# *System Manual*
# *PLCcore-9G20*

## User Manual
Version 1.0

**Edition July 2010**

Document No.: L-1254e_1

SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz
Phone: +49 (3661) 6279-0 Fax: +49 (3661) 6279-99
Web: http://www.systec-electronic.com Mail: info@systec-electronic.com

# Status/Changes

Status: released

| Date/Version | Section | Changes | Editor |
|---|---|---|---|
| 2010/07/15<br>1.0 | All | Creation | R. Sieber |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

This manual includes descriptions for copyrighted products that are not explicitly indicated as such. The absence of the trademark (©) symbol does not infer that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, SYS TEC electronic GmbH assumes no responsibility for any inaccuracies. SYS TEC electronic GmbH neither guarantees nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. SYS TEC electronic GmbH reserves the right to alter the information contained herein without prior notification and does not accept responsibility for any damages which might result.

Additionally, SYS TEC electronic GmbH neither guarantees nor assumes any liability for damages arising from the improper usage or improper installation of the hardware or software. SYS TEC electronic GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.



Inform yourselves:

| **Contact** | **Direct** | **Your local distributor** |
|---|---|---|
| Address: | SYS TEC electronic GmbH<br>August-Bebel-Str. 29<br>D-07973 Greiz<br>GERMANY | Please find a list of our distributors under:<br><br>http://www.systec-electronic.com/distributors |
| Ordering Information: | +49 (0) 36 61 / 62 79-0<br>info@systec-electronic.com | |
| Technical Support: | +49 (0) 36 61 / 62 79-0<br>support@systec-electronic.com | |
| Fax: | +49 (0) 36 61 / 6 79 99 | |
| Web Site: | http://www.systec-electronic.com | |

1st Edition July 2010

# Table of Contents

# 1 Introduction

Thank you that you have decided for the SYS TEC PLCcore-9G20. This product provides to you an innovative and high-capacity PLC-kernel. Due to its high performance on a small manufactured size and due to its low power consumption, it is well-suitable as communication and control processor for embedded applications.

Please take some time to read through this manual carefully. It contains important information about the commissioning, configuration and programming of the PLCcore-9G20. It will assist you in getting familiar with the functional range and usage of the PLCcore-9G20. This document is complemented by other manuals, e.g. for the *OpenPCS* IEC 61131 programming system and the CANopen extension for IEC 61131-3. Table 3 in section 4.1 shows a listing of relevant manuals for the PLCcore-9G20. Please also refer to those complementary documents.

For more information, optional products, updates et cetera, we recommend you to visit our website: http://www.systec-electronic.com. The content of this website is updated periodically and provides to you downloads of the latest software releases and manual versions.

## Declaration of Electro Magnetic Conformity for PLCcore-9G20 (EMC law)

The PLCcore-9G20 has been designed to be used as vendor part for the integration into devices (further industrial processing) or as Development Board for laboratory development (hard- and software development).

After the integration into a device or when changes/extensions are made to this product, the conformity to EMC-law again must be assessed and certified. Only thereafter products may be launched onto the market.

The CE-conformity is only valid for the application area described in this document and only under compliance with the following commissioning instructions! The PLCcore-9G20 is ESD-sensitive and may only be unpacked, used and operated by trained personal at ESD-conform work stations.

The PLCcore-9G20 is a module for the application in automation technology. It features IEC 61131-3 programmability, uses standard CAN-bus and Ethernet network interfaces and a standardized network protocol. Consequently, development times are short and hardware costs are reasonable. PLC-functionality is created on-board through a CANopen network layer. Hence, it is not necessary for the user to create firmware.

# 2 Overview / Where to find what?

The PLCcore-9G20 is based on SYS TEC ECUcore-9G20 hardware and is extended by PLC-specific functionality (PLD software, PLC firmware). There are different hardware manuals for all hardware components such as the ECUcore-9G20 and the PLCcore-9G20 (the hardware of both modules is identical), development boards and reference circuitry. Software-sided, the PLCcore-9G20 is programmed with IEC 61131-3-conform *OpenPCS* programming environment. There are additional manuals for *OpenPCS* that describe the handling of programming tools and SYS TEC-specific extensions. Those are part of the software package *"OpenPCS"*. Table 1 lists up all relevant manuals for the PLCcore-9G20.

*Table 1: Overview of relevant manuals for the PLCcore-9G20*

| **Information about…** | **In which manual?** |
|---|---|
| Basic information about the PLCcore-9G20 (configuration, administration, process image, connection assignment, firmware update, reference designs et cetera) | In this manual |
| Development of user-specific C/C++ applications for the ECUcore-9G20 / PLCcore-9G20, VMware-Image of the Linux development system | System Manual ECUcore-9G20 (Manual no.: L-1253) |
| Hardware description about the ECUcore-9G20 / PLCcore-9G20, reference designs et cetera | Hardware Manual ECUcore-9G20 (Manual no.: L-1255) |
| Development Board for the ECUcore-9G20 / PLCcore-9G20, reference designs et cetera | Hardware Manual Development Board 9G20 (Manual no.: L-1256) |
| Driver Development Kit (DDK) for the ECUcore-9G20 | Software Manual Driver Development Kit (DDK) for ECUcore-9G20 (Manual no.: L-1257) |
| Basics about the *OpenPCS* IEC 61131 programming system | Brief instructions for the programming system (Entry "*OpenPCS Documentation*" in the *OpenPCS* program group of the start menu) (Manual no.: L-1005) |
| Complete description of the *OpenPCS* IEC 61131 programming system, basics about the PLC programming according to IEC 61131-3 | Online help about the *OpenPCS* programming system |
| Command overview and description of standard function blocks according to IEC 61131-3 | Online help about the *OpenPCS* programming system |
| SYS TEC extension for IEC 61131-3:<br>- String functions<br>- UDP function blocks<br>- SIO function blocks<br>- FB for RTC, Counter, EEPROM, PWM/PTO | User Manual *"SYS TEC-specific extensions for OpenPCS / IEC 61131-3"* (Manual no.: L-1054) |

| | |
|---|---|
| CANopen extension for IEC 61131-3<br>(Network variables, CANopen function blocks) | User Manual *"CANopen extension for IEC 61131-3"*<br>(Manual no.: L-1008) |
| Textbook about PLC programming according to IEC 61131-3 | IEC 61131-3: Programming Industrial Automation Systems<br>John/Tiegelkamp<br>Springer-Verlag<br>ISBN: 3-540-67752-6<br>(a short version is available as PDF on the *OpenPCS* installation CD) |

**Section 4** of this manual explains the **commissioning of the PLCcore-9G20** based on the Development Kit for the PLCcore-9G20.

**Section 5** describes the **connection assignment** of the PLCcore-9G20.

**Section 6** explains details about the **application of the PLCcore-9G20**, e.g. the **setup of the process image**, the **meaning of control elements** and it provides basic information about programming the module. Moreover, information is given about the usage of CAN interfaces in connection with **CANopen**.

**Section 0** describes **details about the configuration of the PLCcore-9G20**, e.g. the configuration of Ethernet and CAN interfaces, the Linux Autostart procedure as well as choosing the firmware version. Furthermore, the **administration of the PLCcore-9G20** is explained, e.g. the login to the system, the user administration and the execution of software updates.

**Section 8** defines the **adaptation of in- and outputs** as well as the **process image** and it covers the data exchange between a PLC program and a user-specific C/C++ application via **shared process image**.

# 3 Product Description

The PLCcore-9G20 as another innovative product extends the SYS TEC electronic GmbH product range within the field of control applications. In the form of an insert-ready core module, it provides to the user a complete and compact PLC. Due to CAN and Ethernet interfaces, the PLCcore-9G20 is best suitable to perform decentralized control tasks.

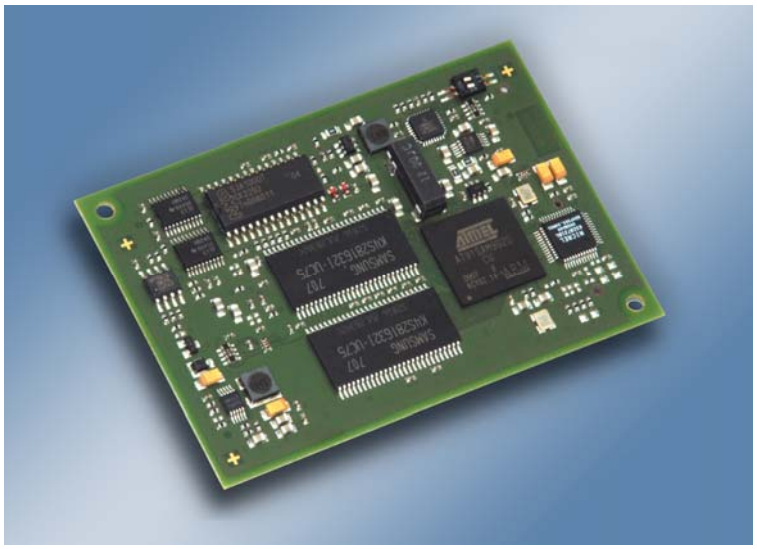


*Figure 1: Top view of the PLCcore-9G20*

These are some significant features of the PLCcore-9G20:

- High-performance CPU kernel (Atmel 32-Bit AT91SAM9G20, 400 MHz CPU Clock, 440 MIPS)
- 32 MByte SDRAM Memory, 16 MByte FLASH Memory
  (max: 64 MByte SDRAM Memory, 64 MByte FLASH Memory)
- 1x 10/100 Mbps Ethernet LAN interface (1x with on–board PHY)
- 1x CAN 2.0B interface, usable as CANopen Manager (CiA 302-conform)
- 5x asynchronous serial ports (UART)
- 19 digital inputs, 8 digital outputs (standard configuration, modifiable via DDK)
- 3 analog inputs
- 4 high-speed counter (Pulse/Dir or A/B)
- 4 PWM-/PTO output (Pulse/Dir)
- Externally usable SPI and $I^2C$
- On-board peripherals: RTC, temperature sensor
- On-board software: Linux, PLC firmware, CANopen Master, HTTP and FTP server
- Programmable in IEC 61131-3 and in C/C++
- Function block libraries for communication (CANopen, Ethernet and UART)
- Function block libraries for hardware components (RTC, Counter, PWM/PTO)
- Support of typical PLC control elements (e.g. Run/Stop switch, Run-LED, Error-LED)
- Linux-based (other user programs may run in parallel)
- Easy, HTML-based configuration via WEB Browser
- Remote Login via Telnet
- Small dimension (78 x 54 mm)

There are different types of firmware available for the PLCcore-9G20. They differ regarding the protocol used for the communication between Programming PC and PLCcore-9G20:

| | |
|---|---|
| Order number: 3390024: | PLCcore-9G20/Z4 (CANopen)<br>communication with Programming PC via CANopen Protocol<br>(Interface CAN0) |
| Order number: 3390025: | PLCcore-9G20/Z5 (Ethernet)<br>communication with Programming PC via UDP Protocol<br>(Interface ETH0) |

Making PLC available as an insert-ready core module with small dimensions reduces effort and costs significantly for the development of user-specific controls. The PLCcore-9G20 is also very well suitable as intelligent network node for decentralized processing of process signals (CANopen and UDP). Additionally, it can be used as basic component for special assemblies or as PLC in hard-to-access areas.

