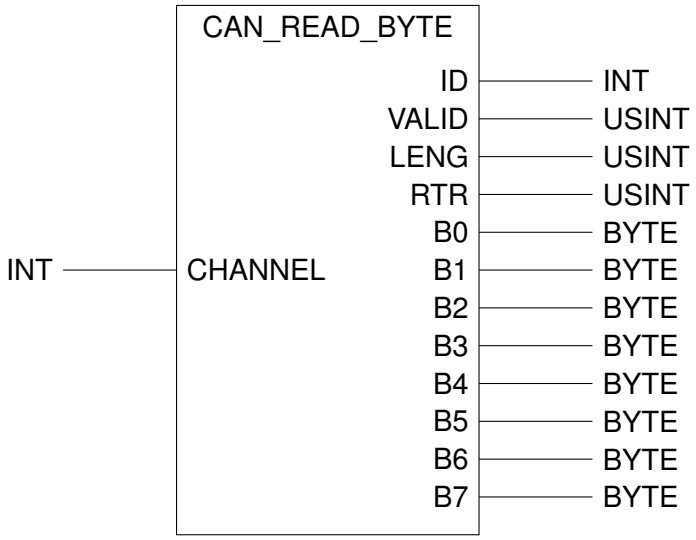


Figure 12.73.: Funktion CAN_READ_BYTE

CAN_READ_4INT

Read a CAN message - Integer format

CAN_READ_4INT can_read_4int(CHANNEL:= | := ID, := VALID, := LENG, := I0, := I1, := I2, := I3);	
Inputs	
INT	CHANNEL *Select the channel you want to read.* *possible values: 0-24 for BUS1, 200-224 for BUS2*
Outputs	
INT	ID *CAN ID of the CAN message, possible values: 0-2048*
USINT	VALID *Indicates a valid message reception*
USINT	LENG *Length of the CAN message*
INT	B0-B7 *Data of the message*

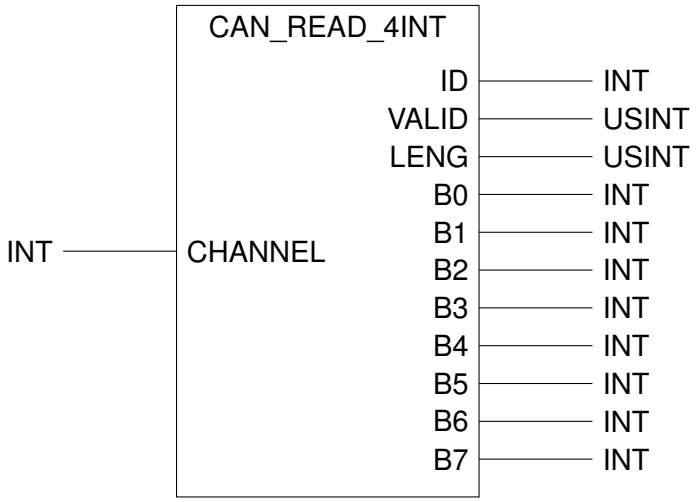


Figure 12.74.: Function CAN_READ_4INT

CAN_READ_2DINT

Read a CAN message - Double Integer format

CAN_READ_2DINT can_read_2dint(CHANNEL:= | := ID, := VALID, := LENG, := D0, := D1);	
Inputs	
INT	CHANNEL *Select the channel you want to read.* *Possible values: 0-24 for BUS1, 200-224 for BUS2*
Outputs	
INT	ID *CAN ID of the CAN message, possible values: 0-2048*
USINT	VALID *Indicates a valid message reception*
USINT	LENG *Length of the CAN message*
DINT	D0-D1 *Data of the message*

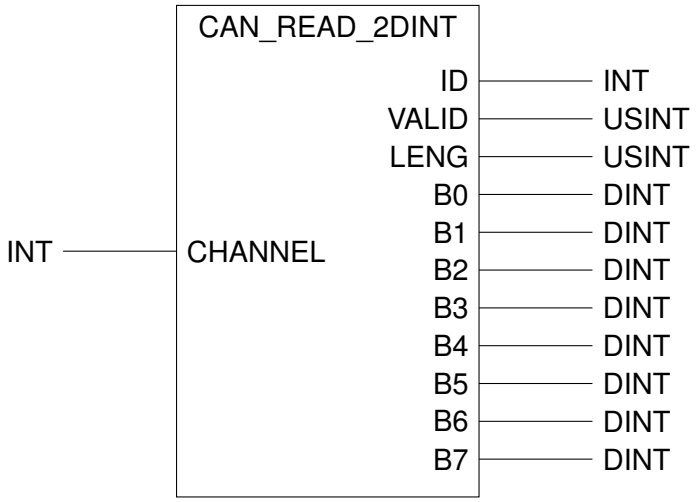


Figure 12.75.: Function CAN_READ_2DNT

12.6.9 Functionblocks following the IEC61131

CTD

This functionblock is used as a decrementer for pulses

CTD ctd(CD:=,LOAD:=,PV:= | :=Q,:=CV);	
Inputs	
BOOL	CD *meter pulse*
BOOL	LOAD *set counter to PV*
INT	PV *value to be set*
Outputs	
BOOL	Q *output signal when CV reaches zero*
INT	CV *actual count*

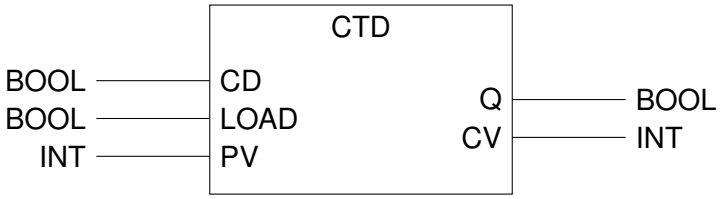


Figure 12.76.: Function CTD

Description

- Counter is set zero within the initialisation
- If LOAD=1, PV is set as the actual count
- Every rising edge at input CD decreases the actual count by one
- CV contains the actual count

- Q switches from zero to one as soon as CV runs zero

CTU

The functionblock CTU is used for incrementing pulses.

CTU ctu(CU:=,RESET:=,PV:= | :=Q,:=CV);	
Inputs	
BOOL	CU *meter pulse*
BOOL	RESET *set initial value*
INT	PV *value to be reached*
Outputs	
BOOL	Q *output signal; possible values: 0, 1*
INT	CV *actual count*

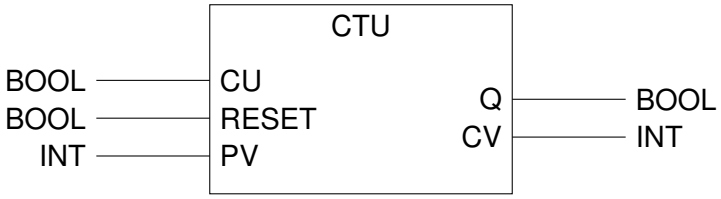


Figure 12.77.: Function CTU

Beschreibung

- Incoming Pulses at CU get counted
- Counter is set zero due to initialisation
- Actual count can be resetted by setting RESET:=1
- Every rising edge at CU increments the counter
- The operand CV outputs the actual count
- Q switches from zero to one as soon as CV>=PV

CTDU

The functionblock CTDU is used for incrementing and decrementing pulses.

CTDU ctdu(CU:=, CD:=,RESET:=,LOAD:=,PV:= | :=QU,:=QD,:=CV)	
Inputs	
BOOL	CU *meter pulse for incrementing, rising edge*
BOOL	CD *meter pulse for decrementing, rising edge*
BOOL	RESET *set initial value*
BOOL	LOAD *set counter to PV*
INT	PV *value to be reached*
Outputs	
BOOL	QU *signal if value PV is reached*
BOOL	QD *signal if zero is reached*
INT	CV *actual count*

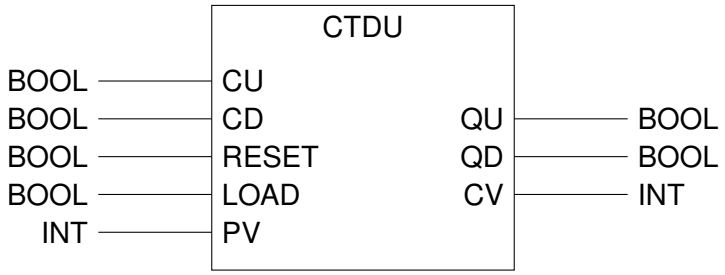


Figure 12.78.: Function CTDU

Description

- Rising edge at input CU (CD) increments (decrements) the counter by one
- If LOAD:=1, the value of PV is set as the counter
- If RESET:=1, the value of the counter is resetted
- As long as RESET equals one, neither the rising edge at PU nor at PD nor the loadingconditions have any influence on the counter
- CV showes the actual count
- QU switches from zero to one as soon as the value of CV reaches CU
- QD switches from zero to one as soon as the value of CV reaches zero

R_TRIG

The functionblock R_TRIG is used to detect a rising edge.

R_TRIG r_trig(CLK:= | :=Q);	
Inputs	
BOOL	CLK Possible values: TRUE, FALSE
Outputs	
BOOL	Q Possible values: TRUE, FALSE

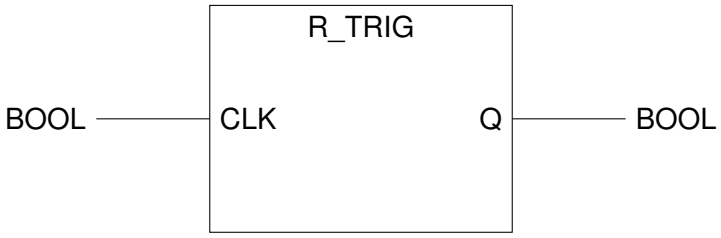


Figure 12.79.: Function R_TRIG

Description

- Ouptput Q is set TRUE as soon as a rising edge at CLK is detected

RS

RS rs(SET:=,RESET1:= | :=Q1);	
Inputs	
BOOL	SET *Possible values: TRUE, FALSE*
BOOL	RESET1 *Possible values: TRUE, FALSE*
Outputs	
BOOL	Q1 *Possible values: TRUE, FALSE*

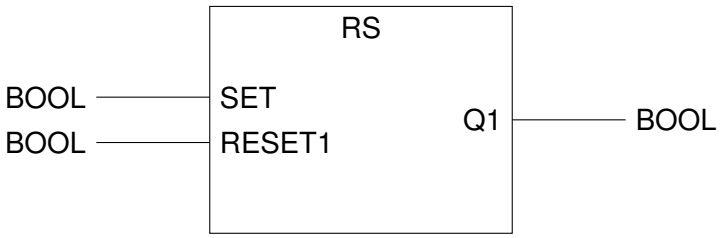


Figure 12.80.: Function RS

Description

The functionblock RS returns a logical value within the Output Q1. This value is the result of the following expression: NOT(RESET1)AND(Q1 OR SET)

The following table shows the possible combinations and the related values of Q1:

RS		Set, Reset1			
		00	**01**	**11**	**10**
Q1	**0**	0	0	0	1
	1	1	0	0	1

NOTE

Keep in mind that the Input RESET1 is dominant in functionblock RS!

SR

SR SR(SET1:=,RESET:= | :=Q1);	
Inputs	
BOOL	SET1 *Possible values: TRUE, FALSE*
BOOL	RESET *Possible values: TRUE, FALSE*
Outputs	
BOOL	Q1 *Possible values: TRUE, FALSE*

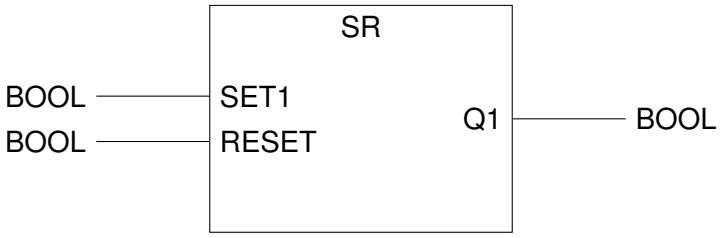


Figure 12.81.: Function SR

Description

The functionblock SR returns a logical value within the Output Q1. This value is the result of the following expression: SET1 OR (Q1 AND NOT RESET)

The following table shows the possible combinations and the related values of Q1:

SR		SET1, RESET			
		00	**01**	**11**	**10**
Q1	**0**	0	0	1	1
	1	1	0	1	1

NOTE

Keep in mind that the Input SET1 is dominant in functionblock SR!

12.7 Further Sample Programs

12.7.1 Sample with GET_EE

In this program the example from chapter Programming Example (12.3.3) is extended. The slow down of the “slow”-switch should be changeable without downloading a new program. So the value of slowspeed is loaded from a user parameter. This is only done once at startup. So if you have changed it you need to restart the PLVC. You could also move the part with the para out of the “if not ini then” condition, then it would be read steady. This would you enable to change the parameter (e.g. with the CAN BC) and check the changes immediately. But if you need many user parameters and have a complicated program you may want to have a faster speed, and put it in the ini-clause.

```
VAR
  joy AT %IW64.0: int;     (*joystick analog input *)
  setp: int;
  prop: ACT_VALVE;  (* FB *)
  speed:  int;  (* the maximum speed *)
  enable_di AT %IB0.0:  bool; (* enable switch *)
  slow_di AT %IB0.2:  bool; (* digital input for slow down *)
  ini:  bool;
  horn_di AT %IB1.0:  bool;
  horn_do AT %QB3.2:  bool;
  para: GET_EE;   (* FB for reading user parameters *)
  slowspeed:  int;  (* holds the read value *)
END_VAR

(* the following is only executed once at reset *)
if not ini then
  ini := true;
  para( CHANNEL :=0);
  slowspeed := para.EE_VAL;
end_if;

(* reset variables *)
setp := 0;
speed := 1000;

if slow_di then
  speed := slowspeed;  (* setting slowspeed speed *)
end_if;

if enable_di then setp := joy; end_if;

(* assignement of horn_di to horn_do. this will switch the horn trough horn_di*)
horn_do := horn_di;

(* set the valves according to the variables *)
prop( CHANNEL := 10, SETPOINT := setp, OVERRIDE := speed );
```

12.7.2 Sample with the Display (DISP_TXT, DISP_VAL)

In this example it is shown how to use the small display to show texts.

```
VAR
  ini:  BOOL;
  print:  DISP_TXT;
  printval: DISP_VAL;
  setdisp:  PUT_PAR;  (* for switching pages in display *)
  right_key AT %IB15.3: BOOL; (* the right key of the display *)
```

```
  left_key AT %IB15.2:  BOOL; (* the left key of the display *)
  act_page: INT := 0; (* holds the visible page of display *)
  act_page_old: INT := -1;  (* holds the old value of act_page *)
  pressure_in AT %IW104.0:INT;  (* analog input of pressure *)
  counter:  INT;  (* counter to slow down display rate *)
END_VAR

if not ini then
  ini := true;
  print( CHANNEL := 2,    (* line 1 of set 0 *)
  OFFSET := 3,    (* 16 chars are available, 3 blanks left *)
  PUT_INFO := 'Hello world'); (* and 11 chars - it will be centered *)

  print( CHANNEL := 3,    (* line 2 of set 0 *)
  OFFSET := 0,    (* no blanks left *)
  PUT_INFO := 'Pressure     psi');  (* later the pressure
  read out by IW104 will be written in those 5 blanks *)

  print( CHANNEL := 4,    (* line 1 of set 1 *)
  OFFSET := 0,
  PUT_INFO := 'right menu');

  print( CHANNEL := 5,    (* line 2 of set 1 *)
  OFFSET := 0,
  PUT_INFO := 'some Text');
end_if;

(* read the keys *)

if right_key then act_page := 0; end_if;
if left_key then act_page := 1; end_if;

counter := counter + 1;
if (act_page = 0) and (counter >= 50) then
  counter := 0;
    printval( CHANNEL :=3, OFFSET :=8, LENG :=5, VAL := pressure_in );
      (* this prints the pressure on line 2 of set 0 on position 8 to 12 *)
end_if;
if not (act_page = act_page_old) then   (* if the page changed *)
  setdisp( CHANNEL :=4,   (* channel 4 for display *)
  PARA := act_page );   (* act_page=0 -> set=0, act_page=1 -> set=1 *)
  act_page_old := act_page; (* update the act_page_old *)
end_if;
```

12.7.3 Sample with AND/OR

This example shows you how to use the logical operations.