The on-board firmware of the PLCcore-9G20 contains the entire PLC runtime environment including CANopen connection with CANopen master functionality. Thus, the module is able to perform control tasks such as linking in- and outputs or converting rule algorithms. Data and occurrences can be exchanged with other nodes (e.g. superior main controller, I/O slaves and so forth) via CANopen network, Ethernet (UDP protocol) and serial interfaces (UART). Moreover, the number of in- and outputs either is locally extendable or decentralized via CANopen devices. For this purpose, the CANopen-Chip is suitable. It has also been designed as insert-ready core module for the appliance in user-specific applications.

The PLCcore-9G20 provides 19 digital inputs (DI0...DI23, 3.3V level), 8 digital outputs (DO0...DO21, 3.3V level), 4 high-speed counter input and 4 PWM/PTO output. This default I/O configuration can be adapted for specific application requirements by using the Driver Development Kit (SO-1106).Saving the PLC program in the on-board Flash-Disk of the module allows an automatic restart in case of power breakdown.

Programming the PLCcore-9G20 takes place according to IEC 61131-3 using the *OpenPCS* programming system of the company infoteam Software GmbH (http://www.infoteam.de). This programming system has been extended and adjusted for the PLCcore-9G20 by the company SYS TEC electronic GmbH. Hence, it is possible to program the PLCcore-9G20 graphically in KOP/FUB, AS and CFC or textually in AWL or ST. Downloading the PLC program onto the module takes place via Ethernet or CANopen – depending on the firmware that is used. Addressing in- and outputs and creating a process image follows the SYS TEC scheme for compact control units. Like all other SYS TEC controls, the PLCcore-9G20 supports backward documentation of the PLC program as well as the debug functionality including watching and setting variables, single cycles, breakpoints and single steps.

The PLCcore-9G20 is based on Embedded Linux as operating system. This allows for an execution of other user-specific programs while PLC firmware is running. If necessary, those other user-specific programs may interchange data with the PLC program via the process image. More information about this is provided in section 8.

The Embedded Linux applied to the PLCcore-9G20 is licensed under GNU General Public License, version 2. Appendix C contains the license text. All sources of LinuxBSP are included in the software package **SO-1105** ("VMware-Image of the Linux development system for the ECUcore-9G20"). If you require the LinuxBSP sources independently from the VMware-Image of the Linux development system, please contact our support:

support@systec-electronic.com

The PLC system and the PLC- and C/C++ programs developed by the user are **not** subject to GNU General Public License!

# 4 Development Kit PLCcore-9G20

## 4.1 Overview

The Development Kit PLCcore-9G20 is a high-capacity, complete package at a particularly favorable price. Based on a compact PLC, it enables the user to perform decentralized, network-compatible automation projects. Moreover, it facilitates the user to get to know the advantages of graphical and textual PLC programming according to IEC 61131-3 – compared to conventional programming languages.

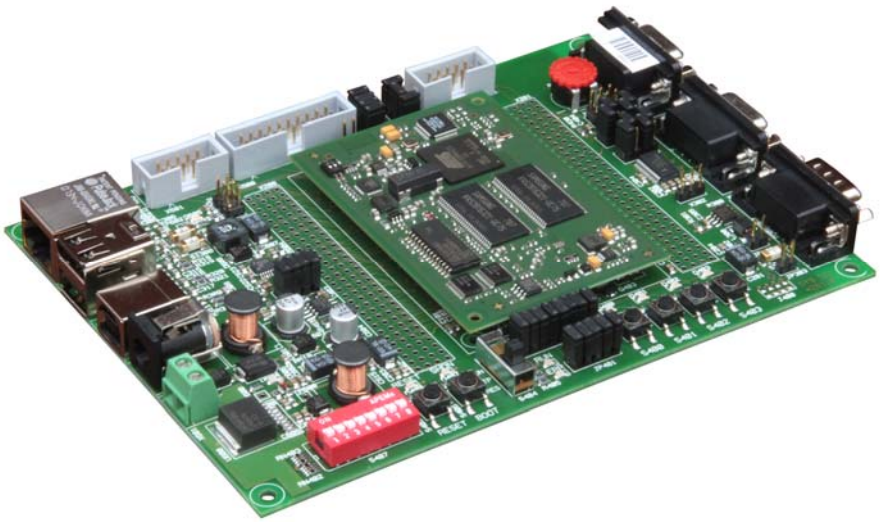


*Figure 2: Development Kit PLCcore-9G20*

The Development Kit PLCcore-9G20 ensures quick and problem-free commissioning of the PLCcore-9G20. Therefore, it combines all hard- and software components that are necessary to create own applications: the core module PLCcore-9G20, the corresponding Development Board containing I/O periphery and numerous interfaces, the *OpenPCS* IEC 61131 programming system as well as further accessory. Thus, the Development Kit forms the ideal platform for developing user-specific applications based on the PLCcore-9G20. It allows for a cost-efficient introduction into the world of decentralized automation technology. All components included in the Kit enable in- and output extensions of the PLCcore-9G20 through CANopen-I/O-assemblies. Thus, the Development Kit may also be used for projects that require PLC with network connection.

The Development Kit PLCcore-9G20 contains the following hardware components:

- PLCcore-9G20
- Development Board for the PLCcore-9G20
- 24V DC Power adapter
- Ethernet cable
- RS232 cable
- CD with programming software, examples, documentation and other tools

The Development Board included in the Kit facilitates quick commissioning of the PLCcore-9G20 and simplifies the design of prototypes for user-specific applications based on this module. Among other equipment, the Development Board comprises different power supply possibilities, Ethernet interface, CAN interface, 4 push buttons and 4 LED as control elements for digital in- and outputs and it comprises a potentiometer for the analog input. Signals that are available from plug connectors of the PLCcore-9G20 are linked to pin header connectors and enable easy connection of own peripheral

circuitry. Hence, the Development Board forms an ideal experimentation and testing platform for the PLCcore-9G20.

The *OpenPCS* IEC 61131 programming system included in the Kit serves as software development platform and as debug environment for the PLCcore-9G20. Thus, the module can either be programmed graphically in KOP/FUB, AS and CFC or textually in AWL or ST. Downloading the PLC program onto the module takes place via Ethernet or CANopen – depending on the firmware that is used. High-capacity debug functionality such as watching and setting variables, single cycles, breakpoints and single steps simplify the development and commissioning of user software for this module.

## 4.2 Electric commissioning of the Development Kit PLCcore-9G20

A 24V DC power adapter necessary for running the Development Kit PLCcore-9G20 and Ethernet and RS232 cables are already included in the Kit delivery. For commissioning the Kit, it is essential to use at least the power supply connections (X700/X701), COM0 (X400 on top) and ETH0 (X500). Furthermore, connection CAN0 (X400 below) is recommended. Table 2 provides an overview over the connections of the Development Kit PLCcore-9G20.

*Table 2: Connections of the Development Kit PLCcore-9G20*

| Connection | Labeling on the Development Board | Remark |
|---|---|---|
| Power supply | X600 oder X601 | The 24V DC power adapter included in the delivery is intended for direct connection to X700. |
| ETH0 (Ethernet) | X304 | This interface serves as communication interface with the Programming PC and is necessary for the program download (PLCcore-9G20/Z5, order number 3390025), besides can be used freely for the user program. |
| COM0 (RS232) | X301 | This interface is used for the configuration of the unit (e.g. setting the IP-address) and can be used freely for general operation of the user program. |
| COM1 (RS232) | X300 | Interface can be used freely for the user program. |
| COM2 (RS232) | X302n | Interface can be used freely for the user program. |
| CAN0 (CAN) | X303 | This interface serves as communication interface with the Programming PC and is necessary for the program download (PLCcore-9G20/Z4, order number 3390024), besides can be used freely for the user program. |

Figure 3 shows the positioning of the most important connections of the Development Board for the PLCcore-9G20. Instead of using the 24V DC power adapter included in the Kit, the power supply may optionally take place via X601 with an external source of 24V/1A.

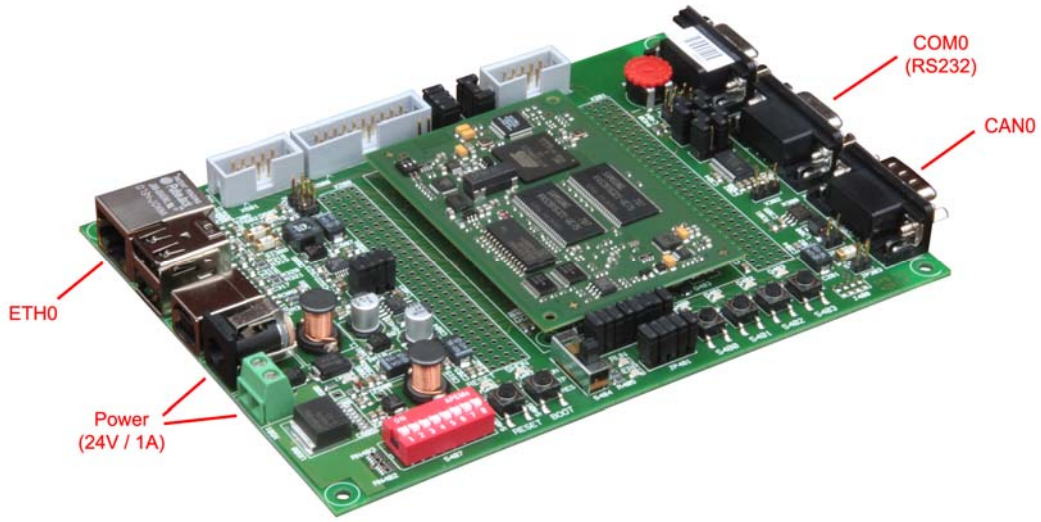



*Figure 3: Positioning of most important connections on the Development Board for the PLCcore-9G20*

| **Advice:** | **Upon commissioning, cables for Ethernet (ETH0, X304) and RS232 (COM0, X301) must be connected prior to activating the power supply (X600 / X601).** |
|---|---|

## 4.3 Control elements of the Development Kit PLCcore-9G20

The Development Kit PLCcore-9G20 allows for easy commissioning of the PLCcore-9G20. It has available various control elements to configure the module and to simulate in- and outputs for the usage of the PLCcore-9G20 as PLC kernel. In Table 3 control elements of the Development Board are listed and their meaning is described.