If you have a big dangerous machine with several security mechanism, the following condition must be satisfied:

- `Light\_barrier1` must be true
- `Light\_barrier2` must be true
- `Step\_sw` must be false
- If in testmode (`testmode` = true) then ignore the lightbarriers and the `step\_sw`
- `Oper\_mode` must be between 1 and 4
- `Em\_out` has to be false

There are some explained assignments before the complicated condition.

```
VAR
  light_barrier1 AT %IB0.0: BOOL; (* light barrier 1 *)
  light_barrier2 AT %IB0.1: BOOL; (* light barrier 2 *)
  step_sw AT %IB0.2:  BOOL; (* step down switch *)
  mod_bt_old: BOOL; (* save for mod_bt *)
  mod_bt AT %IB0.3: BOOL; (* button to switch operational mode *)
  oper_mode:  INT;  (* operational mode *)
  em_out AT %IB3.7: BOOL; (* emergency out *)
  hydraulic_on AT %IB1.0: BOOL; (* if the system is under pressure *)
  testmode: BOOL; (* machine in testing modus *)
END_VAR


(* if no pressure the machine is in testmode
  So we can write:

   if hydraulic_on then
     testmode := true;
   else
     testmode := false;
   end_if;

  But it is way shorter to write: *)
testmode := not hydraulic_on;


(* We want to detect a rising edge of mod_bt. So it is a usual way to define a
    auxiliary
   variable which holds the last state of mod_bt. Then if have to detect if it is
    now
   true and it was false: *)

if mod_bt and not mod_bt_old then
  oper_mode := oper_mode + 1;
```

```
33      if oper_mode > 5 then oper_mode := 0;
        end_if;
35  end_if;
    mod_bt_old := mod_bt;
37  (* now the complicated if-clause. It is splitted over several lines to include the
        comments *)

39  if (
        (light_barrier1 and light_barrier2 and not step_sw)
41                                  (* rather than writing "(light_barrier = true)
        and...." we can
                                       cut the "= true". The meaning is the same. *)
43      or
         testmode
45      )                           (* see the brackets! testmode is combined with the
        long
                                       condition with the lightbarriers. So if testmode
        is on
47                                     the lightbarriers are nonrelevant *)
        and
49       (oper_mode < 4) and (oper_mode > 1) (* there is no condition for ranges so we
        have to
                                               make two conditions here *)
51      and not em_out then

53  (* do some dangerous things *)
    end_if;
```

Part IV.

CAN Bus

13 PLVC and CAN Bus

13.1 Introduction

The PLVC is able to communicate with other components. The realisation requires a capable wiring and several different settings. This chapter contains some basic information about the topic "CAN bus", but raise no claim to completeness. For more comprehensive and detailed information, please consult the relevant literature.

The articles we will discuss in this manual shall help you to use the CAN functionality of your HAWE-products effective and in a proper way.

13.2 Installing

In order to guarantee a possible interchange, the participants need an undisturbed connection.

13.2.1 Topology

The CAN bus operates with a duplex cable[1]. Thereby the method of wiring is very important.

The connection between the participants is electrical parallel. The conductors mustn't hit in a central point, so avoid a star-shaped connection (illustration 13.1).

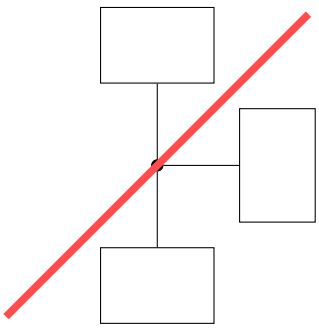


Figure 13.1.: Incorrect Wiring

The signal has to pass the participants one after another. This is achieved by continuing the cable from device to another (illustration 13.2). Now all clamps CAN_HIGH will be linked consecutively and all clamps CAN_LOW too.

[1] Technically a three-conductor-cable is scheduled, one for each conductor: CAN_HIGH, CAN_LOW and ground. Especially the full isolated devices are in need of a seperate connection to ground. In our devices PLVC and HMI the duplex cable delivers the high and the low signal, ground is delivered by the power supply system.

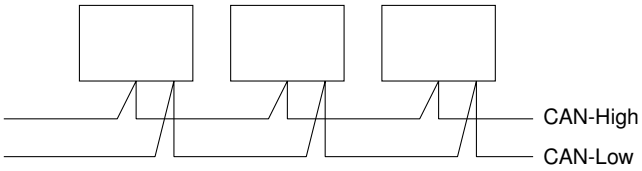


Figure 13.2.: Direct CAN Bus Access

In case of long data lines you can use short branch lines to connect single devices (illustration 13.3).

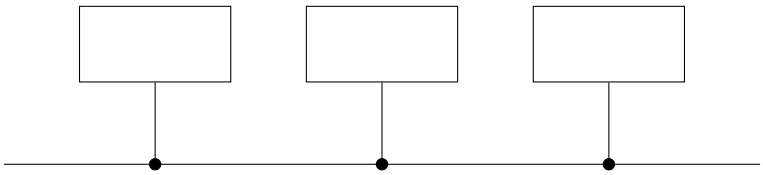


Figure 13.3.: CAN Bus Access with Short Branch Lines

13.2.2 Connection Terminals

The connection terminals for the CAN bus are labeled in the terminal diagram from PLVC and CAN HMI with CAN_H and CAN_L. The chart 13.1 summarizes the devices and their connecting points.

Device	CAN_H	CAN_L
PLVC8*2	M2	M3
PLVC41	X11.28	X11.29
PLVC2 Add-on	X6.1	X6.2
HMI	X3.1 X3.5	X3.2 X3.6
HMI-XS	X3.2 X3.3	X3.8 X3.9

Table 13.1.: CAN Connection Terminals from PLVC and HMI

13.2.3 Termination Resistor

After the wiring the devices at each end of the cable (the first and the last one) need an 120Ω termination resistor. In most of the devices such a resistor is already integrated. In case of a correct connection, the multimeter has to quantify the resistance between both conductors with 60 Ω (please note that the devices must be zero potential before qauging).

All discussed HAWE-devices have an integrated termination resistor, which can be manually activated by connecting the clamps:

- X11.29 and X11.30 at PLVC41
- X3.1 and X3.2 at HMI
- X3.6 and X3.7 at HMI-XS
- X6.2 and X3.7 at PLVC2 - add on
- M3 and N2 at PLVC8*2.

All activating bridges must be placed right next to the device. Connections via external clamps might interfere with the bus.

13.2.4 Cable and Cable Installation

The HAWE CAN bus is a two lead system. To avoid disturbing interferences the used cable should be installed aparted from the power cable. A shielding is recommendable. For very short distances it's enough to use a twisted-pair-cable to eliminate the interferences.

Figure 13.4 shows one possible example of shielding the bus lines. One can alternatively use decoupling capacitors between the shield and the ground.

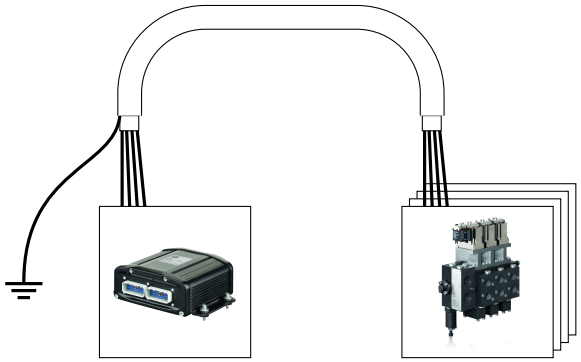


Figure 13.4.: Shielding bus lines

The standard ISO 11898-2 recommends twisted-pair-cables with a characteristic impedance of 108-132 Ω. In trade different cables are available for the CAN bus.

Theoretical it is possible to cover 5000 metres with one cable. In reality this distance can not accomplish, because the bus length is primarily limited by the favoured transmission speed. So the transfer rate (baud rate=bit/sec) limits the maximum possible bus length.

The chart 13.2 can be used as indication. The denoted bus lengths should be as undercut as possible. If necessary, a lower transfer rate must be selected.

baudrate	bus length	maximum length for tap line
1 Mbit/sec	20m	1m
800 kBit/sec	50m	3m
500 kBit/sec	100m	5m
250 kBit/sec	250m	10m
125 kBit/sec	500m	20m
50 kBit/sec	1000m	25m

Continued on the next page. . .

. . . Continued from previous Page

20 kBit/sec	2500m	250m

Table 13.2.: Recommended Bus Line Length Limits

While selecting the bus length note that many equipment manufacturer operate with constant baud rates or complicate the baud rate change. So changing the baud rate could be impossible or difficult and extensive. This can then result in a maximum line length.

13.3 Basic Setting

Two settings are necessary at each CAN participant for a possible communication via the CAN bus.

13.3.1 CAN Address (Device Address)

The CAN bus has no unique master-control. All participants are equal and identify themselves with a “CAN address”. This CAN address has to be different between all devices. It is deposited in the terminal program after the login in the last row (illustration 13.5).

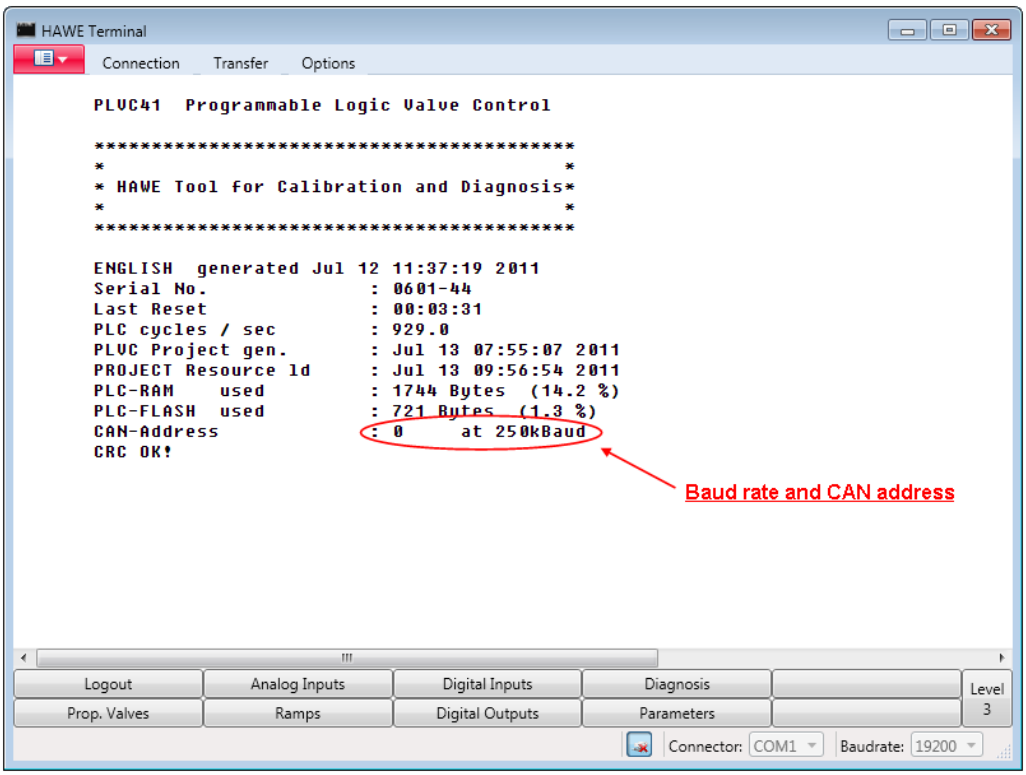


Figure 13.5.: Start Screen of the Terminal Program

With the buttons at the bottom of the terminal program (illustration 13.6) you can reach the parameter menu.