*Table 3: Control elements of the Development Board for the PLCcore-9G20*

| Control element | Name | Meaning |
|---|---|---|
| Pushbutton 0 | S400 | Digital Input DI0 (Process Image: %IX0.0) |
| Pushbutton 1 | S401 | Digital Input DI1 (Process Image: %IX0.1) |
| Pushbutton 2 | S402 | Digital Input DI2 (Process Image: %IX0.2) |
| Pushbutton 3 | S403 | Digital Input DI3 (Process Image: %IX0.3) |
| LED 0 | D400 | Digital Output DO0 (Process Image: %QX0.0) |
| LED 1 | D401 | Digital Output DO1 (Process Image: %QX0.1) |
| LED 2 | D402 | Digital Output DO2 (Process Image: %QX0.2) |
| LED 3 | D403 | Digital Output DO3 (Process Image: %QX0.3) |
| Poti (ADC) | R429 | Analog Input AI0 (Process Image: %IW8.0) |
| Run/Stop Switch | S404 | Run / Stop to operate the PLC program, Reset control (see section 6.7.1) |

| | | |
|---|---|---|
| Run-LED | D405 | Display of activity state of the PLC (see section 6.7.2) |
| Error-LED | D406 | Display of error state of the PLC (see section 6.7.3) |
| | | |
| DIP-Switch | S407 | Configuration of bitrate and master mode CAN0 (see section 7.4.2) |

Table 8 in section 6.4.1 provides a complete listing of the process image.

## 4.4 Optional accessory

### 4.4.1 USB-RS232 Adapter Cable

The SYS TEC USB-RS232 Adapter Cable (order number 3234000) provides a RS232 interface via an USB-Port of the PC. Together with a terminal program, it enables the configuration of the PLCcore-9G20 from PCs, e.g. laptop computers which do not have RS232 interfaces any more (see section 6.1).

*Figure 4: SYS TEC USB-RS232 Adapter Cable*

### 4.4.2 Driver Development Kit (DDK)

The ECUcore-9G20 Driver Development Kit (order number SO-1106) allows the user to independently adjust the I/O level to his own baseboard. Section 8.2 provides information about the Driver Development Kit.

# 5 Pinout of the PLCcore-9G20

Connections of the PLCcore-9G20 are directed to the outside via two female headers that are double-row and mounted on the bottom of the module (X500, see Figure 5). Appropriate pin header connectors as correspondent to the PLCcore-9G20 are available from company “W + P“:

W+P name: SMT Pin Headers, 1.27mm Pitch, Vertical, Double Row - 1.0mm Body
W+P order number: 7072-100-10-00-10-PPST (deliverable in other sizes)

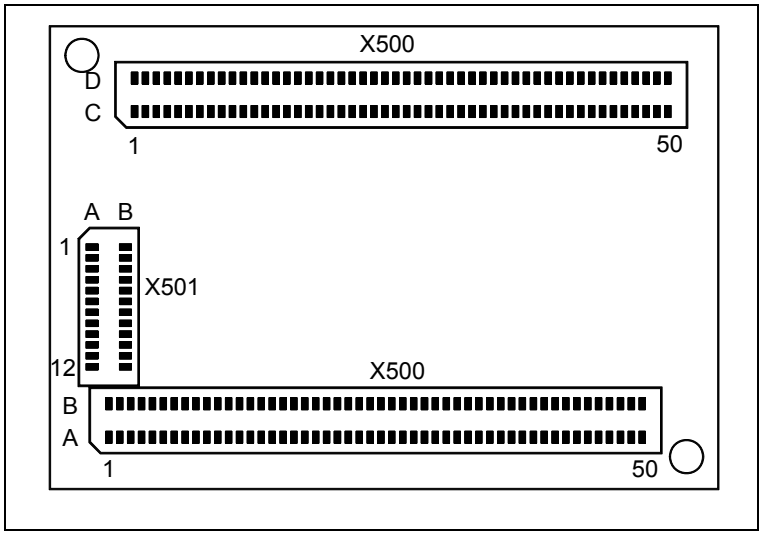



*Figure 5: Pinout of the PLCcore-9G20 - top view*

Figure 5 exemplifies the positioning of female headers (X500) on the PLCcore-9G20. The complete connection assignment of this module is listed up in Table 4. The additional female header X501 shown in Figure 5 is reserved for a JTAG interface. It is only equipped on special development boards. For the usage of the PLCcore-9G20 as PLC kernel it is without any importance. A detailed description of all module connectors is located in the Hardware Manual ECUcore-9G20 (Manual no.: L-1255). Appendix B includes reference designs for using the PLCcore-9G20 in customer-specific applications.

*Table 4: Connections of the PLCcore-9G20, completely, sorted by connection pin*

| Signal | Pin | Pin | Signal | | Signal | Pin | Pin | Signal |
|---|---|---|---|---|---|---|---|---|
| GND | A01 | B01 | GND | | GND | C01 | D01 | +2V5_EPHY |
| /BOOT | A02 | B02 | /MR | | ETH0_TX- | C02 | D02 | GND |
| WKUP | A03 | B03 | /RESET | | ETH0_TX+ | C03 | D03 | ETH_SPEED |
| SHDN | A04 | B04 | /PFI | | ETH0_RX+ | C04 | D04 | ETH_LINK/ACT |
| BMS | A05 | B05 | WDI | | ETH0_RX- | C05 | D05 | GND |
| GND | A06 | B06 | PS_IO | | GND | C06 | D06 | AD0 |
| DRXD | A07 | B07 | GND | | ADTRG | C07 | D07 | AD1 |
| DTXD | A08 | B08 | RTS0 | | ADVREF | C08 | D08 | AD2 |
| DSR0 | A09 | B09 | CTS0 | | GND | C09 | D09 | GND |
| DTR0 | A10 | B10 | RTS1 | | SD_MCDA0 | C10 | D10 | SD_MCDB0 |
| DCD0 | A11 | B11 | CTS1 | | SD_MCDA1 | C11 | D11 | SD_MCDB1 |
| GND | A12 | B12 | GND | | SD_MCDA2 | C12 | D12 | SD_MCDB2 |
| TXD0 | A13 | B13 | TXD1 | | SD_MCDA3 | C13 | D13 | SD_MCDB3 |
| RXD0 | A14 | B14 | RXD1 | | SD_MCCK | C14 | D14 | SD_MCCDA |
| TXD2 | A15 | B15 | TXD3 | | GND | C15 | D15 | SD_MCCDB |

| Signal | Pin | Pin | Signal | | Signal | Pin | Pin | Signal |
|---|---|---|---|---|---|---|---|---|
| RXD2 | A16 | B16 | RXD3 | | SCK0 | C16 | D16 | GND |
| GND | A17 | B17 | TXD5 | | SCK1 | C17 | D17 | TIOA1 |
| USB_HDPA | A18 | B18 | RXD5 | | SCK2 | C18 | D18 | TIOB1 |
| USB_HDMA | A19 | B19 | GND | | PCK1 | C19 | D19 | TIOA2 |
| USB_HDPB | A20 | B20 | USB_DDP | | RK0 | C20 | D20 | TIOB2 |
| USB_HDMB | A21 | B21 | USB_DDM | | TK0 | C21 | D21 | TD0 |
| GND | A22 | B22 | GND | | RF0 | C22 | D22 | RD0 |
| I2C_DATA | A23 | B23 | CAN_TXD | | TF0 | C23 | D23 | GND |
| I2C_CLK | A24 | B24 | CAN_RXD | | GND | C24 | D24 | FPGA_IO0 |
| GND | A25 | B25 | CAN_VCC | | FPGA_IO1 | C25 | D25 | FPGA_IO2 |
| FPGA_IO44 | A26 | B26 | GND | | FPGA_IO3 | C26 | D26 | FPGA_IO4 |
| FPGA_IO46 | A27 | B27 | FPGA_IO45 | | FPGA_IO5 | C27 | D27 | FPGA_IO6 |
| FPGA_IO48 | A28 | B28 | FPGA_IO47 | | FPGA_IO7 | C28 | D28 | GND |
| FPGA_IO50 | A29 | B29 | FPGA_IO49 | | GND | C29 | D29 | FPGA_IO8 |
| FPGA_IO52 | A30 | B30 | FPGA_IO51 | | FPGA_IO9 | C30 | D30 | FPGA_IO10 |
| GND | A31 | B31 | FPGA_IO53 | | FPGA_IO11 | C31 | D31 | FPGA_IO12 |
| FPGA_IO54 | A32 | B32 | GND | | FPGA_IO13 | C32 | D32 | FPGA_IO14 |
| FPGA_IO56 | A33 | B33 | FPGA_IO55 | | FPGA_IO15 | C33 | D33 | GND |
| FPGA_IO58 | A34 | B34 | FPGA_IO57 | | FPGA_IO17 | C34 | D34 | FPGA_IO16 |
| FPGA_IO60 | A35 | B35 | FPGA_IO59 | | GND | C35 | D35 | FPGA_IO18 |
| FPGA_IO62 | A36 | B36 | FPGA_IO61 | | FPGA_IO19 | C36 | D36 | FPGA_IO20 |
| GND | A37 | B37 | FPGA_IO63 | | FPGA_IO21 | C37 | D37 | FPGA_IO22 |
| FPGA_IO64 | A38 | B38 | GND | | FPGA_IO23 | C38 | D38 | FPGA_IO24 |
| FPGA_IO66 | A39 | B39 | FPGA_IO65 | | FPGA_IO25 | C39 | D39 | GND |
| FPGA_IO68 | A40 | B40 | FPGA_IO67 | | FPGA_IO27 | C40 | D40 | FPGA_IO26 |
| FPGA_IO70 | A41 | B41 | FPGA_IO69 | | GND | C41 | D41 | FPGA_IO28 |
| FPGA_IO72 | A42 | B42 | FPGA_IO71 | | FPGA_IO29 | C42 | D42 | FPGA_IO30 |
| GND | A43 | B43 | FPGA_IO73 | | FPGA_IO31 | C43 | D43 | FPGA_IO32 |
| FPGA_IO74 | A44 | B44 | GND | | FPGA_IO33 | C44 | D44 | FPGA_IO34 |
| FPGA_IO76 | A45 | B45 | FPGA_IO75 | | FPGA_IO35 | C45 | D45 | GND |
| FPGA_IO78 | A46 | B46 | FPGA_IO77 | | FPGA_IO37 | C46 | D46 | FPGA_IO36 |
| FPGA_IO80 | A47 | B47 | FPGA_IO79 | | GND | C47 | D47 | FPGA_IO38 |
| VBAT | A48 | B48 | FPGA_IO81 | | FPGA_IO39 | C48 | D48 | FPGA_IO40 |
| GND | A49 | B49 | GND | | FPGA_IO41 | C49 | D49 | FPGA_IO42 |
| +3V3 | A50 | B50 | +3V3 | | FPGA_IO43 | C50 | D50 | GND |

Table 5 is a subset of Table 4 and only includes all in- and outputs of the PLCcore-9G20 sorted by their function.