The setting of the CAN address takes place in the menu:

Parameters (illustration 13.6) → **Submenu 4: Communication** (illustration 13.7) → **Digital Inputs (a) Transmit** (illustration 13.8)

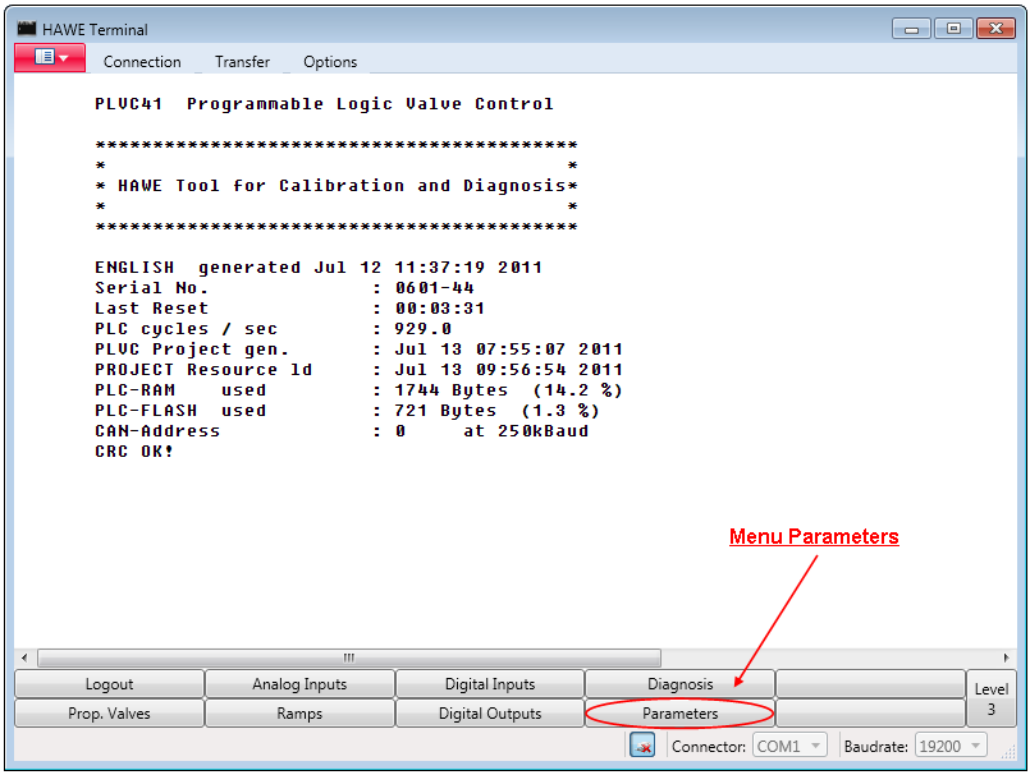


Figure 13.6.: Parameter Menu

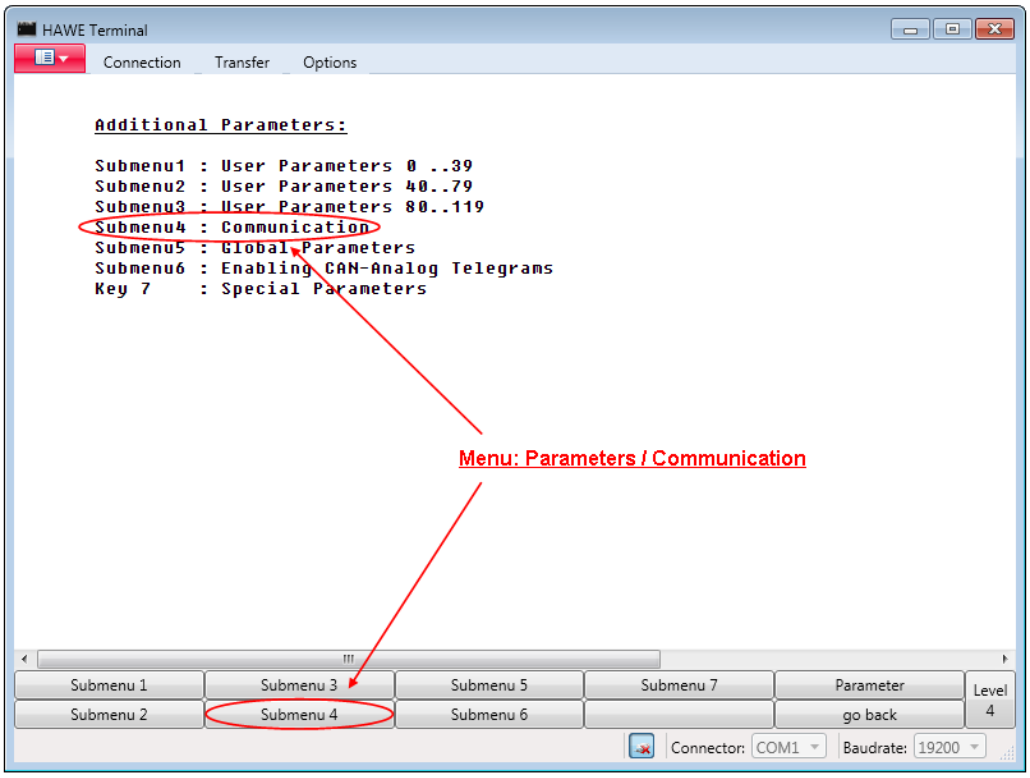


Figure 13.7.: Communication Menu

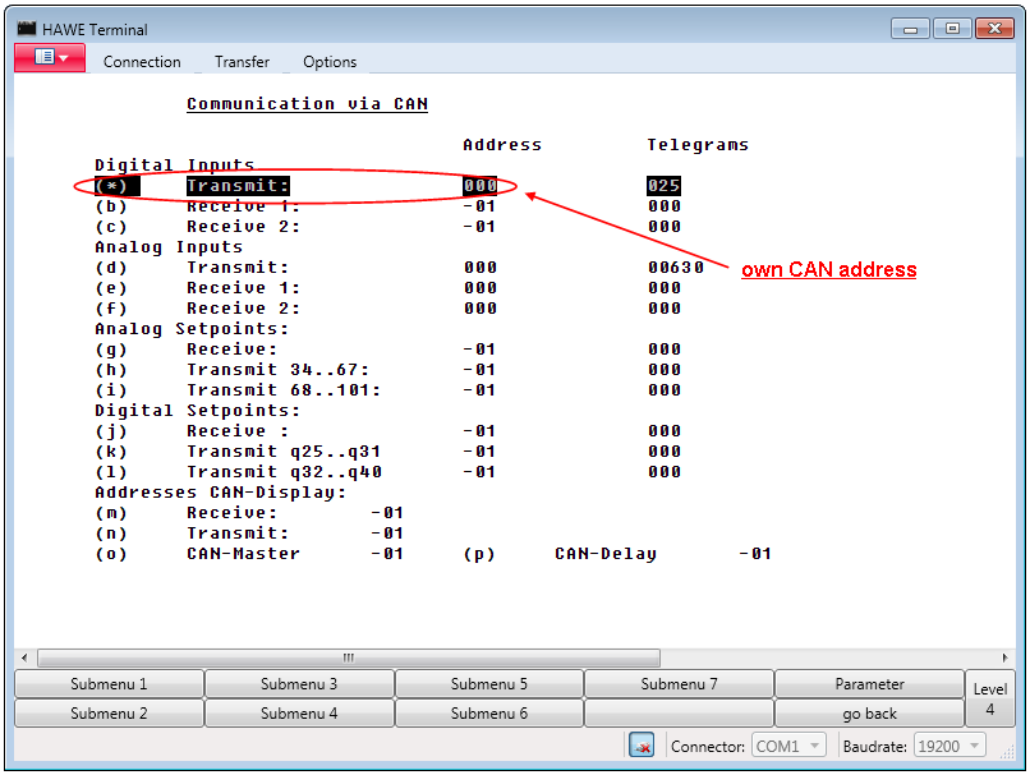


Figure 13.8.: Setting of the CAN Addresses

The registered value fixes the CAN address. -01 means that the PLVC isn't configured for the CAN communication, so no telegram can be sent.

The address value is possible to preset with 0..8. After the value has been adjusted, the whole system needs a restart for adopting the modification. The new CAN address is now visible in the terminal program (illustration 13.5).

The HMI CAN address is permanently set upon value 3 and can't be modified normally.

In case of a data collision on the bus, the telegrams, coming from the control with the lower CAN address, get priority. This means that more important conrols should get lower CAN addresses.

13.3.2 Baud Rate

The CAN bus can be operated with different speeds. The characteristic value is the baud rate which is displayed behind the CAN addresses in the terminal program (illustration 13.5).

The bus transmission speed depends primarily on the length of the bus connection (cable length). Further the quantity of the CAN telegrams and the necessary reaction rate should be regarded. The third is that not all CAN bus participants support every bus speed. Some of the devices are set permanently at just one baud rate or they are preconfigured according to customer demand. Other devices need special soft- and/or hardware for changing their setting.

In the mobile technology common baud rates are about 125 kbaud or 250 kbaud. Both of the rates are enough for the most applications. For a raised bus load choose 250 kbaud.

The baud rate for the PLVC can be justified comfortable with just one parameter.

The setting takes place in the menu:

Parameters → **Submenu 7: Special Parameters** → **(d) CAN-BAUD** (illustration 13.9)

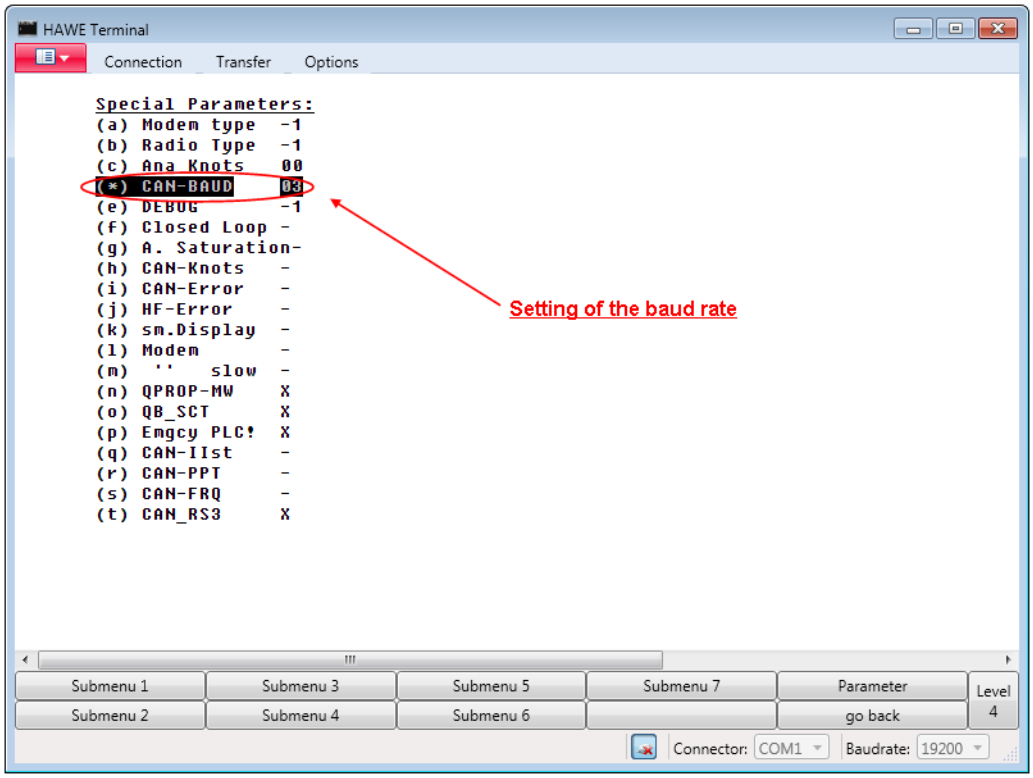


Figure 13.9.: Setting of the Baud Rate

In this place the baud rate will be set successive from 50 kbaud to 1000 kbaud. The numbers 0 until 5 stand for the common rates. 6 until 8 supply intermediate values. After the value has been adjusted, the whole system (all devices) needs a restart for adopting the modification. The new baud rate is now visible in the terminal program behind the CAN addresses.

The baud rate has to match for all participants!

13.4 Diagnosis

13.4.1 Diagnostic Menu in the Terminal Program

The terminal program contains all important information about the CAN bus.

After log into a PLVC the start screen displays the first information: CAN address and baud rate are readable in the last line.

More information is available at

Diagnosis → **Submenu 4: CAN, CAN-Nodes** (illustration 13.10)

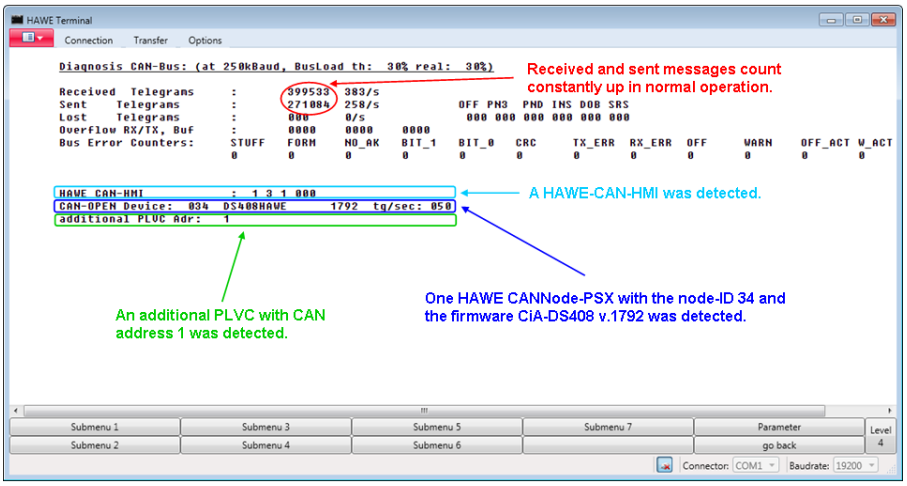


Figure 13.10.: Diagnostic Menu

At this place, beside transmitted and received telegrams, a whole series of bus errors is listed. Thereby the error causes can be very various. An error list with possible reasons can be found inside the appendix 13.A of this document.

The lower range of the screen displays devices, which are already identified from the PLVC. In the line **CAN node** you can find a list of identified, but namely unspecified control devices. Generally they are integrated as slave so their function is controlled by the functionality of the PLVC. In the lines beneath you can find further namely registered devices such as CAN receiver or the HMI.

13.4.2 CAN Bus Adaptor

With a CAN bus-adaptor the CAN bus data can be inspected on a notebook. This can be helpfull for localizing errors, for example programming errors (Are the requested telegrams even generated and do they contain the data they should?). During the connection between devices from different manufacturers double generated telegrams may occur. For example it is possible that a data designed telegram is used from another control unit as a control telegram. Some manufacturers provide incomplete or at least for untrained user difficult to understand protocol descriptions. Consequent obscurities can be solved with the CAN adaptor too.

Such an adaptor is recommended in case of periodic working with the CAN bus system or when participants, that come from different manufacturers, are integrated in the CAN bus.

HAWE already tested and recommend the CAN USB-adaptor of the company Peak-System. “Peak-System”.

13.4.3 CAN Bus Tester

CAN bus-tester localize physical errors in CAN bus-networks. Several are already available on the market. Those devices can help to find problems in difficult environments or larger CAN bus-networks

(many paricipants, large distances, higher baud rates). In smaller networks the technics shouldn't be necessary.

13.5 Data Telegrams

13.5.1 Telegram Contents

Any information on the CAN bus is summarized in telegrams. Each telegram includes up to 8 bytes. This is equal to four analog input signals from standard sensors (for example pressure transducer, rotary encoder,...) or the states of 64 digital inputs. Also mixing the data is allowed. For example byte 0 and byte 1 contain digital values and byte 2 and byte 3 contain together a numeric value. Each control system is able to access the telegrams and the included information via the bus.

13.5.2 Telegram Label

Apart from the data each telegram contains some control- and checkup bits and an identifier, which is used to indicate the telegram. This ID (ID=identifier) is a uniquely hexadecimal number with a length of two bytes, from hex 0 up to hex 77F (decimal: 0 up to 1919). The remaining bits are used for the sender's CAN address. This results in a maximum of 127 CAN participants.