*Table 5: Connections of the PLCcore-9G20, only I/O, sorted by function*

| Connector | I/O-Pin (FPGA) | PLC Function 1 | PLC Function 2<br>A=alternative, S=simultaneous |
|---|---|---|---|
| | | | |
| B43 | IO73 | DI0 [Switch0] | S: CNTR0 (IN/A) |
| A44 | IO74 | DI1 [Switch1] | S: CNTR0 ('DIR/B) |
| B45 | IO75 | DI2 [Switch2] | S: CNTR1 (IN/A) |
| A45 | IO76 | DI3 [Switch3] | S: CNTR1 ('DIR/B) |
| B39 | IO65 | DI4 | |
| A39 | IO66 | DI5 | |
| B40 | IO67 | DI6 | |
| A40 | IO68 | DI7 | |
| | | | |

| | | | |
|---|---|---|---|
| A29 | IO50 | DI8 | S: CNTR2 (IN/A) |
| B30 | IO51 | DI9 | S: CNTR2 (DIR/B) |
| A30 | IO52 | DI10 | S: CNTR3 (IN/A) |
| B31 | IO53 | DI11 | S: CNTR3 (DIR/B) |
| A32 | IO54 | DI12 | |
| B33 | IO55 | DI13 | |
| A33 | IO56 | DI14 | |
| B34 | IO57 | DI15 | |
| | | | |
| A34 | IO58 | DI16 | |
| B35 | IO59 | DI17 | |
| A35 | IO60 | DI18 | |
| | | | |
| B41 | IO69 | DO0 [LED0] | A: PWM0 (OUT) |
| A41 | IO70 | DO1 [LED1] | A: PWM1 (OUT) |
| B42 | IO71 | DO2 [LED2] | A: PWM2 (OUT) |
| A42 | IO72 | DO3 [LED3] | A: PWM3 (OUT) |
| B36 | IO61 | DO4 | |
| A36 | IO62 | DO5 | |
| B37 | IO63 | DO6 | |
| A38 | IO64 | DO7 | |
| | | | |
| A46 | IO78 | /Error-LED | |
| B46 | IO77 | /Run-LED | |
| B47 | IO79 | R/S/M-Switch | |
| A47 | IO80 | R/S/M-Switch | |
| B48 | IO81 | R/S/M-Switch | |
| | | | |

Table 6 defines the coding of the Run/Stop switch. Functionality of the Run/Stop switch for PLC firmware is explained in section 6.7.1. If no Run/Stop switch is intended for the usage of the PLCcore-9G20 on an application-specific baseboard, the coding for *"Run"* must be hard-wired at the module connections (also see reference design in Appendix B).

*Table 6: Coding of the Run/Stop switch*

| Modus | Pin B47 (IO79) | Pin A47 (IO80) | Pin B48 (IO81) |
|---|---|---|---|
| Run | 1 | 0 | 1 |
| Stop | 1 | 1 | 0 |
| MRes | 0 | 0 | 0 |

# 6 PLC Functionality of the PLCcore-9G20

## 6.1 Overview

The PLCcore-9G20 realizes a complete Linux-based compact PLC as an insert-ready core ("Core"). There, the PLCcore-9G20 is based on the hardware ECUcore-9G20 and extends it by PLC-specific functionality (FPGA software, PLC firmware). Both modules, the ECUcore-9G20 and the PLCcore-9G20, use the same Embedded Linux as operating system. Consequently, the configuration and the C/C++ programming of the PLCcore-9G20 are almost identical with the ECUcore-9G20.

## 6.2 System start of the PLCcore-9G20

By default, the PLCcore-9G20 loads all necessary firmware components upon Power-on or Reset and starts running the PLC program afterwards. Hence, the PLCcore-9G20 is suitable for the usage in autarchic control systems. In case of power breakdown, such systems resume the execution of the PLC program independently and without user intervention. Figure 6 shows the system start in detail:

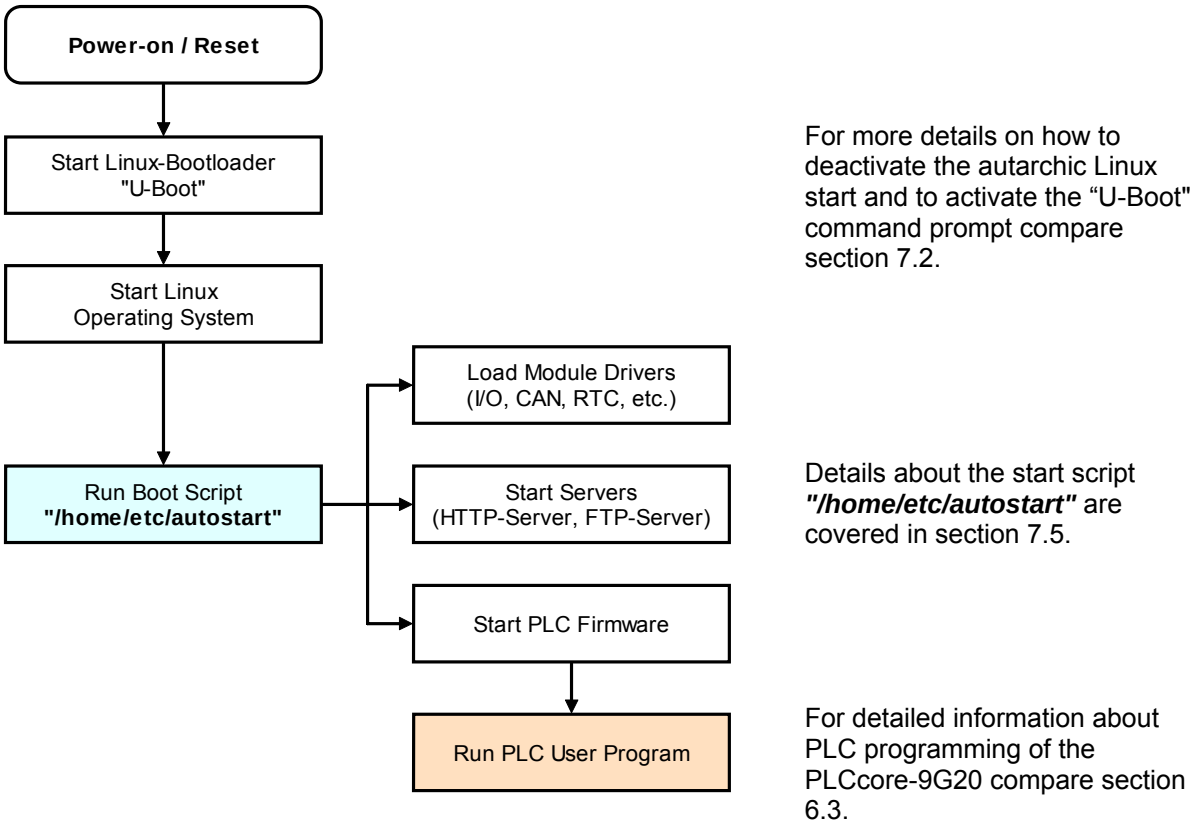



*Figure 6: System start of the PLCcore-9G20*

## 6.3 Programming the PLCcore-9G20

The PLCcore-9G20 is programmed with IEC 61131-3-conform *OpenPCS* programming environment. There exist additional manuals about *OpenPCS* that describe the handling of this programming tool. Those are part of the software package *"OpenPCS"*. All manuals relevant for the PLCcore-9G20 are listed in Table 1.

PLCcore-9G20 firmware is based on standard firmware for SYS TEC`s compact control units. Consequently, it shows identical properties like other SYS TEC control systems. This affects especially the process image setup (see section 6.4) as well as the functionality of control elements (Hex-Encoding switch, DIP-Switch, Run/Stop switch, Run-LED, Error-LED).

Depending on the firmware version used, PLCcore-9G20 firmware provides numerous function blocks to the user to access communication interfaces. Table 7 specifies the availability of FB communication classes (SIO, CAN, UDP) for different PLCcore-9G20 firmware versions. Section 7.6 describes the selection of the appropriate firmware version.

*Table 7: Support of FB communication classes for different types of the PLCcore*

| Type of Interface | PLCcore-9G20/Z3 Art. no: 3390023 | PLCcore-9G20/Z4 Art. no: 3390024 | PLCcore-9G20/Z5 Art. no: 3390025 | Remark |
|---|---|---|---|---|
| CAN | - | x | X | FB description see manual L-1008 |
| UDP | - | x | X | FB description see manual L-1054 |
| SIO | x | x | X | FB description see manual L-1054 |

Table 22 in Appendix A contains a complete listing of firmware functions and function blocks that are supported by the PLCcore-9G20.

Detailed information about using the CAN interfaces in connection with CANopen is provided in section 6.9.

## 6.4 Process image of the PLCcore-9G20

### 6.4.1 Local In- and Outputs

Compared to other SYS TEC compact control systems, the PLCcore-9G20 obtains a process image with identical addresses. All in- and outputs listed in Table 8 are supported by the PLCcore-9G20.