13.5.3 Reserved Telegram-IDs

At the CANopen standard some of this IDs are reserved. So they can not used as normal data telegrams. For the periodic data transfer the range with the IDs hex 181 until hex 57F (decimal: 385 until 1407) is reserved. The PLVC uses some of this IDs for the communication among each other too. Depending on the configuration this can cause collisions. You must ensure that any ID is sent just from one control unit. If necessary change your setup.

13.6 Write and Read Data in ST-code

The PLVC offers several opportunities for accessing the CAN bus. With the ST-code (the application program) you can generate telegrams and write it on or read already existing telegrams from the CAN bus. For this tasks special function modules exist which are described in the documentation to OpenPCS resp. in the manufacturer documentation to the function modules.

13.6.1 Write a CAN Telegram

An example program for writing CAN telegrams with the ST-program:

```
(*Define variables*)
VAR
  schreibe_auf_can:          CAN_WRITE;
  bitwerte1:                 BYTE;
  bitwerte2:                 BYTE;
  zahlenwert1:               BYTE;
  zahlenwert2:               BYTE;
  schreibe_auf_can_integer:  CAN_WRITE_INT;
  zahlenwert3:               INT;
  zahlenwert4:               INT;
END_VAR

(* Programcode *)

(* Allocating the values to the particular bytes*)
schreibe_auf_can.B0 := bitwerte1;
schreibe_auf_can.B1 := bitwerte2;
schreibe_auf_can.B2 := zahlenwert1;
schreibe_auf_can.B3 := zahlenwert2;
(* Write CAN telegrams onto the bus *)
schreibe_auf_can(ID :=16#256,LENG :=4);

(* Alternative spelling *)

(* Allocating the values actually while calling the function*) schreibe_auf_can(ID
    :=16#256,
    LENG :=4,
    B0 :=bitwerte1,
    B1 :=bitwerte2,
    B2 :=zahlenwert1,
    B3 :=zahlenwert2
  );

(* It is better to write Integer values (numbers between +/- 32000) with the
    function
modules*)

schreibe_auf_can_integer.I0 := zahlenwert3;
schreibe_auf_can_integer.I1 := zahlenwert4;
schreibe_auf_can_integer(ID :=16#257,LENG :=4);
```

Between VAR and END_VAR the required variables and function modules are defined.

- Line 3 and line 8 define two functions for writing onto the CAN bus.

- Lines 4 t 7 define 8 bit (= 1 byte) variables, which can be fullfilled with particular bits or numeric values up to 255.
- Line 9 and line 10 define two integer variables, which are able to contain values between +/- 32000.

In the first part of the program (lines 15 to 21) a telegram is generated with CAN ID 256 and a telegram length of 4 bytes. This position declares also how many bytes have been used in the telegram. 1 to 8 bytes are possible.

- In line 16 until 20 particular values get allocation to the bytes.
- In line 21 the previously allocated values, the ID and the telegram length are committed to the function modules. At this point of the program the telegram will be written onto the bus.

In lines 23 to 32 the same function is executed, but in a different notation (in the program should stand the function of course just in some form **or** another). The values are dedicated to the function during the function call. The software engineer has the choice which notation he selects.

13.6.2 Read a CAN-Telegram

An example program for writing a CAN telegram with the ST-program:

```
(* Define variables *)
VAR
  can_lesen:       CAN_READ;
  can_initiealieren:  CAN_REC_INI;
    bittwerte1:      BYTE;
    bittwerte2:      BYTE;
  zahlenwert1:     INT;
  zahlenwert2:     INT;
END_VAR

(* Programcode *)

(* Just in the first program run *)
IF NOT erstdurchlauf THEN
  (* Apply the telegrams *)
  can_initialisieren(CHANNEL :=1,ID :=16#295);
  can_initialisieren(CHANNEL :=2,ID :=16#296);
  erstdurchlauf := TRUE;
END_IF

(* Read telegram *)
can_lesen(CHANNEL :=1);
(* Request the validation *)
if can_lesen.valid =1 then
  (* Transform and transfer valid values into BYTE *) bitwerte1 :=
```

```
26    dint_to_byte(usint_to_dint(can_lesen.b0)); bitwerte2 :=
        dint_to_byte(usint_to_dint(can_lesen.b1));
  END_IF;
28
  (* Read telegram *)
30 can_lesen(CHANNEL :=2);
  (* Request the validation *)
32 if can_lesen.valid =1 then
    (* Transform and transfer valid values into INTEGER *)
34  zahlenwert1 := usint_to_int (can_lesen.b0);
    zahlenwert1 := zahlenwert1 + usint_to_int (can_lesen.b1)*256;
36  zahlenwert2 := usint_to_int (can_lesen.b2);
    zahlenwert2 := zahlenwert2 + usint_to_int (can_lesen.b3)*256;
38 END_IF;
```

The reading of telegrams is splitted into two steps.

Within the first step (lines 13 to 19) telegrams will be initialized. Only the preselected telegrams are able to be read afterwards. Therefore the function module can_rec_ini is in use, which can read a maximum of 10 telegrams. In the first part of the program you get a channel number (1-10) and the telegram-ID. This function has to be called just once, hence it is set in an if-check.

Afterwards (from line 21) the telegrams will be read in and the values will be attributed to the variables periodically (with every program run).

The assignment here was also set in an IF query. By using the value **valid** the user can find out, if the telegram is imported correctly and if the values are valid. In the example the old values stay in the variables as long as valid isn't TRUE. The assignment here was also set in an IF query The data inside the telegrams must convert separate into the format you need in the program. The function **dint_to_byte** resp. **usint_to_int** execute the converting. If a two byte value (numerical values +/- 32000) shall be imported, it has to split up into two seperate values with each one byte (line 35 and 36).

13.7 Data Linking via Communication Parameter

Attention:

With firmware later 2012, GET_ANA functionality has increased. All CAN Telegrams from 281_{hex} to $2F8_{hex}$ and 181_{hex} to $1F8_{hex}$ can be read now. For more onformation read chapter 12.6.4.
If the following funktionality is used, take care that CAN telegrams are only be generated by this funktion. Otherwise it could overlap with the automatic read-in of PDOs.

In order to interchange separate values between different devices via CAN bus the functions of the previous chapters are enough. But for mailing and importing bigger data volume it is exhausting and costs a lot of memory capacity and machine time.

Additional CAN bus functions permit the integration of several PLVCs into a single system and also to read in bigger data volume. Such a functionality is able to extend the in- and output level of a single PLVC by one or two further PLVCs. Therefore you need communication parameters, which choose predefined telegram ranges to substitute in- and output values.

The advantages of this functions are:

- less lines of code
- less time for writing and reading
- integer values doesn't need a transformation
- illustration of the values in the terminal program

13.7.1 Transmit Digital Inputs via CAN

The first step for transmitting digital inputs via CAN already starts at the CAN address allocation. Each PLVC writes a telegram with its own digital input values continously onto the CAN bus. This telegram is identified by the CAN ID (the identifier) hex 181 + CAN address. Therefore the PLVC with the CAN address 0 transmits its digital input values with the CAN ID hex 181. A PLVC with the CAN address 4 transmits its digital input values with the ID 181 + 4 = hex 185. Thereby the status of the physical inputs IB0.0 until IB3.7 is written into the first four data byte of the telegram.

(PLVC with CAN address 0, telegram-ID 181h)

Byte	Bit	Digital input
byte 0	0	IB0.0
0	1	IB0.1
0	2	IB0.2
0	3	IB0.3
0	4	IB0.4
0	5	IB0.5
0	6	IB0.6
0	7	IB0.7

Continued on the next page. . .

. . . Continued from previous Page

Byte 1	0	IB1.0
1	1	IB1.1
1	2	IB1.2
1	3	IB...
and so on		

Table 13.3.: Allocation of Digital Input Bits

For a PLVC with a different CAN address only the telegram-ID changes. The allocation of the data bits remains continual.

13.7.2 Transmit Analog Inputs via CAN

The way to write the analog input values “automatically” onto the CAN bus is the same as with the digital inputs. In this case the PLVC generates a whole series of telegrams.

Which IDs are used for which analog input value is obvious in the terminal program (illustration 13.11):

Parameters → Submenu 6: Enabling CAN-Analog Telegrams

If there are no IDs, the PLVC is not parameterized for sending analog input signals. Otherwise a telegram-ID has to stand in every line. In this view you can see which values will be transmitted with the telegram-IDs. A summary is available in the appendix 13.A of this document.

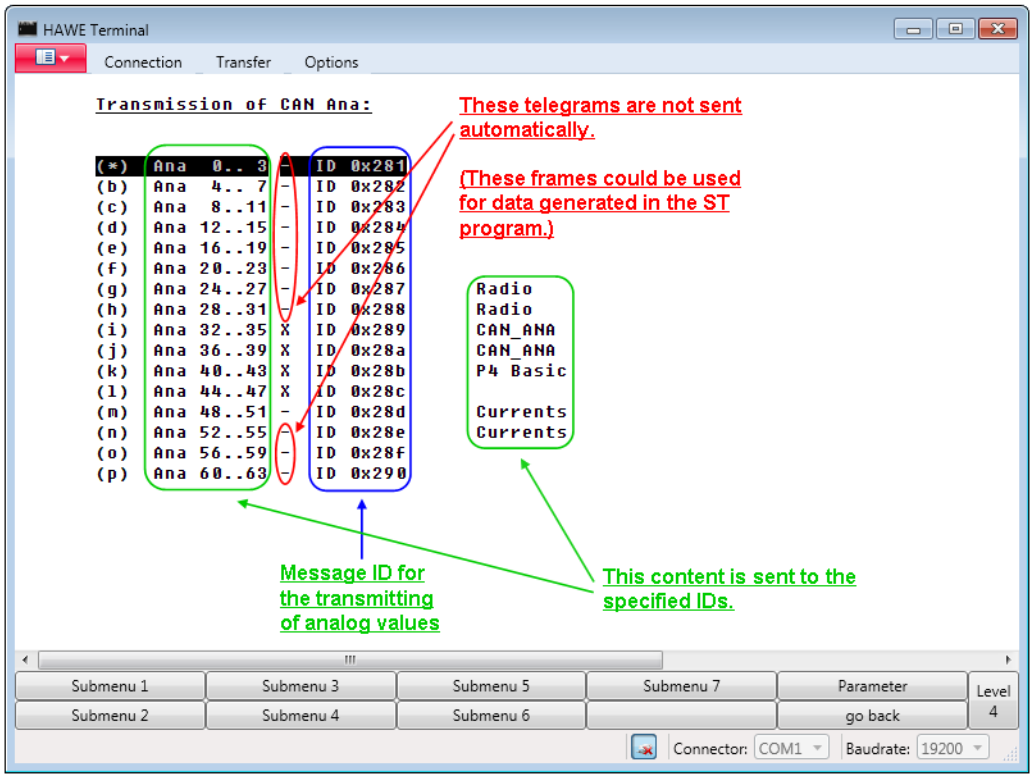


Figure 13.11.: Transmit Analog Values

Probably not all of the listed analog values are required. Depending on the hardware a lot of analog inputs do not exist at the PLVC. In this menu you can turn off particular telegrams by choosing the regarding item and add a minus. The consequence is a smaller bus load and free IDs. If necessary they can be send manually out of the ST-program with new values.

In the terminal program you can fix which telegram-ID shall be used for which analog value:

> **Parameters** → **Submenu 4: Communication** → **Analog Inputs (d) Transmit** (illustration 13.12)

Depending on the at this place predefined number (0 to 3) a different part of IDs is chosen for the telegrams.

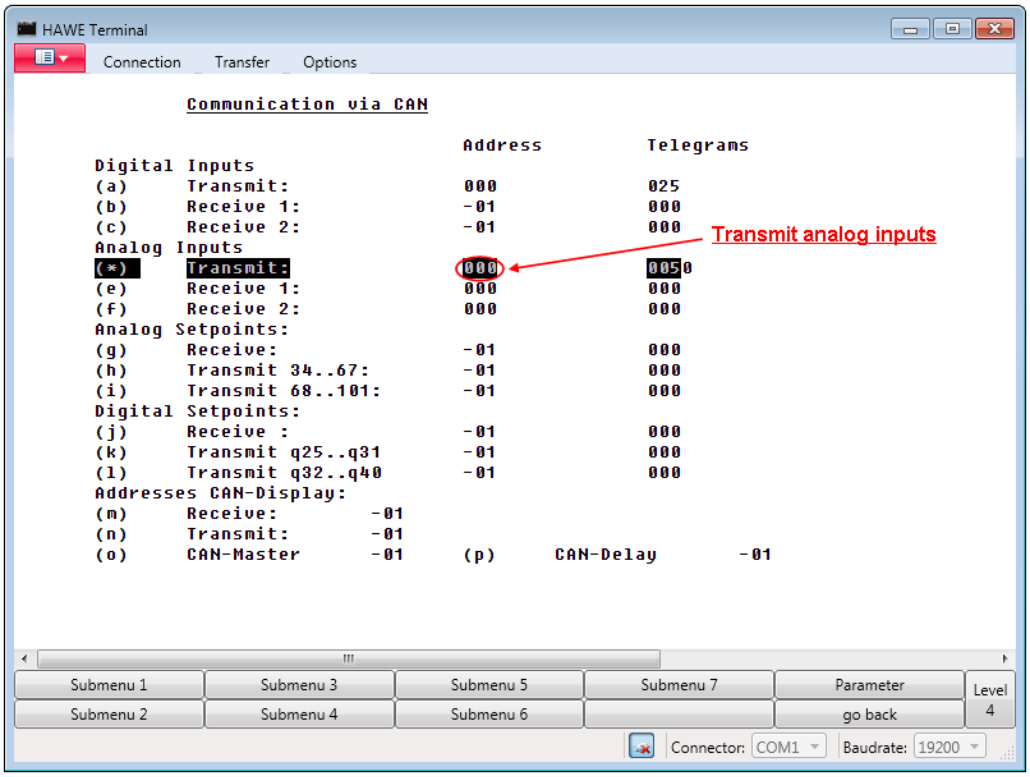


Figure 13.12.: Transmit Analog Values

13.7.3 Read Digital Inputs via CAN

In order to read digital input values of a neighboring PLVC, it is enough to import the standard telegram (hex 181 + CAN address) of the corresponding PLVC. This can be set comfortabel with the **communication parameter b** and **c**:

> **Parameters → Submenu 4: Communication → Digital Inputs (b) Receive 1 / (c) Receive 2**