*Table 8: Assignment of in- and outputs to the process image of the PLCcore-9G20*

| I/O of the PLCcore-9G20 | Address and Data type in the Process Image |
|---|---|
| DI0 ... DI7 | **%IB0.0** as Byte with DI0 … DI7<br>%IX0.0 … %IX0.7 as single Bit for each input |
| DI8 ... DI15 | **%IB1.0** as Byte with DI8 … DI15<br>%IX1.0 … %IX1.7 as single Bit for each input |
| DI16 … DI23<br>(DI19 … DI23 as user specific extension only) | **%IB2.0** as Byte with DI16 … DI23<br>%IX2.0 … %IX2.7 as single Bit for each input |
| DI24 ... DI31<br>(as user specific extension only) | **%IB3.0** as Byte with DI24 … DI31<br>%IX3.0 … %IX3.7 as single Bit for each input |
| DI32 ... DI39<br>(as user specific extension only) | **%IB4.0** as Byte with DI32 … DI139<br>%IX4.0 … %IX4.7 as single Bit for each input |
| DI40 … DI47<br>(as user specific extension only) | **%IB5.0** as Byte with DI40 … DI47<br>%IX5.0 … %IX5.7 as single Bit for each input |
| AI0<br>(external ADC of the Development Board), see **(1)** | **%IW8.0** 15Bit + sign (0 … +32767) |
| C0 | **%ID40.0** 31Bit + sign ($-2^{31}$ - $2^{31}$ -1)<br>counter input: DI0, direction: DI1, see section 6.6.1 |
| C1 | **%ID44.0** 31Bit + sign ($-2^{31}$ - $2^{31}$ -1)<br>counter input: DI2, direction: DI3, see section 6.6.1 |
| C2 | **%ID48.0** 31Bit + sign ($-2^{31}$ - $2^{31}$ -1)<br>counter input: DI8 direction: DI9, see section 6.6.1 |
| C3 | **%ID52.0** 31Bit + sign ($-2^{31}$ - $2^{31}$ -1)<br>counter input: DI10, direction: DI11, see section 6.6.1 |
| On-board Temperature Sensor, see **(1)** | **%ID72.0** 31Bit + sign as 1/10000 °C |
| DO0 ... DO7 | **%QB0.0** as Byte with DO0 … DO7<br>%QX0.0 … %QX0.7 as single Bit for each output |
| DO8 … DO15<br>(as user specific extension only) | **%QB1.0** as Byte with DO8 … DO15<br>%QX1.0 … %QX1.7 as single Bit for each output |
| DO16 … DO23<br>(as user specific extension only) | **%QB2.0** as Byte with DO16 … DO23<br>%QX2.0 … %QX2.7 as single Bit for each output |
| DO24 … DO31<br>(as user specific extension only) | **%QB3.0** as Byte with DO24 … DO31<br>%QX3.0 … %QX3.7 as single Bit for each output |
| DO32 … DO39<br>(as user specific extension only) | **%QB4.0** as Byte with DO32 … DO39<br>%QX4.0 … %QX4.7 as single Bit for each output |
| DO40 … DO47<br>(as user specific extension only) | **%QB5.0** as Byte with DO40 … DO47<br>%QX5.0 … %QX5.7 as single Bit for each output |

| | |
|---|---|
| P0 | **%QX0.0** (default value for inactive generator)<br>Impulse output: DO0, see section 6.6.2 |
| P1 | **%QX0.1** (default value for inactive generator)<br>Impulse output: DO1, see section 6.6.2 |
| P2 | **%QX0.2** (default value for inactive generator)<br>Impulse output: DO2, see section 6.6.2 |
| P3 | **%QX0.3** (default value for inactive generator)<br>Impulse output: DO3, see section 6.6.2 |

(1) This marked components are only available in the process image, if the **Option *"Enable extended I/Os"*** is activated within the PLC configuration (see section 7.4.1). Alternatively, entry *"EnableExtIo="* can directly be set within section *"[ProcImg]"* of the configuration file ***"/home/plc/plccore-9g20.cfg"*** (see section 7.4.3). The appropriate configuration setting is evaluated upon start of the PLC firmware.

In- and outputs of thePLCcore-9G20 are not negated in the process image. Hence, the H-level at one input leads to value "1" at the corresponding address in the process image. Contrariwise, value "1" in the process image leads to an H-level at the appropriate output.

### 6.4.2 In- and outputs of user-specific baseboards

The connection lines leading towards the outside provides to the user most effective degrees of freedom for designing the in-/output circuit of the PLCcore-9G20. Therewith, all in- and outputs of the PLCcore-9G20 can be flexibly adjusted to respective requirements. This implicates that the process image of PLCcore-9G20 is significantly conditioned by the particular, user-specific in-/output circuit. Including the software for in-/output components into the process image requires the "*Driver Development Kit for ECUcore-9G20*" (order number SO-1106).

## 6.5 Communication interfaces

### 6.5.1 Serial interfaces

The PLCcore-9G20 features 5 serial interfaces (COM0 … COM4) that function as RS-232. Details about hardware activation are included in the *"Hardware Manual Development Board ECUcore-9G20"* (Manual no.: L-1256).

**COM0:** Interface COM0 primarily serves as service interface to administer the PLCcore-9G20. By default, in boot script *"/etc/inittab"* it is assigned to the Linux process *"getty"* and is used as Linux console to administer the PLCcore-9G20. Even though interface COM0 may be used from a PLC program via function blocks of type *"SIO_Xxx"* (see manual *"SYS TEC-specific Extensions for OpenPCS / IEC 61131-3"*, Manual no.: L-1054), only signs should be output in this regard. The module tries to interpret and to execute signs that it receives as Linux commands.

To freely use an interface from a PLC program, boot script *"/etc/inittab"* must be adjusted appropriately which is only possible by modifying the Linux image. This requires software package SO-1105 ("VMware-Image of the Linux Development System for the ECUcore-9G20").

**COM1..4:** Interfaces COM1 … COM4 are disposable and support data exchange between the PLCcore-9G20 and other field devices kept under control of the PLC program.

Interfaces COM1 … COM4 may be used from a PLC program via function blocks of type *"SIO_Xxx"* (see manual *"SYS TEC-specific Extensions for OpenPCS / IEC 61131-3"*, Manual no.: L-1054).

### 6.5.2 CAN interfaces

The PLCcore-9G20 features 1 CAN interface (CAN0). Details about the hardware activation are included in the *"Hardware Manual Development Board ECUcore-9G20"* (Manual no.: L-1256).

The CAN interface allow for data exchange with other devices via network variables and they are accessible from a PLC program via function blocks of type *"CAN_Xxx"* (see section 6.9 and *"User Manual CANopen Extension for IEC 61131-3"*, Manual no.: L-1008).

Section 6.9 provides detailed information about the usage of the CAN interface in connection with CANopen.

### 6.5.3 Ethernet interfaces

The PLCcore-9G20 features 1 Ethernet interface (ETH0). Details about the hardware activation are included in the *"Hardware Manual Development Board ECUcore-9G20"* (Manual no.: L-1256).

The Ethernet interface serves as service interface to administer the PLCcore-9G20 and it enables data exchange with other devices. The interface is accessible from a PLC program via function blocks of type *"LAN_Xxx"* (see manual *"SYS TEC-specific Extensions for OpenPCS / IEC 61131-3"*, Manual no.: L-1054).

The exemplary PLC program *"UdpRemoteCtrl"* illustrates the usage of function blocks of type *"LAN_Xxx"* within a PLC program.

## 6.6 Specific peripheral interfaces

### 6.6.1 Counter inputs

The PLCcore-9G20 features 4 fast counter inputs (C0 … C3). Prior to its usage, all counter inputs must be parameterized via function block *"CNT_FUD"* (see manual *"SYS TEC-specific Extensions for OpenPCS / IEC 61131 3"*, Manual no.: L 1054). Afterwards, in a PLC program the current counter value is accessible via process image (see Table 8 in section 6.4.1) or via function block *"CNT_FUD"*. Table 9 lists the allocation between counter channels and inputs.

*Table 9: Allocation between counter channels and inputs*

| Counter channel | Counter input | | Optional direction input | | Counter value in process image |
|---|---|---|---|---|---|
| C0 | C0 (DI0) | %IX0.0 | DI1 | %IX0.1 | %ID40.0 |
| C1 | C1 (DI2) | %IX0.2 | DI3 | %IX0.3 | %ID44.0 |
| C2 | C2 (DI8) | %IX1.0 | DI9 | %IX1.1 | %ID48.0 |
| C3 | C2 (DI10) | %IX1.2 | DI11 | %IX1.3 | %ID52.0 |

To ensure the minimum slew rate for the counter inputs, required by FPGA, it is necessary to use the interface connection as shown in Figure 35 in Appendix B. A too small slew rate may lead to wrong counter values.

### 6.6.2 Pulse outputs

To release PWM and PTO signal sequences, the PLCcore-9G20 features 4 pulse outputs (P0 ... P3). Prior to its usage, all pulse outputs must be parameterized using function block *"PTO_PWM"* (see manual *"SYS TEC-specific Extensions for OpenPCS / IEC 61131 3"*, Manual no.: L 1054). After the impulse generator is started, it takes over the control of respective outputs. After the impulse generator is deactivated, the respective output adopts the corresponding value that is filed in the process image for this output (see Table 8 in section 6.4.1). Table 10 lists the allocations between impulse channels and outputs.

*Table 10: Allocation between impulse channels and outputs*

| Impulse channel | Impulse output | |
|---|---|---|
| P0 | P0 (DO0) | %QX0.0 |
| P1 | P1 (DO1) | %QX0.1 |
| P2 | P0 (DO2) | %QX0.2 |
| P3 | P1 (DO3) | %QX0.3 |

## 6.7 Control and display elements

### 6.7.1 Run/Stop switch

Module connections *"IO79"*, *"IO80"* and *"IO81"* (see Table 5 and see reference design in Appendix B) are designed to connect a Run/Stop switch. Using this Run/Stop switch makes it possible to start and interrupt the execution of the PLC program. Together with start and stop pushbuttons of the *OpenPCS* programming environment, the Run/Stop switch represents a "logical" AND-relation. This means that the PLC program will not start the execution until the local Run/Stop switch is positioned to *"Run"* **AND** additionally the start command (cold, warm or hot start) is given by the *OpenPCS* user interface. The order hereby is not relevant. A run command given by *OpenPCS* while at the same time the Run/Stop switch is positioned to *"Stop"* is visible through quick flashing of the Run-LED (green).

Positioned to *"MRes"* (*"Modul Reset"*), the Run/Stop switch allows for local deletion of a PLC program from the PLCcore-9G20. This might for example be necessary if an error occurs and the PLC program is running an infinite loop and consequently, accessing the *OpenPCS* programming environment is no longer possible. The procedure for deleting a PLC program is described in section 6.8.

### 6.7.2 Run-LED (green)

The module connection *"/Run-LED"* (see Table 5 and reference design in Appendix B) is designed for connecting a Run-LED. This Run-LED provides information about the activity state of the control system. The activity state is shown through different modes:

*Table 11: Display status of the Run-LED*

| LED Mode | PLC Activity State |
|---|---|
| Off | The PLC is in state *"Stop"*:<br>• the PLC does not have a valid program,<br>• the PLC has received a stop command from the *OpenPCS* programming environment or<br>• the execution of the program has been canceled due to an internal error |
| Quick flashing in relation 1:8 to pulse | The PLC is on standby but is not yet executing:<br>• The PLC has received a start command from the *OpenPCS* programming environment but the local Run/Stop switch is still positioned to *"Stop"* |
| Slow flashing in relation 1:1 to pulse | The PLC is in state *"Run"* and executes the PLC program. |
| Quick flashing in relation 1:1 to pulse | The PLC is in mode *"Reset"*, compare section 6.8 |

### 6.7.3 Error-LED (red)

Module connection *"/Error-LED"* (see Table 5 and reference design in Appendix B) is designed for connecting an Error-LED. This Error-LED provides information about the error state of the control system. The error state is represented through different modes:

*Table 12: Display status of the Error-LED*

| LED Mode | PLC Error State |
|---|---|
| Off | No error has occurred; the PLC is in normal state. |
| Permanent light | A severe error has occurred:<br>• The PLC was started using an invalid configuration (e.g. CAN node address 0x00) and had to be stopped or<br>• A severe error occurred during the execution of the program and caused the PLC to independently stop its state *"Run"* (division by zero, invalid Array access, ...), see below |
| Slow flashing in relation 1:1 to pulse | A network error occurred during communication to the programming system; the execution of a running program is continued. This error state will be reset independently by the PLC as soon as further communication to the programming system is successful. |
| Quick flashing in relation 1:1 to pulse | The PLC is in mode *"Reset"*, compare section 6.8. |
| Quick flashing in relation 1:8 to pulse | The PLC is on standby, but is not yet running:<br>• The PLC has received a start command from the *OpenPCS* programming environment but the local Run/Stop switch is positioned to *"Stop"* |

In case of severe system errors such as division by zero of invalid Array access, the control system passes itself from state *"Run"* into state *"Stop"*. This is recognizable by the permanent light of the

Error-LED (red). In this case, the error cause is saved by the PLC and is transferred to the computer and shown upon next power-on.

## 6.8 Local deletion of a PLC program

If the Run/Stop switch is positioned to *"MRes"* (*"Modul Reset"*) (see section 6.7.1), it is possible to delete a program from the PLCcore-9G20. This might for example be necessary if an error occurs and the PLC program is running an infinite loop and consequently, accessing the *OpenPCS* programming environment is no longer possible. To prevent deleting a PLC program by mistake, it is necessary to keep to the following order:

(1) Position the Run/Stop switch to *"MRes"*

(2) Reset the PLCcore-9G20 (by pressing the reset pushbutton of the Development Board or through temporary power interrupt)
⇨ Run-LED (green) is flashing quickly in relation 1:1 to the pulse

(3) Position the Run/Stop switch to *"Run"*
⇨ Error-LED (red) is flashing quickly in relation 1:1 to the pulse

(4) Reposition Run/Stop switch back to *"MRes"* **within 2 seconds**
⇨ PLCcore-9G20 is deleting PLC program
⇨ Run-LED (green) and Error-LED (red) are both flashing alternately

(5) Reposition Run/Stop switch to *"Stop"* or *"Run"* and reset again to start the PLCcore-9G20 and bring it into normal working state

If Reset of the PLCcore-9G20 is activated (e.g. through temporary power interrupt) while at the same time the Run/Stop switch is positioned to *"MRes"*, the module recognizes a reset requirement. This is visible through quick flashing of the Run-LED (green). This mode can be stopped without risk. Therefore, the Run/Stop switch must be positioned to *"Run"* or *"Stop"* (Error-LED is flashing) and it must be waited for 2 seconds. The PLCcore-9G20 independently stops the reset process after 2 seconds and starts a normal working state with the PLC program which was saved last.

## 6.9 Using CANopen for CAN interfaces

The PLCcore-9G20 features 1 CAN interface (CAN0), usable as CANopen Manager (conform to CiA Draft Standard 302). The configuration of this interface (active/inactive, node number, Bitrate, Master on/off) is described in section 7.4.

The CAN interface allow for data exchange with other devices via network variables and is usable from a PLC program via function blocks of type *"CAN_Xxx"*. More details are included in *"User Manual CANopen Extension for IEC 61131-3"*, Manual no.: L-1008.

The CANopen services **PDO** (***P***rocess ***D***ata ***O***bjects) and **SDO** (***S***ervice ***D***ata ***O***bjects) are two separate mechanisms for data exchange between single field bus devices. Process data sent from a node (**PDO**) are available as broadcast to interested receivers. PDOs are limited to 1 CAN telegram and therewith to 8 Byte user data maximum because PDOs are executed as non-receipt broadcast messages. On the contrary, **SDO** transfers are based on logical point-to-point connections ("Peer to Peer") between two nodes and allow the receipted exchange of data packages that may be larger than 8 Bytes. Those data packages are transferred internally via an appropriate amount of CAN telegrams. Both services are applicable for interface CAN0 as well as for CAN1 of the PLCcore-9G20.

SDO communication basically takes place via function blocks of type *"CAN_SDO_Xxx"* (see *"User Manual CANopen Extension for IEC 61131-3"*, Manual no.: L-1008). Function blocks are also available for PDOs (*"CAN_PDO_Xxx"*). Those should only be used for particular cases in order to also activate non-CANopen-conform devices. For the application of PDO function blocks, the CANopen configuration must be known in detail. The reason for this is that the PDO function blocks only use 8 Bytes as input/output parameter, but the assignment of those Bytes to process data is subject to the user.

Instead of PDO function blocks, network variables should mainly be used for PDO-based data exchange. Network variables represent the easiest way of data exchange with other CANopen nodes. Accessing network variables within a PLC program takes place in the same way as accessing internal, local variables of the PLC. Hence, for PLC programmers it is not of importance if e.g. an input variable is allocated to a local input of the control or if it represents the input of a decentralized extension module. The application of network variables is based on the integration of DCF files that are generated by an appropriate CANopen configurator. On the one hand, DCF files describe communication parameters of any device (CAN Identifier, etc.) and on the other hand, they allocate network variables to the Bytes of a CAN telegram (mapping). The application of network variables only requires basic knowledge about CANopen.

In a CANopen network, exchanging PDOs only takes place in status *"OPERATIONAL"*. If the PLCcore-9G20 is not in this status, it does not process PDOs (neither for send-site nor for receive-site) and consequently, it does not update the content of network variables. The CANopen Manager is in charge of setting the operational status *"OPERATIONAL"*, *"PRE-OPERATIONAL"* etc. (mostly also called "CANopen Master"). In typical CANopen networks, a programmable node in the form of a PLC is used as CANopen-Manager. The PLCcore-9G20 is optionally able to take over tasks of the CANopen Manager. How the Manager is activated is described in section 7.4.

As CANopen Manager, the PLCcore-9G20 is able to parameterize the CANopen I/O devices ("CANopen-Slaves") that are connected to the CAN bus. Therefore, upon system start via SDO it transfers DCF files generated by the CANopen configurator to the respective nodes.

### 6.9.1 CAN interface CAN0

Interface CAN0 features a dynamic object dictionary. This implicates that after activating the PLC, the interface does not provide communication objects for data exchange with other devices. After downloading a PLC program (or its reload from the non-volatile storage after power-on), the required communication objects are dynamically generated according to the DCF file which is integrated in the PLC project. Thus, CAN interface CAN0 is extremely flexible and also applicable for larger amount of data.

For the PLC program, all network variables are declared as *"VAR_EXTERNAL"* according to IEC61131-3. Hence, they are marked as „outside of the control“, e.g.:

```
VAR_EXTERNAL
    NetVar1 : BYTE ;
    NetVar2 : UINT ;
END_VAR
```

A detailed procedure about the integration of DCF files into the PLC project and about the declaration of network variables is provided in manual *"User Manual CANopen Extension for IEC 61131-3"* (Manual no.: L-1008).

When using CAN interface CAN0 it must be paid attention that the generation of required objects takes place upon each system start. This is due to the dynamic object directory. "Design instructions" are included in the DCF file that is integrated in the PLC project. **Hence, changes to the configuration can only be made by modifying the DCF file.** This implies that after the network configuration is changed (modification of DCF file), the PLC project must again be translated and loaded onto the PLCcore-9G20.

### 6.9.2 Additional CAN interfaces

In general, the PLC firmware used for PLCcore-9G20 is able to simultaneously operate several CAN interfaces (like other PLC types such as the PLCcore-5484 or PLCmodule-C32).

If necessary, more CAN interfaces can be connected to the module externally. Please contact our support employee if you are interested in this option:

support@systec-electronic.com

# 7 Configuration and Administration of the PLCcore-9G20

## 7.1 System requirements and necessary software tools

The administration of the PLCcore-9G20 requires any Windows or Linux computer that has available an Ethernet interface and a serial interface (RS232). As alternative solution to the on-board serial interface, SYS TEC offers a USB-RS232 Adapter Cable (order number 3234000, see section 4.4.1) that provides an appropriate RS232 interface via USB port.

All examples referred to in this manual are based on an administration of the PLCcore-9G20 using a Windows computer. Procedures using a Linux computer would be analogous.

To administrate the PLCcore-9G20 the following software tools are necessary:

**Terminal program**

A Terminal program allows the communication with the **command shell** of the PLCcore-9G20 via a **serial RS232 connection to COM0 of the PLCcore-9G20**. This is required for the Ethernet configuration of the PLCcore-9G20 as described in section 7.3. After completing the Ethernet configuration, all further commands can either be entered in the Terminal program or alternatively in a Telnet client (see below).

Suitable as Terminal program would be *"HyperTerminal"* which is included in the Windows delivery or *"TeraTerm"* which is available as Open Source and meets higher demands (downloadable from: http://ttssh2.sourceforge.jp).

**Telnet client**

Telnet-Client allows the communication with **command shell** of the PLCcore-9G20 via **Ethernet connection to ETH0 of the PLCcore-9G20**. Using Telnet clients requires a completed Ethernet configuration of the PLCcore-9G20 according to section 7.3. As alternative solution to Telnet client, all commands can be edited via a Terminal program (to COM0 of the PLCcore-9G20).