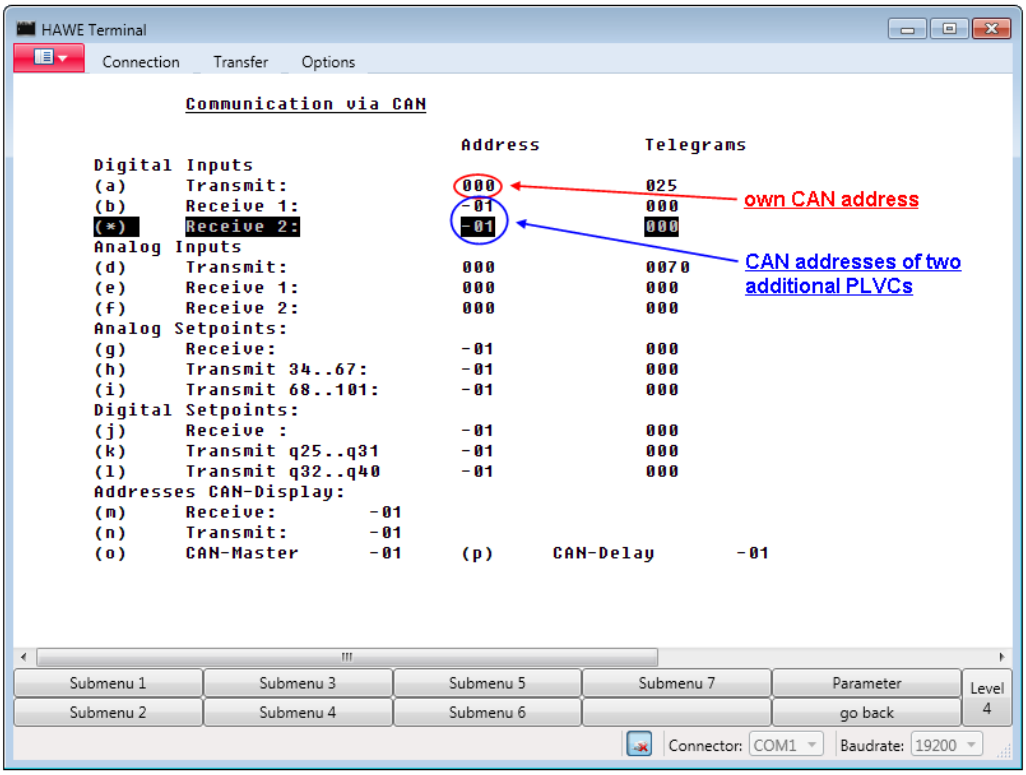


Figure 13.13.: Receive Digital Inputs

Enter in this part the CAN addresses of the PLVCs, which inputs shall be read (illustration 13.13). The corresponding telegrams will be imported and bit by bit analysed. Single values can be accessed in the ST-program via IB-addresses, a special programming for transforming or analysing the values is not necessary.

After that the digital inputs appear as IB8.0 to IB11.7 for the first registered PLVC and the second PLVC inputs appear as IB16.0 to IB19.7.

Both entries must be placed in ascending order! A PLVC with CAN address 0 meaningfully reads in b = 1 and c = 2 and a PLVC with CAN address 1 reads in b = 0 and c = 2 (instead of b = 2 and c = 0). If b and c aren't filled in sequence (no ascending order), the values possibly may be inconsistent. This is caused by the calculation of the automatic integration of the input values at CAN HMI and a further PLVC.

A HAWE-HMI at the bus doesn't need extra adjustment via the communication parameter. The digital inputs are automatically integrated as IB12.0 to IB15.7.

If both parameters (b and c) have an assigned PLVC, a further PLVC (if available) with CAN address 4 is also read in automatically and integrated as 4.PLVC extern.

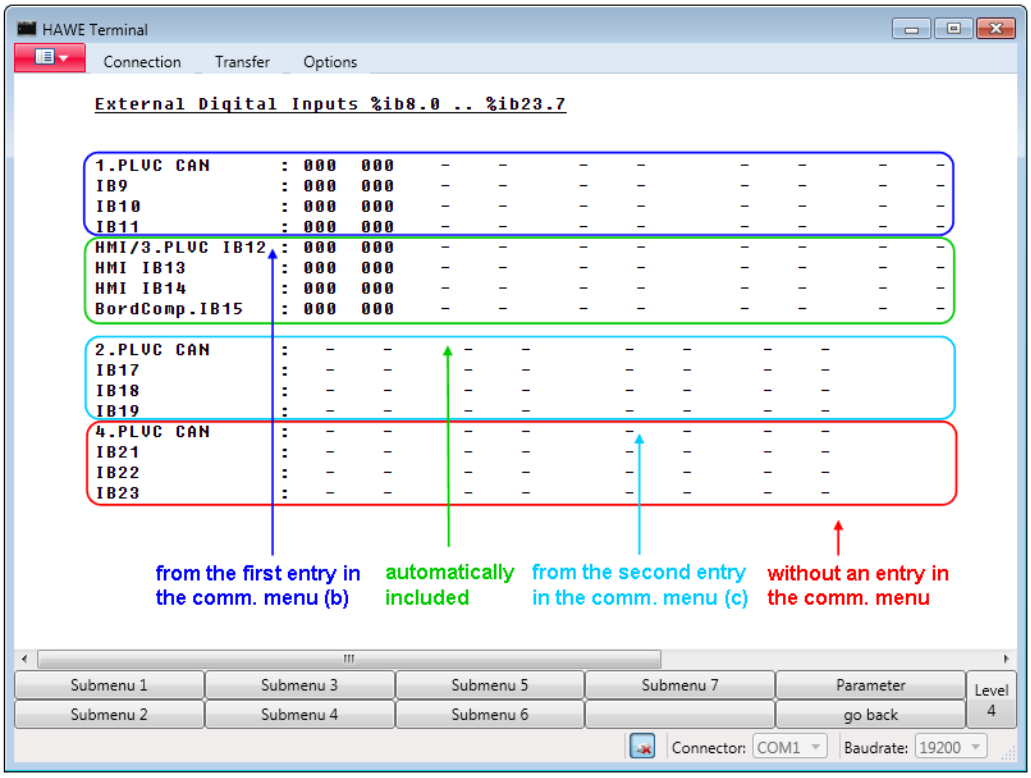


Figure 13.14.: Digital Inputs From External PLVC and HMI

13.7.4 Read Analog Inputs via CAN

The way to read in analog input values via CAN is the same as with the digital inputs. With registrate CAN addresses of other PLVCs in the parameter menu

> **Parameters → Submenu 4: Communication → Analog Inputs (e) Receive 1 / (f) Reveive 2**

the corresponding telegrams are appraised and made available as analog values. Accordingly they are visible in the terminal program at **Analog Inputs → Submenu 7** (illustration 13.15)

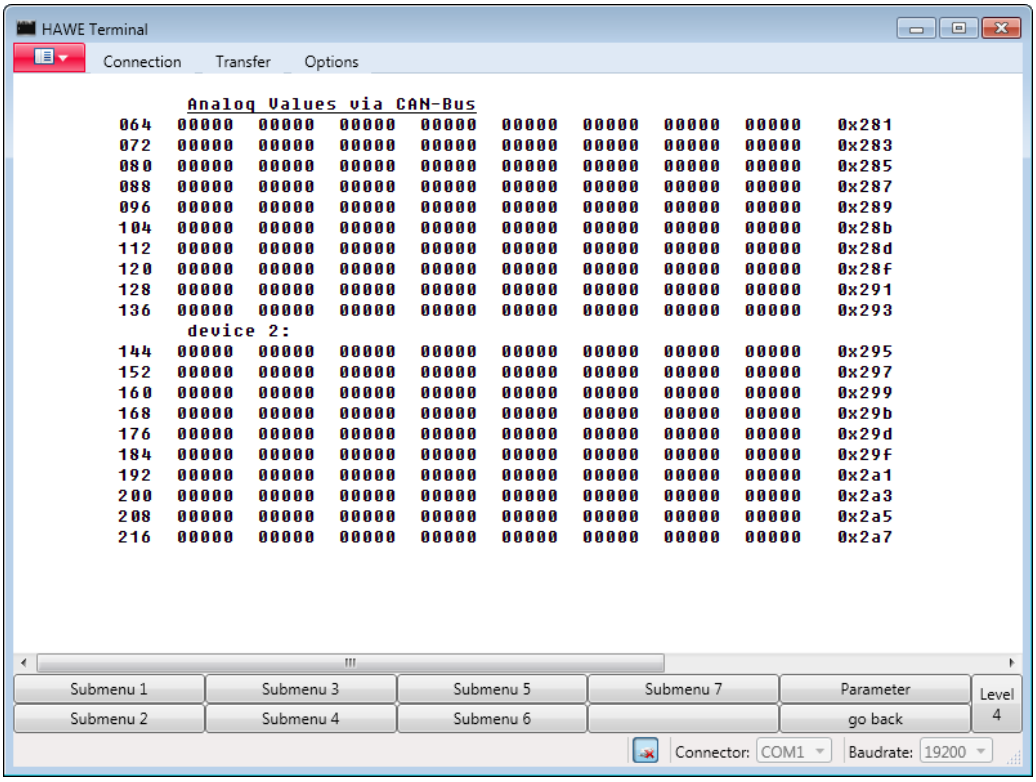


Figure 13.15.: Analog Inputs From External PLVC

Thereby the telegrams are disassembled into four integer values. A summary of the allocation is available in the appendix 13.A of this document. For using these values in the ST-program, they must be read in with the function module GET_ANA.

For example:

```
(* Define variables *)
VAR
  schreibe_auf_can:          CAN_WRITE;
  bitwerte1:                 BYTE;
  bitwerte2:                 BYTE;
  zahlenwert1:               BYTE;
  zahlenwert2:               BYTE;
  schreibe_auf_can_integer:  CAN_WRITE_INT;
  zahlenwert3:               INT;
  zahlenwert4:               INT;
END_VAR

(* Program code *)

(* Allocate the particular bytes the appropriate values *) schreibe_auf_can.B0
:= bitwerte1; schreibe_auf_can.B1 := bitwerte2;
schreibe_auf_can.B2 := zahlenwert1;
schreibe_auf_can.B3 := zahlenwert2;
(* Write the telegram onto the bus *)
```

```
20 schreibe_auf_can(ID :=16#680,LENG :=4);

22 (* Alternative spelling *)

24 (* The values will be allocated the function during the call *) schreibe_auf_can(ID
       :=16#680,
       LENG :=4,
26     B0 :=bitwerte1,
       B1 :=bitwerte2,
28     B2 :=zahlenwert1,
       B3 :=zahlenwert2
30   );

32 (* Integer values (numeric values from +/- 32000) better be written with
   the integer function modul *)
34
   schreibe_auf_can_integer.I0 := zahlenwert3;
36 schreibe_auf_can_integer.I1 := zahlenwert4;
   schreibe_auf_can_integer(ID :=16#681,LENG :=4);
```

The advantages of this function are:

- less programming effort
- the input saves relevant computing power, in comparison to the input with CAN_REC_INI and CAN_READ
- the values don't need a transformation, they are immediately available as integer

This functionality can be used not just for transmitting analog input values. Also selfmade telegrams (e.g. computation results) can be read in comfortably.

Please note: These values will be set automatically to -1, if a timeout arrives. If this poses a problem in your application, it must be intercepted in the program.

13.7.5 Send and Receive Output Values via CAN

The setpoints of the digital and analog outputs can be sent and received just the same as the input values.

The operation corresponds to that of the entrance level. Each PLVC is able to receive and for two control units also send such setpoints. Thereby a telegram is created (i.e. read in) for every digital and analog setpoint.

Digital Outputs

In order to head digital outputs of a further PLVC, select them in the ST-program like local entrances. The addressing (QB. . .) for the first external PLVC will be postponed thereby to 25 and for the second PLVC to 33 counters.

The table 13.4 gives an overview about the local outputs with the allocation of the same outputs on the external PLVC.

local	1. external	2. external
QB0.0	QB25.0	QB33.0
QB0.1	QB25.1	QB33.1
QB0.2	QB25.2	QB33.2
QB0.3	QB25.3	QB33.3
QB0.4	QB...	...
QB1.0	QB26.0	QB34.0
QB2.0	QB27.0	QB35.0
QB3.0	QB28.0	QB36.0
QB4.0	QB29.0	QB37.0
QB5.0	QB30.0	QB38.0
QB6.0	QB31.0	QB39.0
QB7.0	QB32.0	QB40.0
QB7.1	QB32.1	...
and so on		

Table 13.4.: Allocation of the Digital Outputs

For the sending PLVC, one and/or two CAN addresses must be entered in the terminal program at

> **Parameters → Submenu 4: Communication Digital Setpoints (k) Transmit q25..q31 / (l) Transmit q32..q40**

Please note that at this place the parameter “k” is “filled” with QB25. . . and the parameter “l” with QB33. . . .

In reverse the PLVC, which shall be remote controlled, needs the same number at the back:

Parameters → Submenu 4: Communication Digital Setpoints (j) Receive

This number has to agree with the number, that was registered in the transmitting PLVC at k and/or l, depending on which values should control the local outputs.

Analog Outputs (Proportional Outputs)

The selection of the local PWM- and IPWM-outputs is made by the function module ACT_VALVE in the ST-program. This function module gets a “CHANNEL”, “SET POINT” and an “OVERRIDE”. “CHANNEL” designates the exit.

A higher channel number is denoted for taking remote control about such analog outputs.

local	1. extern	2. extern
Prop0 until Prop32	34 until 66	68 until 100

Table 13.5.: Addressing of External Analog Outputs

To determine which values are sent with which telegram-ID, the parameters must be set at this position too.

The reading or remote controlled PLVC needs the same value in the corresponding communication parameter for receiving the analog setpoints.

13.8 CAN-Open

13.8.1 Introduction CAN Open

Latest Firmware for PLVC41 / PLVC82 now supports features given in PLVC.EDS i.e. it can also be used as CANOPEN slave.

Free “Slave-to-Slave” Interconnection between several PLVC/CANIO can be realized via \Parameter\submenu4 Parameters a) to l). (see chapter 13.3)

13.8.2 Master mode

PLVC up to now, normally has been used as CANOPEN Master (mainly starting CANopen Slaves, and reading Radio/Joystick). This is enabled by Setting Parameter CAN-Master to 1, i.e. setting first bit (illustration 13.16).

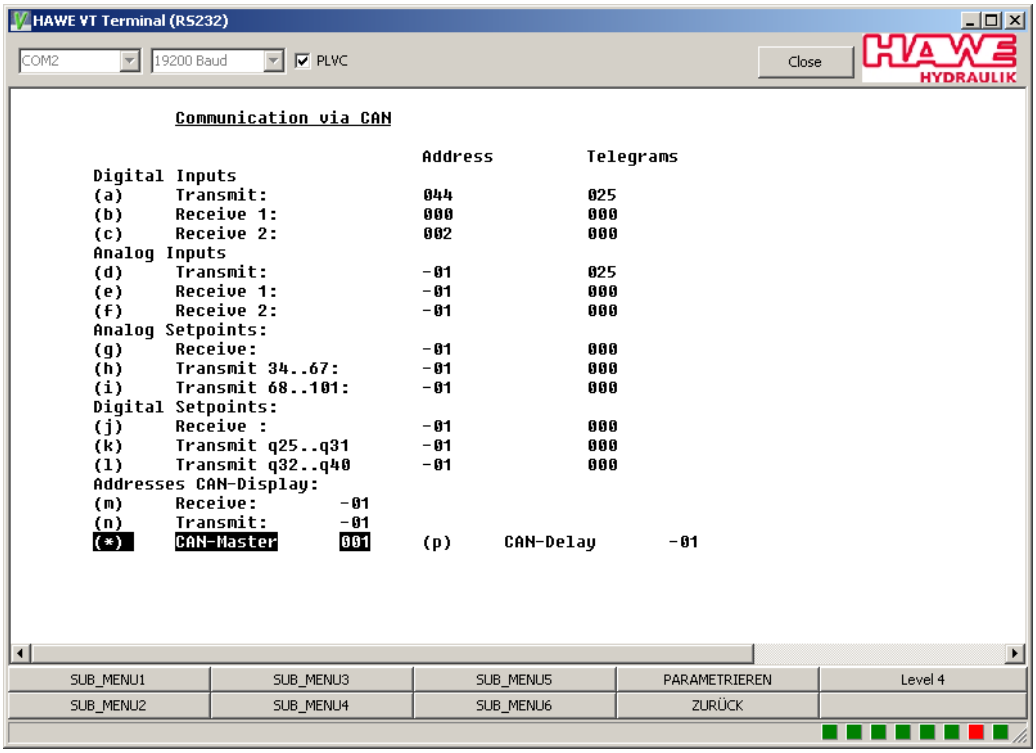


Figure 13.16.: Communication CAN Master 1

For HMI-Slaves with dig. Outputs, 4 has to be added->set to 5.

To bring device to mode operational, a startup telegram has to be generated with Telegram ID 0, b0=1, b1=0 (for all, or node-id for special device).

If bit 1 of Parameter CAN_Master is set PLVC will generate this telegram

a) at boot-up

b) on reception of reception of Telegram 700_{hex}+x (boot-up of slave)

c) on first reception of telegram 181_{hex} + x (PDO1)

After this telegram, slaves normally send cyclically their value (position/angle/joystick), either on 181_{hex}+ID or 281_{hex}+ID (PDO1/2).

Some slave are in SYNC-mode as default, so that CAN-Master has to send a SYNC telegram continuously (Telegram ID 80_{hex}, no data). Then CAN-Master has to be set accordingly.

Therefore parameter o) must be increased by one oft he following values:

- By Adding 8 to the value, SYNC telegram is generated every 10ms.
- By Adding 16 to the value, SYNC telegram is generated every 20ms.
- By Adding 24 to the value, SYNC telegram is generated every 40ms.

The PDO values from slave can be seen in AnalogInputs/submenu 8, 9, A, B, C and D and can be read as integer via GET_ANA in OpenPCS (Chapter 12.6.4)

13.8.3 Slave Mode

For Slave-mode, negative Values have to be set.

-1 is unvalid

-2 enables SDO support (CAN_OPEN, acc. To plvc.eds-file)

-3 = -1 + -2

-4 enables Heartbeat

-6 = -2 + -4

-8 enables reaction to SYNC (normally cyclic sending is used)

So normally -6 has to be set, minus node-ID (With Node-ID = 1, -7 must be entered).

This Parameters can also be set via LSS (CAN-Open, with Serial number known).

Different from PLVC-style version, NODE-ID is defined by parameter a) +1 only. -1 deactivates, 0 -> Node-ID 1 -> PDO 181_{hex}, 281_{hex}, 381_{hex}, 481_{hex}, 201_{hex}, 301_{hex}, 401_{hex} are supported. So Node-Id I would look like this:

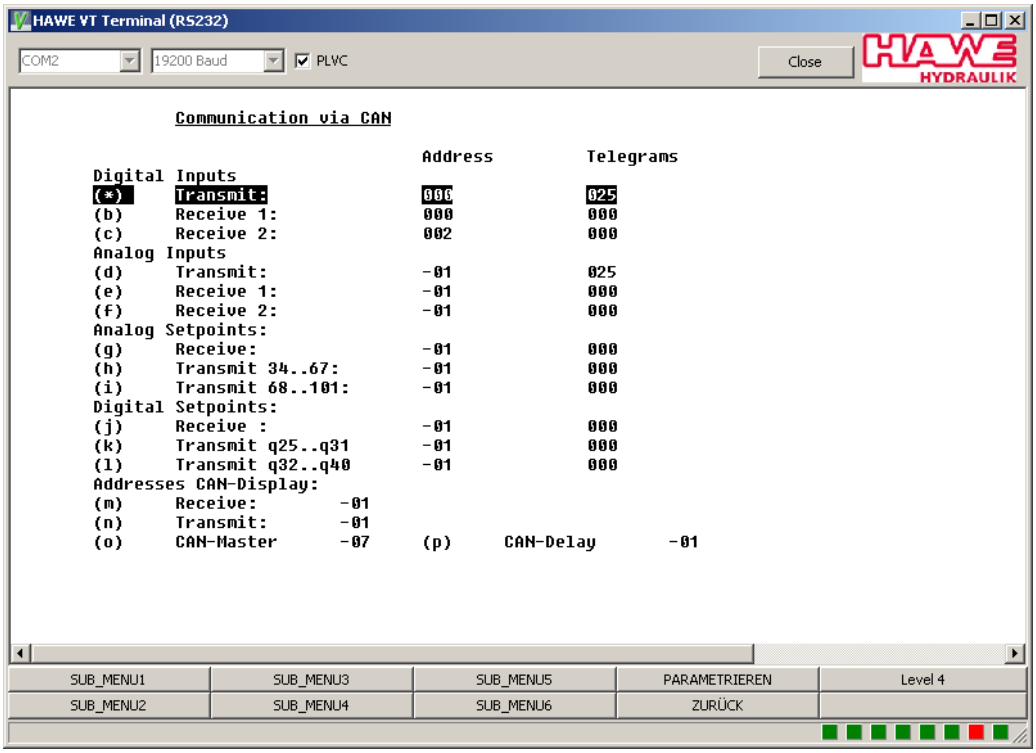


Figure 13.17.: Slave Mode - CAN Slave ID 1

At reset, only one 701_{hex} Heartbeat-Telegram is sent. A Startup-Telegram with ID=0, b0=1, b1=0 is needed to start. After Startup, the following PDOs can be seen.

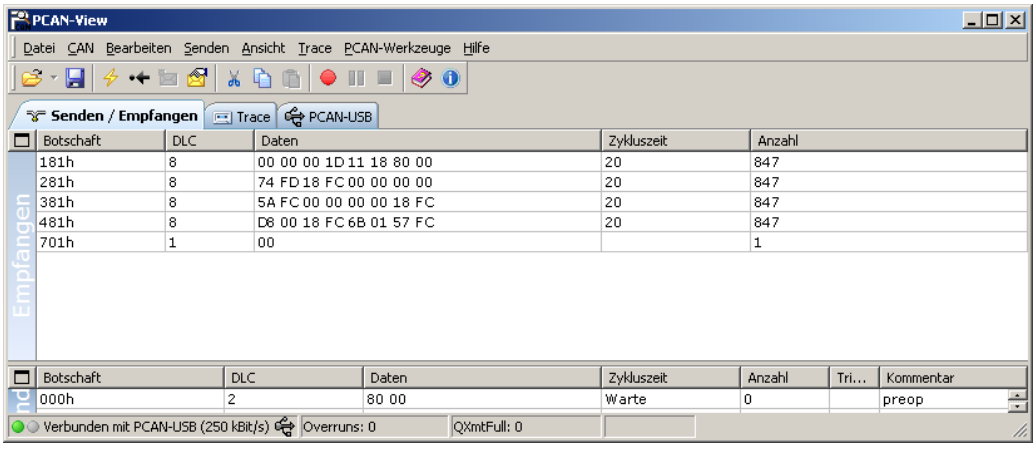


Figure 13.18.: Slave Mode - Heartbeat

Parameter\submenu6 is used to dis/en-able PDOs 281_{hex} to 481_{hex}

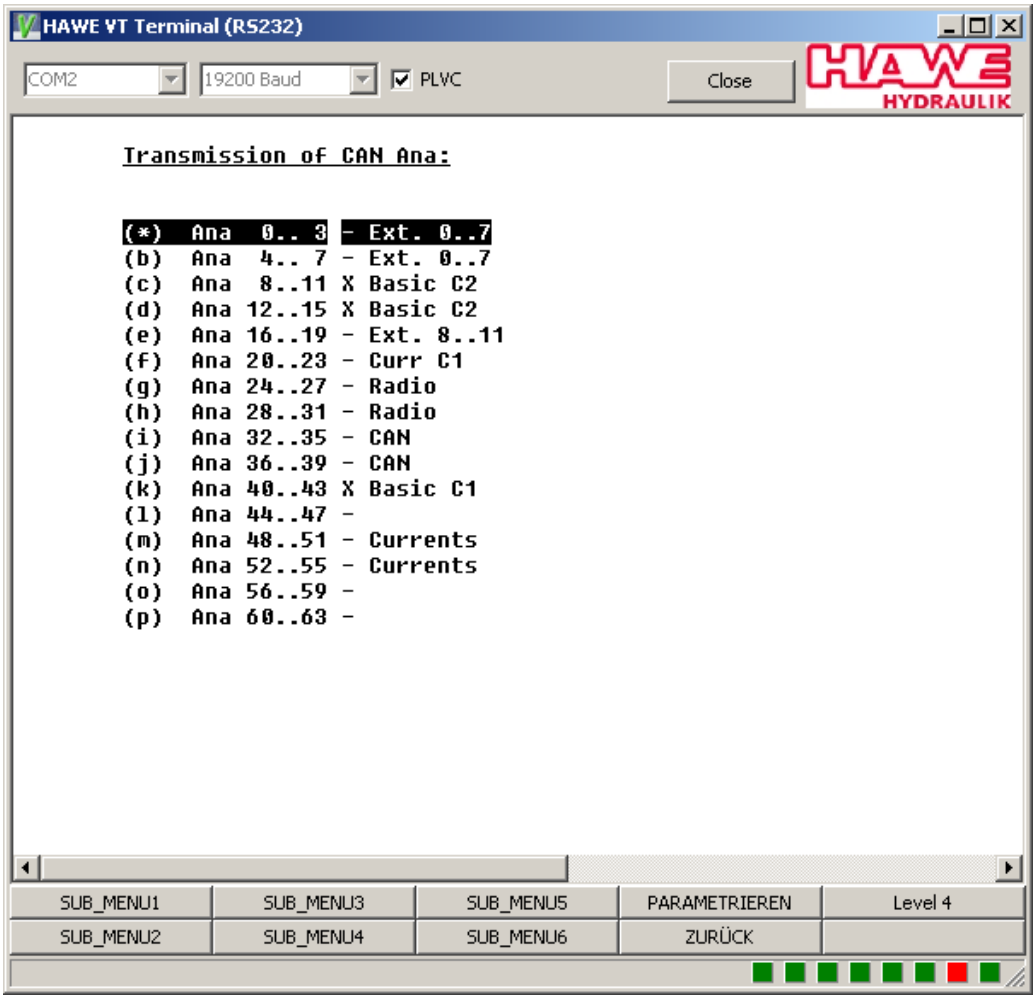


Figure 13.19.: PDO Select

This example aktivates the following

Parameter c) 281_{hex} is enabled and sends inputs 8..11.

Parameter d) 381_{hex} is enabled and sends inputs 12..15.

Parameter k) 481_{hex} is enabled and sends inputs 40..43.

m) and n) can be enabled instead of c) and d) to get measured current on the bus

Screenshot of eds-file: Inputs.

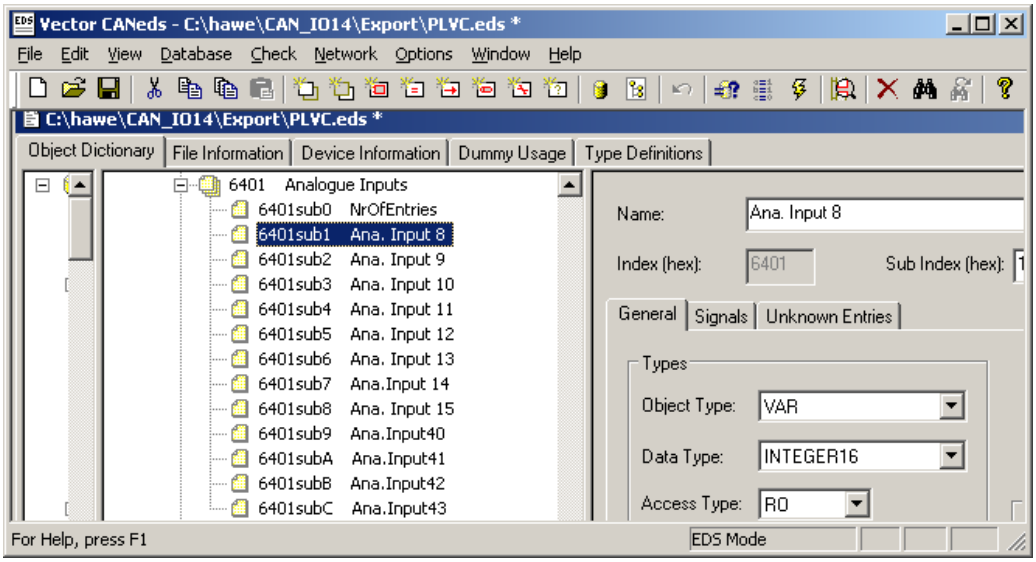


Figure 13.20.: EDS Screen of Inputs

Cycle time can be modified via SDO, by setting event-timer of respective Object.