Suitable as Telnet client would be *"Telnet"* which is included in the Windows delivery or *"TeraTerm"* which can also be used as Terminal program (see above).

**FTP client**

An FTP client allows for file exchange between the PLCcore-9G20 (ETH0) and the computer. This allows for example **editing configuration files** by transferring those from the PLCcore-9G20 onto the computer where they can be edited and get transferred back to the PLCcore-9G20. Downloading files onto the PLCcore-9G20 is also necessary to **update the PLC firmware**. (Advice: The update of *PLC firmware* is not identical with the update of the *PLC user program*. The PLC program is directly transferred to the module from the *OpenPCS* programming environment. No additional software is needed for that.)

Suitable as FTP client would be *"WinSCP"* which is available as Open Source (download from: http://winscp.net). It only consists of one EXE file that needs no installation and can be booted immediately. Furthermore, freeware *"Core FTP LE"* (downloadable from: http://www.coreftp.com) or *"Total Commander"* (integrated in the file manager) are suitable as FTP client.

**TFTP server** The TFTP server is necessary to update the Linux-Image on the PLCcore-9G20. Freeware *"TFTPD32"* (download from: http://tftpd32.jounin.net) is suitable as TFTP server. It only consists of one EXE file that needs no installation and can be booted immediately.

For programs that communicate via Ethernet interface, such as FTP client or TFTP server, it must be paid attention to that rights in the Windows-Firewall are released. Usually Firewalls signal when a program seeks access to the network and asks if this access should be permitted or denied. In this case access is to be permitted.

## 7.2 Activation/Deactivation of Linux Autostart

During standard operation mode, the bootloader "U-Boot" automatically starts the Linux operating system of the module after Reset (or Power-on). Afterwards, the operating system loads all further software components and controls the PLC program execution (see section 6.1). For service purposes, such as configuring the Ethernet interface (see section 7.3) or updating the Linux-Image (see section 7.13.2), it is necessary to disable this Autostart mode and to switch to "U-Boot" command prompt instead (configuration mode).

The automatic boot of Linux operating system is connected with the **simultaneous compliance** with various conditions ("AND relation"). Consequently, for disabling Linux Autostart, it is sufficient to simply **not comply** with one of the conditions.

Table 13 lists up all conditions that are verified by the bootloader “U-Boot”. All of them must be complied with to start an Autostart for the Linux-Image.

*Table 13: Conditions for booting Linux*

| No. | Condition | Remark |
|---|---|---|
| 1 | DIP1 of PLCcore-9G20 = "Off"<br><br>AND<br><br>Connection "/BOOT" = High (pushbutton S406 on the Development Board **not** pressed) | DIP-Switch 1 on the PLCcore-9G20 and module connection "/BOOT" are electrically connected in parallel. Only if both elements are not active (DIP switch 1 open, module connection "/BOOT" not active), the Signal "/BOOT" is at H-level for PLCcore-9G20 and releases the Linux Autostart.<br>The position of DIP-Switch 1 on the PLCcore-9G20 is shown in Figure 7, the position of connection "/BOOT" on the module pin connector is defined in the Hardware Manual ECUcore-9G20 (Manual no.: L-1255). |
| 2 | **No** abort of Autostart via COM0 of the PLCcore-9G20 | If the conditions above are met, "U-Boot" checks the serial interface COM0 of the PLCcore-9G20 for about 1 second after Reset regarding the reception of SPACE signals (ASCII 20H). If such a signal is received within that time, "U-Boot" will disable the Linux Autostart and will activate its own command prompt instead. |

**According to Table 13, the Linux boot is disabled after Reset (e.g. pushbutton S405 on the Development Board) and the "U-Boot" command prompt is activated instead if the following conditions occur:**

**(1) DIP1 = "On" or /BOOT = "Low"** — DIP1: see Figure 7, /BOOT: see Manual L-1255, Development Board: "/BOOT" = pushbutton S406 DIP1 and "/BOOT" are connected in parallel

**- OR -**

**(2) Reception of a SPACE signal (ASCII 20H) within 1 second after Reset**

After activating the Reset pushbutton (e.g. pushbutton S405 on the Development Board), the "U-Boot" command prompt answers.

Figure 7 shows the positioning and meaning of DIP-Switch 1 on the PLCcore-9G20. This DIP-Switch might be hard to access if the module is built in. Thus, the Portpin of the processor that is connected to the switch is available as connection "/BOOT" on the pin connector of the PLCcore-9G20 (see Table 5).

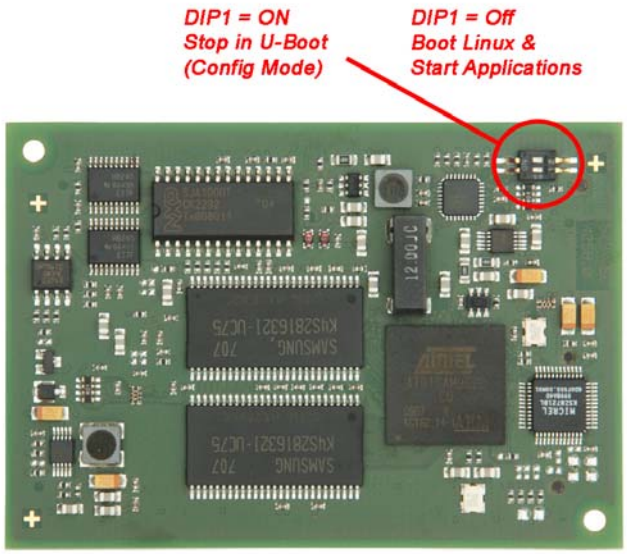



*Figure 7: Positioning and meaning of DIP-Switch 1 on the PLCcore-9G20*

Communicating with the bootloader "U-Boot" only takes place via the serial interface COM0 of the PLCcore-9G20. As receiver on the computer one of the terminal programs must be started (e.g. HyperTerminal or TeraTerm, see section 7.1) and must be configured as follows (see Figure 8):

- 115200 Baud
- 8 Data bit
- 1 Stop bit
- no parity
- no flow control

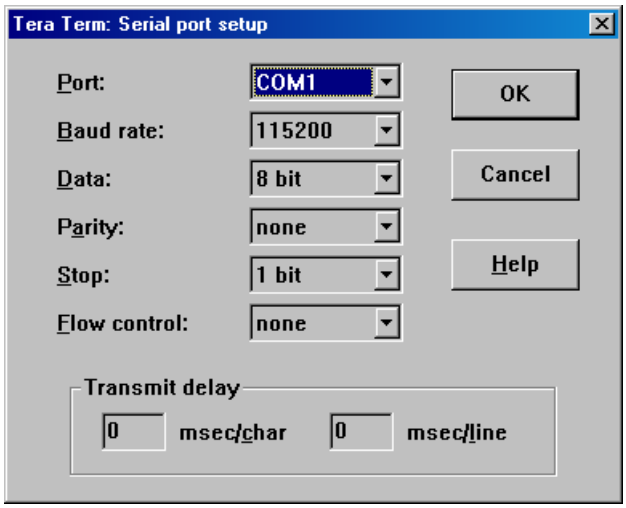



*Figure 8: Terminal configuration using the example of "TeraTerm"*

## 7.3 Ethernet configuration of the PLCcore-9G20

The main Ethernet configuration of the PLCcore-9G20 takes place within the bootloader "U-Boot" and is taken on for all software components (Linux, PLC firmware, HTTP server etc.). The Ethernet configuration is carried out via the serial interface COM0. **Therefore, the "U-Boot" command prompt must be activated as described in section 7.2**. Table 14 lists up "U-Boot" commands necessary for the Ethernet configuration of the PLCcore-9G20.

*Table 14: "U-Boot" configuration commands of the PLCcore-9G20*

| Configuration | Command | Remark |
|---|---|---|
| MAC address | setenv ethaddr <xx:xx:xx:xx:xx:xx> | The MAC address worldwide is a clear identification oft he module and is assigned by the producer. **It should not be modified by the user.** |
| IP address | setenv ipaddr <xxx.xxx.xxx.xxx> | This command sets the local IP address of the PLCcore-9G20. The IP address is to be defined by the network administrator. |
| Network mask | setenv netmask <xxx.xxx.xxx.xxx> | This command sets the network mask of the PLCcore-9G20. The network mask is to be defined by the network administrator. |
| Gateway address | setenv gatewayip <xxx.xxx.xxx.xxx> | This command defines the IP address of the gateway which is to be used by the PLCcore-9G20. The gateway address is set by the network administrator.<br><br>**Advice:** If PLCcore-9G20 and Programming PC are located within the same sub-net, defining the gateway address may be skipped and value "0.0.0.0" may be used instead. |
| Saving the configuration | saveenv | This command saves active configurations in the flash of the PLCcore-9G20. |

Modified configurations may be verified again by entering "*printenv*" in the "U-Boot" command prompt. Active configurations are permanently saved in the Flash of the PLCcore-9G20 by command

***saveenv***

Modifications are adopted upon next Reset of the PLCcore-9G20.

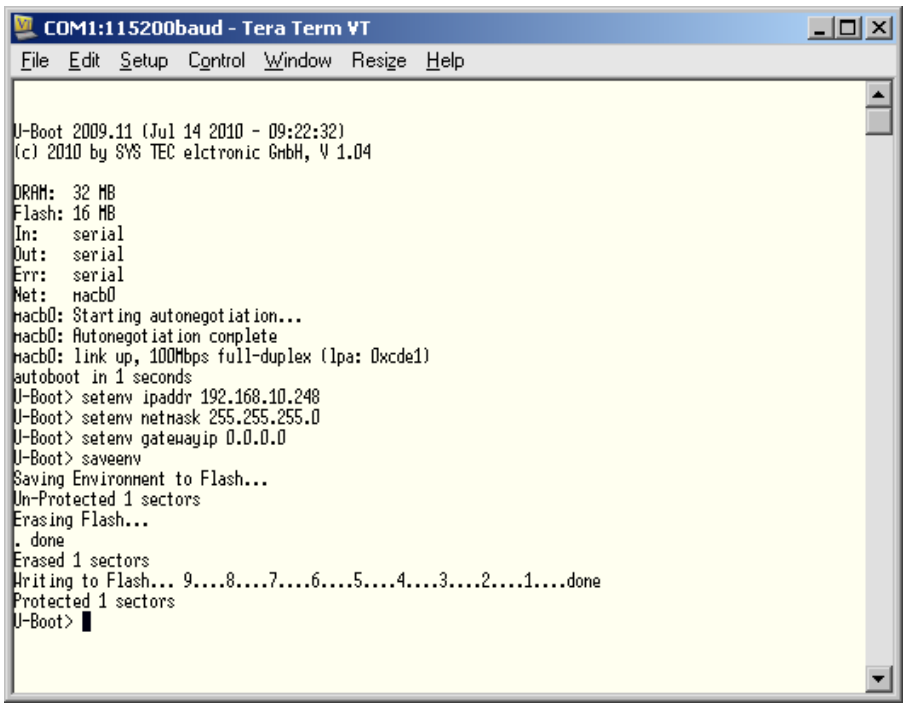



*Figure 9: Ethernet configuration of the PLCcore-9G20*

**After the configuration is finished and according to section 7.2, all conditions for a Linux Autostart must be re-established.**

Upon Reset (e.g. pushbutton S405 on the Development Board) the module starts using the active configurations.