Writing outputs of PLVC:

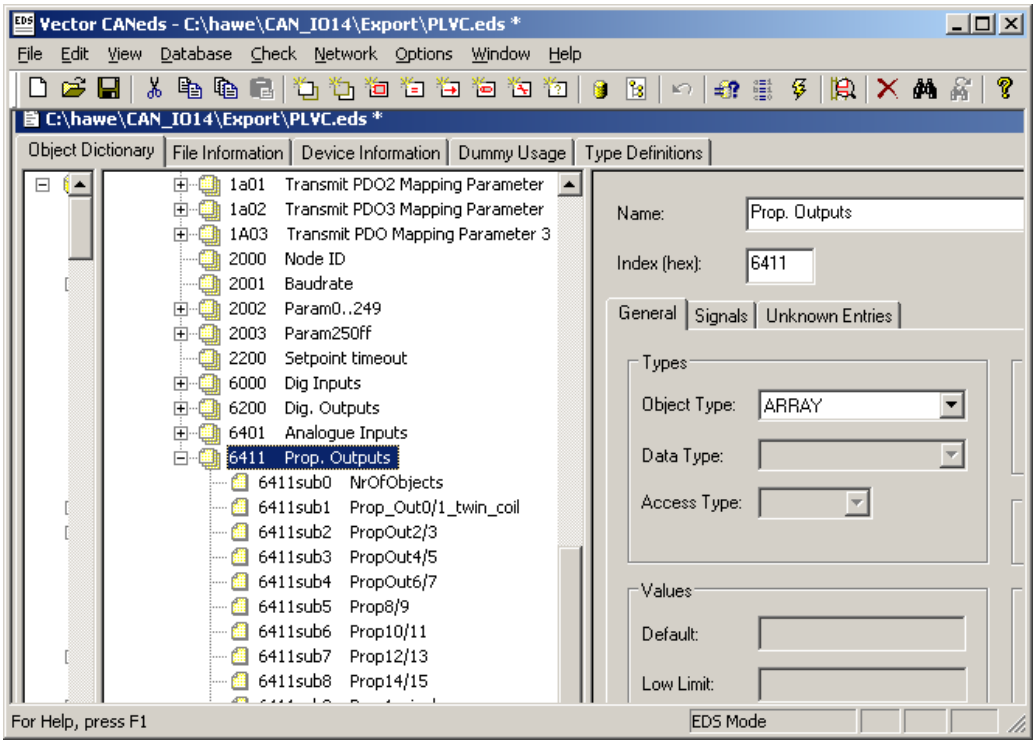


Figure 13.21.: EDS Screen of Outputs

Via telegram 201_{hex}, the digital outputs are set from qb0.0 to qb6.7 Via telegram 301_{hex}, 4 prop. Outputs are set (Twin-coils on 0,2,4,6) Via telegram 401_{hex}, 4 prop. Outputs are set (Twin-coils on 8,10,12,14)

13.9 Appendix

13.9.1 Digital Inputs

The CAN address specifies which telegram ID is used for sending the digital input values.

Parameters → Submenu 4: Communication → Digital Inputs (a) Transmit

specifies the CAN address and sends the digital input values via the telegram ID:

CAN address	**telegram-ID**	commentary
0	181	
1	182	
2	183	
3	184	used by HMI
4	185	
5	186	
6	187	
7	188	
8	189	

Table 13.6.: Assignment of the CAN Address to the Telegram-ID

The values of the input addresses will be sent in **packets of four data bytes**. Which telegrams are read in and integrated as an input is specified by:

Parameters → Submenu 4: Communication → Digital Inputs (b) Receive 1/ (c) Receive 2

The entries must take place in **ascending** order:

line	**item**	or	**line**	**item**	or	**line**	**item**
(b) receive	0		(b) receive	1		(b) receive	0
(c) receive	1		(c) receive	2		(c) receive	2

Continued on the next page. . .

. . . Continued from previous Page

Table 13.7.: Regard the Correct Order

In this telegram the values of the inputs IB0.0 to IB3.7 will be send.

The receiving PLVC integrates them as IB8.0 to IB11.7 (for the first entry) and as IB16.0 to IB19.7 (for the second entry):

read the **parameter entry** of the digital input	**byte number** inside the telegram	**transmitting PLVC**	**receiving PLVC**
(b)	0	IB0.0 to IB0.7	IB8.0 to IB8.7
	1	IB1	IB9
	2	IB2	IB10
	3	IB3	IB11
(c)	0	IB0	IB16
	1	IB1	IB17
	2	IB2	IB18
	3	IB3	IB19

Table 13.8.: Assignment of the Receiving Parameter to the IB-Address

13.9.2 Analog Inputs

Depending on the hardware configuration the inputs of the PLVC are different. But they generally pertain the following allocations:

Parameters → Submenu 4: Communication → Analog Inputs (d) Transmit

The value which has to be registered here (0 to 3) doesn't have to match with the CAN address in a). . . . But it doesn't make sense to switch at will between the values yet.

According to the value a series of sequenced telegram-IDs are selected for writing the analog values onto the CAN bus.

parameter send analog inputs	**telegram-IDs**
0	281 bis 290
1	295 to 2A4
2	2A9 to 2B8
3	2BD to 2CC

Table 13.9.: Assignment of the Parameter to the Telegram-IDs

Via the parameters

> **Parameters → Submenu 4: Communication → Analog Inputs (e) Reveice 1 / (f) Receive 2**

two groups of telegrams can be integrated as integer values.

The addresses correspond to the telegrams in accordance with table 13.9. In detail the following allocations result.

The “number” designates, which value was entered in the parameters e and/or f. According to that, the IDs standing below will be integrated. With each ID 4 integer values will be received.

Depending on the place where the ID group number was registered (e or f), the original analog input values (0 to 63) can be read in as analog input 64 to 127 (aim 1) or 144 to 207 (aim 2). In addition the analog input number has to be delivered as channel to the function module GET_ANA.

ID at number 0	**ID** at number 1	**ID** at number 2	**ID** at number 3	**source**	**aim 1**	**aim 2**
281	295	2A9	2BD	0 to 3	64 to 67	144 to 147
282	296	2AA	2BE	4 to 7	68..	148..
283	297	2AB	2BF	8 to 11	72..	152..
284	298	2AC	2C0	12 to 15	76..	156..
285	299	2AD	2C1	16 to 19	80..	160..
286	29A	2AE	2C2	20 to 23	84..	164..
287	29B	2AF	2C3	24 to 27	88..	168..

Continued on the next page. . .

. . . Continued from previous Page

288	29C	2B0	2C4	28 to 31	92..	172..
289	29D	2B1	2C5	32 to 35	96..	176..
28A	29E	2B2	2C6	36 to 39	100..	180..
28B	29F	2B3	2C7	40 to 43	104..	184..
28C	2A0	2B4	2C8	44 to 47	108..	188..
28D	2A1	2B5	2C9	48 to 51	112..	192..
28E	2A2	2B6	2CA	52 to 55	116..	196..
28F	2A3	2B7	2CB	56 to 59	120..	200..
290	2A4	2B8	2CC	60 to 63	124 to 127	204 to 207

Table 13.10.: Allocation of the Analog Values, ID and Aim

The analog input numbers can be taken from the terminal diagrams. So the analog inputs 40 to 34 stand for the analog inputs of the base device PLVC41-G.

The analog inputs 24 to 31 contain the analog values of a CAN receiver, which is recognized of the PLVC.

And the analog inputs 48 to 63 contain the values of the current measurement of the current controlled outputs via a shunt resistor back to the appropriated inputs.

13.9.3 J1939

In the engine management, the CAN bus works with a bit different protocol. It is called J1939. A essential difference is the usage of an extended identifier. For this protocol explicit rules have been defined. How the individual manufacturer implements these rules must be taken from the documents about the engine electronic.

Via the function modules for the ST-program, it is also possible to access these data or send telegrams to an engine electronic. To send CAN telegrams, the function module CAN_WRITE_29 is used, which permits the input of the extended identifier.

Most data is expected by the engine electronic in relatively short cycles. If a telegram is sent within the ST-program, it is quite possible that these times are exceeded. The way to prevent, is to repeat the function call in the program code several times. So it is expedient to write the function call and the data allocation separately. Only the function call should be placed at a further place (or more) in the program. The telegram is always sent at the particular place of the program call.

13.9.4 Structure of a CAN Message

Each message is parceled in an own CAN bus mode. This packing is called “frame”.

Each frame consist of 7 characteristic diagrams:

- Start-condition
- Message identifier
- Control bits
- Data (0-8 bytes)
- Parity bits
- Acknowledge-bit
- Stop-condition

In addition the frames can be distinguished by the length of the identifier:

- Standard frame (11 bit identifier)
- Extended frame (29 bit identifier)

After the frametype a possible distinction type is

- Data frame (the data will be send without special invitation)
- Remote data frame (data will be requested - a receiver identifies the REMOTE and transmits his message afterwards)

Standard-frame according to standard CAN 2.0A:

	start	identifier	RTR	IDE	r0	DLC	data	CRC	ACK	EOF+IFS
	1 bit	11 bit	1 bit	1 bit	1 bit	4 bit	0..8 byte	15 bit	2 bit	10 bit

start	identifier	SRR	IDE	identifier	RTR	r1	r0	DLC	data	CRC	ACK	EOF+IFS
1 bit	11 bit	1 bit	1 bit	18 bit	1 bit	1 bit	1 bit	4 bit	0..8 byte	15 bit	2 bit	10 bit

start: Dominant, relevant to synchronisation

identifier: Information to the recipient and priority information to the bus arbitration

RTR: Recessive, differentiate between data- (dominant) and data request telegram (recessive)

IDE: Identifier extension

r0: Reserved

DLC: Contains the length information of the following data

DATA: Contains the proper telegram data

CRC: Marks the error code for the preceding information. The CRC check sum is used for error detection

ACK: Feedback from other participants in the case of correct reception

EOF: Marks the end of the data telegram (7 recessive bits)

IFS: Marks the time period for the transfer of a correct received message

SRR: Replaces in the extendedf frame the RTR of the standard frame

IDE: Shows that 18 further bits will follow

r1, r0: Reserved bits

DLC: Length information of the following data

Bitstuffing

At the bit level the coding of the single bits is checked. The CAN protocol uses the NRZ coding (non-return-to zero), which ensures a maximum efficiency during the bit coding. The synchronisation flanks will be produced according to the bitstuffing method. A transmitter inserts a stuff bit with complementary value into the bit stream after five following equivalent bits, which the receiver removes automatically.

If one or several failures are discovered by at least one node with the help of the mechanisms described on top, the running transmission is cancelled by sending an “Error flag”. The effect is that the acception of the transferred message is prevented at other stations and therefore the network wide data consistency is guaranteed. After stopping the transfer the transmitter starts automatically to send his message again.

Effective Transmission Rate for Data Bytes

Despite automatic access of a CAN node to the bus lead it is possible to specify reference values of the effective transmission rate for a node of the top priority. A standard formatted message with eight data bytes needs maximum 130 bits. Thereby, it can assumed that a maximum number of 19 stuff bits and 3 space bits are required:

	1	start bit
+	11	identifier bits
+	1	RTR bit
+	6	control bits
+	64	data bits
+	15	CRC bits
+	19	(maximum) stuff bits
+	1	CRC delimiter bit
+	1	ACK slot bit
+	1	ACK delimiter bit
+	7	EOF bits
+	3	IFS (Inter Frame Space) bits
=	130	bits

Example

- Transmission rate 250k => per second 250 000 bit transmission means 4μs per bit
- An address with 8 bytes of data requires max. 130 bits according to the listing above.
- 130 x 4 μs = 520 μs = 0,52ms
- 18 addresses amount: 18 x 0,52ms = 9,36ms

Example

- If the CAN IDs contain 7 data byte, the data traffic is reduced to
- 130 bit - 8bit (1 byte) - 2 stuff bit (estimate) = 120 bit
- 120 * 4 μs = 480 μs
- 18 addresses amount: 480 x 18 = 8,64ms

Example

- If the CAN IDs contains even 4 data byte the data traffic is reduced to
- 130 bit - 32bit (8 byte) - 5 stuff bit (estimate) = 93 bit
- 98 x 4 μs = 372 μs
- 18 addresses amount: 372 μs x 18 = 6,7ms

13.9.5 Valve Nodes as PSL-CAN for PLVC Control Modules

To configure PSL-CAN, a functionblock is given through CAN_VALVE. The functionality of that functionblock is desribed in chapter CAN_VALVE(12.6.3).

Plug&Play functionality expects merely the following requirement for the address assignment: The external and via CAN bus selected valves must be placed on CAN node-IDs from 32, all other data traffic and the belonging observation and security functions are made by the PLVC.

The function block ACT_VALVE(12.6.1) is used for control.

Single valves are addressed with consecutive indexes starting from 2000.

The indices of the double valves are calculated from $2000 + 2 \cdot n$, where n is the number of the section. The combination of the IDs is shown in table 13.11.

Section number.	Node-ID	COB-ID of setpoint	Actual COB-ID
1	32	0x220	0x1A0
2	34	0x222	0x1A2
3	36	0x224	0x1A4
4	38	0x226	0x1A6
5	40	0x228	0x1A8
6	42	0x22A	0x1AA
7	44	0x22C	0x1AC
8	46	0x22E	0x1AE
9	48	0x230	0x1B0
10	50	0x232	0x1B2

Table 13.11.: Node-IDs in the Process

Each connected CAN node receives the required setpoint message with control word on receive PDO1. The CANopen standard addressing is essential.

The CAN bus master of the PLVC should be actived. This is achieved by setting the parameter 0 or -1 in the communication menu (**Parameters** → **Submenu 4: Communication**) to 1 as shown in figure 13.22.

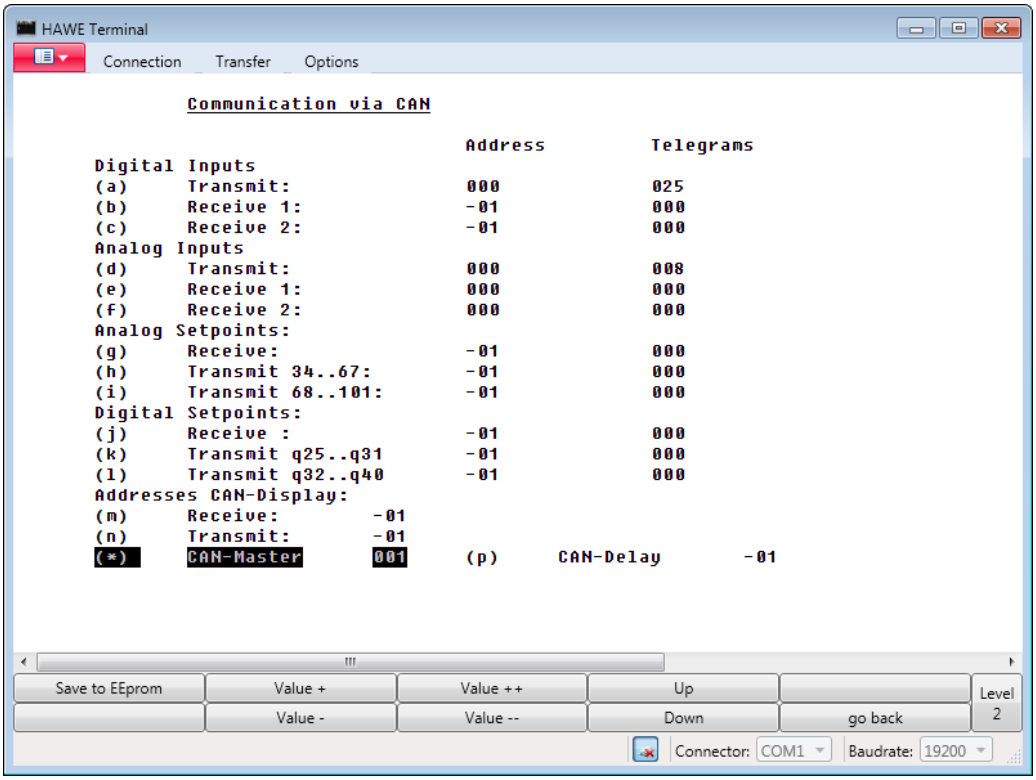


Figure 13.22.: Activation of the PLVC41 CAN Masters

The CAN baud rate must be set identical for all participants (**Parameters** → **Submenu 7: Special Parameters**).

In menu **Prop. Valves** shown in figure 13.23 (**Prop. Valves** → **Submenu 6: CAN-Valves**), the function of the CAN nodes can be monitored. Here the setpoints, actual values and error messages are shown.

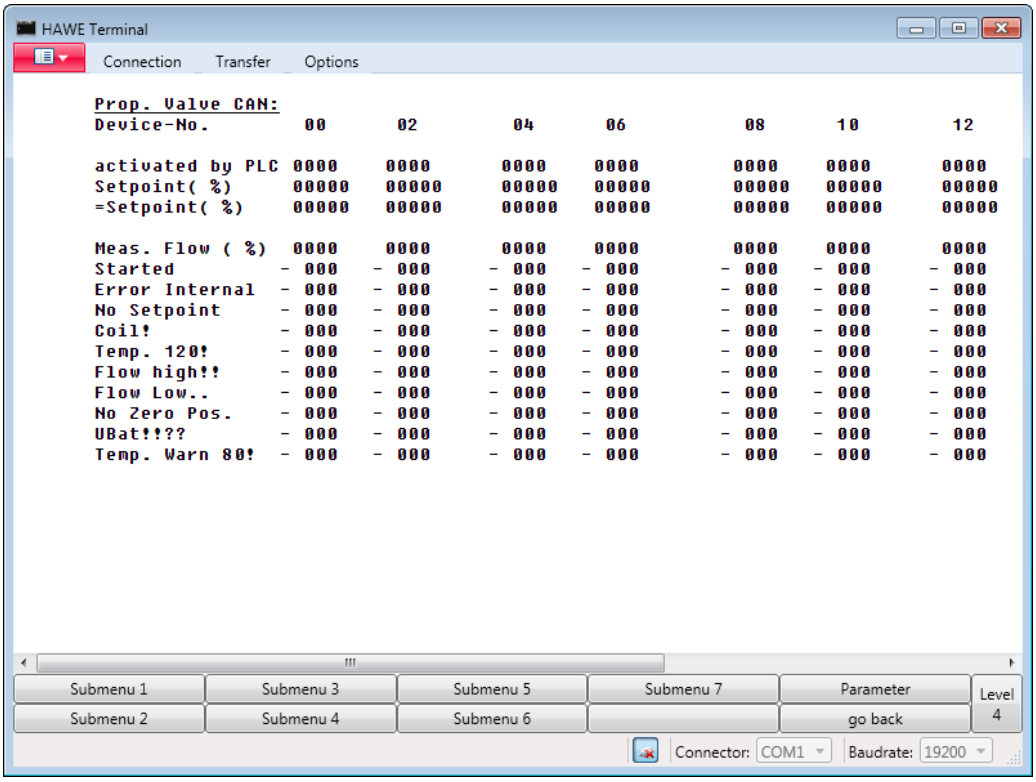


Figure 13.23.: Overview of the CAN Nodes

After committed setpoint message, the PLVC monitores the actual values of the CAN node on timeout (about 200ms). When the CAN node has received a setpoint message, it monitors this setpoint message on timeout (configurable).

Part V.

Tips and Tricks

14 Tips and Tricks

14.1 CAN position transducer according to DS 406

Such position transducer can be used simplified. Therefore the NodeID has to be set right according to the LMT transcript. (Example realized with a Balluff transition transducer):

7e5h 04 01 → release of LMT data
7e5h 24 01 → output of producer-name; answer: 7e4h 24 42 41 4C 4C 55 46 46 → Balluff
7e5h 11 40 → change NodeID to 40; Antwort : 7e4h 11 err err , err err has to be 00 00, otherwise error
7e5 17 00 → save setting to standard; answer: 7e4h 17 err err , err err has to be 00 00, otherwise error

Afterwards the valves for position and speed can be read out in the OpenPCS:

```
pos :POS_READ;
poscan1 :DINT;
pos1 :INT;
speedcan1 :DINT;
pos(CHANNEL :=68);
poscan1 := (pos.pos1 - 22648) / 100;
pos1:= DINT_TO_INT(poscan1);
speedcan1 := pos.pos2;
```

14.2 Loss of application after reboot

If the user-parameter number 99 is 4711, the OpenPCS recognizes no ressource on the PLVC after a reboot and suggests to download it new. It seems like a memory loss on the PLVC (in the terminalprogramm the number of SPS-runs would be 0).

Measure:
Set user-parameter to a value differing to 4711.

14.3 Defect OpenPCS programm produces "infinite loop"

A defect OpenPCS programm produces an error, which causes a reboot. Immideately after the reboot this error appears again. There is no way to interrupt with the terminal by setting parameter 99 = 4711 and stop the programm.

With PLVC VT there is a way to set the parameter directly via Connection→Terminal-old.

14.4 10V-Output

The 10V-output is triggered like a prop valve output:

```
prop: ACT_VALVE;
sollwert: INT;
prop(CHANNEL:=32, SETPOINT:=sollwert, OVERRIDE:=1000);
```

The output has to be parameterized by analog input 46!

This output besides can get a ramp (ramp to analog input 46 = ramp 62).

14.5 Synchronization Control

For the hydraulic synchrinization control, backed up with the valve-SPS PLVC you have to be aware of following facts: Position transducer to measure the length of the cylinders: You can use analog as well as digital position transducers. Digital position transducers contain an integrated electronic with CAN-Bus system.
Using the analog option you have to keep in mind that the included ADC (analog digital converter) which makes a parameter you can deal with out of the analog signal, has a resolution of 10 bit.
That means, that the signal is captured with 1000 steps maximum. Part of this 1000 steps drop out because of a lower area being used for the recognition ob a broken cable and position transducer capturing a bigger area.
The last bit is possibly a bit noisy so that all in all circa 500 steps remain for a secure position detection.
By comparing two or more position transducers it is attemped to achieve synchronism, meaning two values with around 500 steps resolution are compared with.
Relating this to the overall length you get the best precision you can achieve. Because of a certain dynamic of the system and the fact that you may not accomplish a 100% correction due to the deviation of one bit, it seems logic that there has to be some kind of hysteresis range ($\pm$1Bit), where you don't adjust as well as a range where you react at first slow and then faster and faster. The experience allows us to say that the position transducer has to be at least 10 times better than the required

maximum deviation, or the other way round: with 10 bit you can control/position with the accurancy of 1% at its best.
A further difficulty is the hysteresis of the valves, which can always lead to a dead time and thereby to further errors. That can be partly compensated by Hall sensors i.e. the dead time gets reduced.
Moreover the operating capacity should be choosen in a way, where both valves use their full range. Especially extremly slow moving in the range is problematic.
To be able to reach a better precision for example at longer distances you have to invest in a sensor system with better resolution. That's why draw wire sensors by ASM or position transducers with absolute encoder mostly have 16 bit, i.e. 32000 steps resolution.
The requiered movingspeed or rather the overall time to cover the overall distance is very important. If you work e.g. intern with 20msec resolution an arrangement which covers the whole distance in 1sec (i.e. 50cycles) can't be extremly precise. That means: The smaller the moving speed, the higher the precision.

14.6 CAN-Adress per GET_EE

With the function block GET_EE the own CAN-Bus-adress can be read out:

```
VAR
      para: get_ee;
      can_adresse: INT;
END_VAR
para( CHANNEL := 580|can_adresse := ee_val);
```

This only works with newer operating systems (since 2008)!

14.7 Use MW or MB

To buffer values you can use QB or QW adresses you don't use otherwise. In this way e.g. more digital values (BOOL) can be combined in bytes and treated common (Example: create CAN-message).

Instead of QB (or rather QW for 2 byte) you can basically access 16MB or rather MW.

Then it's called:

```
statusbyte AT %MB4.0: BYTE;
```

in place of:

```
statusbyte AT %QB4.0: BYTE;
```

and the correspondiong bits:

```
status1 AT %MB4.0: BOOL;
status2 AT %MB4.1: BOOL;
...
```

in place of:

```
status1 AT %QB4.0: BOOL;
status2 AT %QB4.1: BOOL;
...
```

There are 16 memory MB available.

14.8 Free QB

Depending on the configuration as well as the need to access digital outputs of other PLVCs.

14.9 Save variables of the type DINT in EEPROM

As is well known only variables of the type INT can be saved in EEPROM. Following code shows how to convert a DINT-value into two INT-values and how to generate again one DINT-value out of those two INT-values.

```
VAR
  var1: DINT; (* Input variable *)
  var2: INT;  (* First variable for EE_SAVE *)
  var3: INT;  (* Second variable for EE_SAVE *)
  var4: DINT; (* Output variable *)
END_VAR
var1:=1234567890;
var2:=dint_to_int(var1/65535);
var3:=dint_to_int(var1-(int_to_dint(var2)*65535));
var4:=int_to_dint(var3)+(int_to_dint(var2)*65535);
```

14.10 Set single bits specifically

There are several ways to set single bits of one byte for example to connect a row of outputs (QB...) or generate status messages. One Way is for example the logic combination of integer-values which match with the values of the bit. The first bit of a byte has the value 1, the second 2, the third 4, the

fourth 8. In some cases it would be nicer to adress the bits with their bitnumber or any number. That makes the programm code more coherent and more clear. Here an array can help.

Example:

```
VAR
fenster_nummer:      INT;
fenster_bit:         ARRAY[0..8] OF BYTE :=[0,1,2,4,8,16,32,64,128];
fenster AT %MB0.0:   BYTE;
END_VAR
fenster := fenster_bit[fenster_nummer];
```

In the array 9 values have been defined and numbered from 0 to 8. Those values have attached values in the second square brackets. If fenster_nummer includes the value 4, then this value is read out of the array, which is attached to this counter. Here it is the 8. The value 8 is equivalent to the fourth bit of a byte. Therefore the fourth bit in the programming line is set in the variable "fenster" This programm only sets one bit in the byte "fenster". If an existing value should be kept and another bit additional set, the programming linesays:

```
fenster := fenster OR
fenster_bit[fenster_nummer];
```

In this example virtual outputs are set by the byte "fenster".

14.11 Maximum number QB..

There are maximum 128QB.. available.

14.12 Maximum number of byte on Profibus

For the Profibus there are maximum around 100 byte available. Possibly there are more. This can be detectet only by tests.

14.13 Move two cylinders parallel with one joystick

As needed one of the two cylinders can be moved faster or slower.

variables declaration:

```
VAR
joy_x AT %IW104.0: INT;
diff AT %IW106.0: INT; (* y-Achse des Joysticks *)
prop: ACT_VALVE;
setp_x: INT;
setp_y: INT;
END_VAR
```

code:

```
setp_x:=joy_x-diff;
setp_y:=joy_x+diff;
prop(CHANNEL :=0,SETPOINT :=setp_x,OVERRIDE :=1000);
(* First twincoil*)
prop(CHANNEL :=2,SETPOINT :=setp_y,OVERRIDE :=1000);
(* Second twincoil *)
```

14.14 Save parameter per OpenPCS (Shift+S)

variables declaration:

```
eeprom: put_par;
```

code: (run when you want to save)

```
eeprom( CHANNEL :=32000, PARA :=1 );
eeprom( CHANNEL :=32000, PARA :=2 );
```

Part VI.

Troubleshooting

The following table lists error conditions and tips on troubleshooting:

Error condition	Reason	Solution
Control does not start (LEDs do not light)	No power supply	Control the power supply and the fuses
	Download of the operating system is incomplete	Download the operating system again
	Cable break in the supply line	Replace cable
No login possible	Control is turned off	Turn on control
	Serial interface is not connected or regular twisted	Connect serial interface properly
	Download of the operating system is incomplete	Download the operating system again
Program is not running	Program stopped via user parameters	User parameter 99 is not allowed to have the value 4711
	Program transfer not succesful	Having logged in via the terminal program, the program name must be seen on the front page
Input signal (digital/analog) is not recognized	Wire not connected	Connect wire
	No signal on the wire	Check signal level with a multimeter
Valve output without function	Wire not connected	Connect wire
	Output is not driven	Control activation via terminal program/visual tool (error message OPN=open)
CAN communication interrupted	Incorrect baud rate	Control and, where necessary, switch baud rate. All controls must be set to the same baud rate.
	Interference from other cables	Use shielded cable. Do not run power cables close to them.

Continued on the next page . . .

. . . continued from previous page

Table 14.1.: Errors and Ways to Eliminate

Part VII.

Attachment

Suggestions for improvement

Suggestions for improvement referring to: **PLVC Manual**

Ideas to improve this manual:

Mistakes in this manual:

Sent by:

Name: _____

Company Name: _____

Adress: _____

Please send to:

HAWE Hydraulik SE
Einsteinring 17
85609 Aschheim / Munich
Germany
email: techsupport@hawe.de

Disclaimer

Information in this document is provided solely in connection with HAWE products. HAWE Hydraulik SE and its subsidiaries ("HAWE") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All HAWE products are sold pursuant to HAWE's terms and conditions of sale.
Purchasers are solely responsible for the choice, selection and use of the HAWE products and services described herein, and HAWE assumes no liability whatsoever relating to the choice, selection or use of the HAWE products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by HAWE for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN HAWE'S TERMS AND CONDITIONS OF SALE HAWE DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF HAWE PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZE REPRESENTATIVE OF HAWE, HAWE PRODUCTS ARE NOT DESIGNED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS, WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE.

Information in this document supersedes and replaces all information previously supplied.

HAWE Hydraulik SE

Einsteinring 17 | 85609 Aschheim/München | Postfach 11 55 | 85605 Aschheim/München | Germany

Tel +49 89 379100-1000 | Fax +49 89 379100-91000 | info@hawe.de | www.hawe.com

Release: 2.0.6-0-g5b0081f (2019-08-12)