**Advice:** After the configuration is finished, the serial connection between PC and PLCcore-9G20 is no longer necessary.

## 7.4 PLC configuration of the PLCcore-9G20

### 7.4.1 PLC configuration via WEB-Frontend

After finishing the Ethernet configuration (see section 7.3), all further adjustments can take place via the integrated WEB-Frontend of the PLCcore-9G20. For the application of the PLCcore-9G20 using the Development Kit, basic configurations may also be set via local control elements (see section 7.4.2).

To configure the PLCcore-9G20 via WEB-Frontend it needs a WEB-Browser on the PC (e.g. Microsoft Internet Explorer, Mozilla Firefox etc.). To call the configuration page, prefix *"http://"* must be entered into the address bar of the WEB-Browser prior to entering the IP address of the PLCcore-9G20 as set

in section 7.2, e.g. *"http://192.168.10.248"*. Figure 10 exemplifies calling the PLCcore-9G20 configuration page in the WEB-Browser.

The standard setting (factory setting) requires a user login to configure the PLCcore-9G20 via WEB-Frontend. This is to prevent unauthorized access. Therefore, user name and password must be entered (see Figure 10). On delivery of the module, the following user account is preconfigured (see section 7.7):

User: PlcAdmin
Password: Plc123

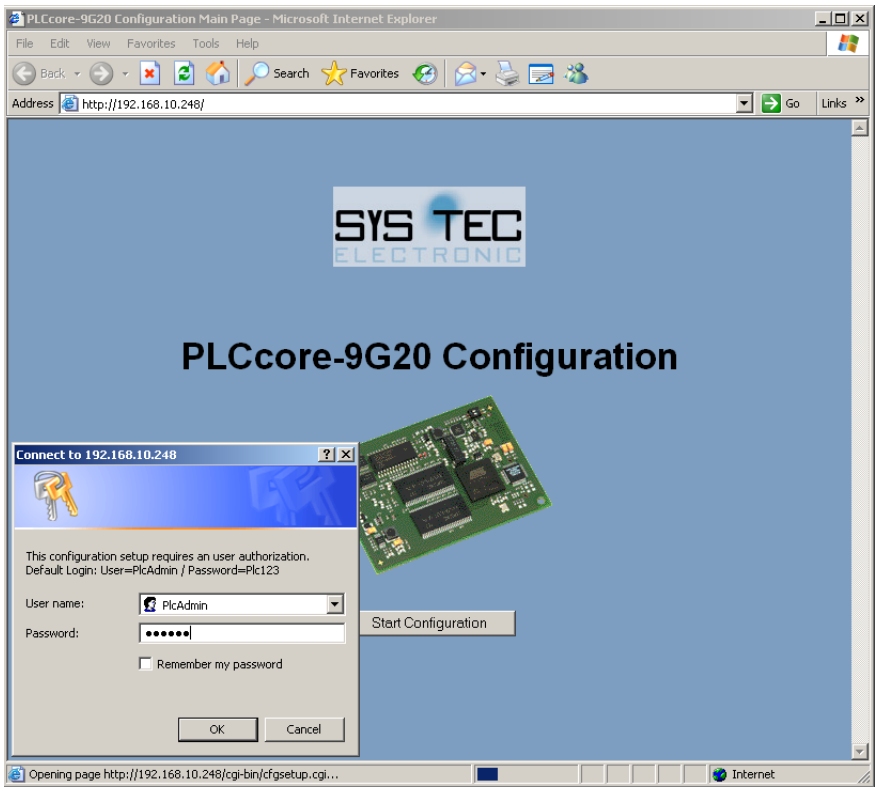



*Figure 10: User login dialog of the WEB-Frontend*

All configuration adjustments for the PLCcore-9G20 are based on dialogs. They are adopted into the file ***"/home/plc/plccore-9g20.cfg"*** of the PLCcore-9G20 by activating the pushbutton "Save Configuration" (also compare section 7.4.3). After activating Reset (e.g. pushbutton S405 on the Development Board), the PLCcore-9G20 starts automatically using the active configuration. Figure 11 shows the configuration of the PLCcore-9G20 via WEB-Frontend.

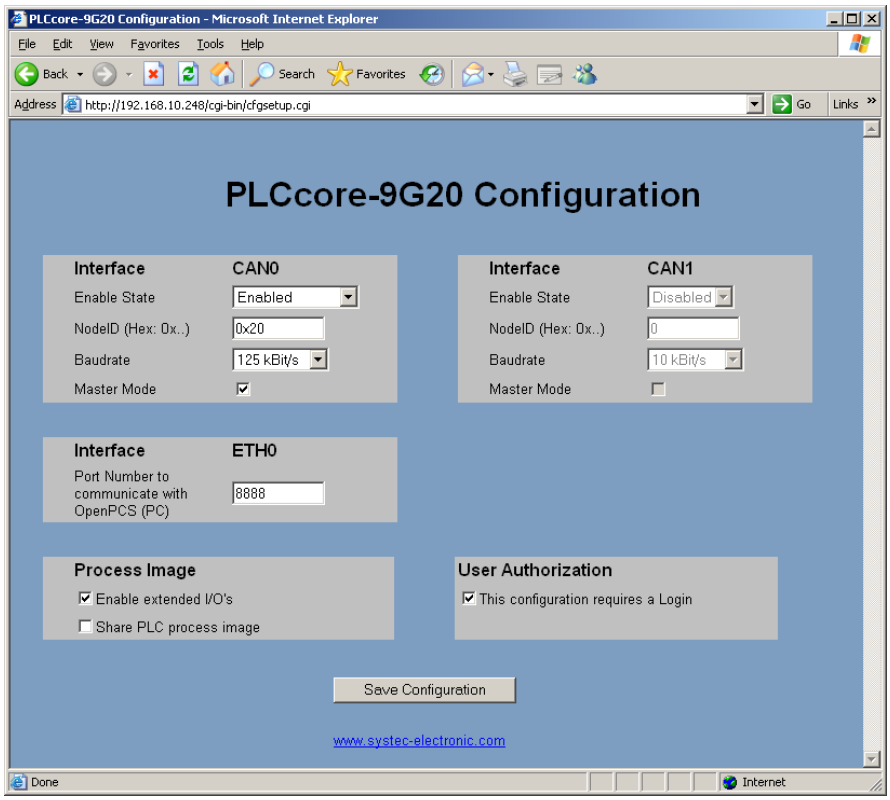



*Figure 11: PLC configuration via WEB-Frontend*

If "DIP/Hex-Switch" is chosen as Enable State of Interface CAN0, the configuration of this interface takes place via local control elements of the Development Kit PLCcore-9G20 (see section 7.4.2).

The standard setting (factory setting) of the PLCcore-9G20 requires a user login to access the WEB-Frontend. Therefore, only the user name indicated in configuration file ***"/home/plc/plccore-9g20.cfg"*** is valid (entry *"User="* in section *"[Login]"*, see section 7.4.3). Procedures to modify the user login password are described in section 7.10. To allow module configuration to another user, an appropriate user account is to be opened as described in section 7.9. Afterwards, the new user name must be entered into the configuration file ***"/home/plc/plccore-9g20.cfg"***. Limiting the user login to one user account is cancelled by deleting the entry *"User="* in section *"[Login]"* (see 7.4.3). Thus, any user account may be used to configure the module. By deactivating control box *"This configuration requires a Login"* in the field *"User Authorization"* of the configuration page (see Figure 11) free access to the module configuration is made available without previous user login.

### 7.4.2 PLC configuration via control elements of the Development Kit PLCcore-9G20

The PLC configuration via control elements is not supported by the currently available Development Board with the PCB version 4261.2. This feature is reserved to a later hardware version of the Development Board.

The PLC configuration is possible either by the WEB-Frontend (see section 7.4.1) or by directly editing the file ***"/home/plc/plccore-9g20.cfg"*** (see section 7.4.3).

### 7.4.3 Setup of the configuration file "plccore-9g20.cfg"

The configuration file ***"/home/plc/plccore-9g20.cfg"*** allows for comprehensive configuration of the PLCcore-9G20. Although, working in it manually does not always make sense, because most of the

adjustments may easily be edited via WEB-Frontend (compare section 7.4.1). The setup of the configuration file is similar to the file format "Windows INI-File". It is divided into "*[Sections]*" which include different entries "*Entry=*". Table 16 shows all configuration entries. Entries of section "*[CAN0]*" take priority over settings via control elements (see section 7.4.2).

*Table 16: Configuration entries of the CFG file*

| Section | Entry | Value | Meaning |
|---|---|---|---|
| [CAN0] | Enabled | -1, 0, 1 | -1: Interface CAN0 is activated, configuration takes place via control elements of the Development Board (factory setting, see section 7.4.2)<br>0: Interface CAN0 is deactivated<br>1: Interface CAN0 is activated, configuration takes place via entries of the configuration file below |
| | NodeID | 1 … 127 or<br>0x01 … 0x7F | Node number for interface CAN0<br>(decimal or hexadecimal with prefix "0x") |
| | Baudrate | 10, 20, 50, 125, 250, 500, 800, 1000 | Bitrate for interface CAN0 |
| | MasterMode | 0, 1 | 1: Master mode is activated<br>0: Master mode is deactivated |
| [CAN1] | Enabled | | By default, section "[CAN1]" is not evaluated, but if necessary it enables the extension of PLCcore-9G20 by an additional CAN interface (see section 6.9.2). |
| | NodeID | | |
| | Baudrate | | |
| | MasterMode | | |
| [ETH0] | PortNum | Default Port no: 8888 | Port number for the communication with the Programming-PC and for program download (only for PLCcore-9G20/Z5, order number 3390025) |
| [ProcImg] | EnableExtIo | 0, 1 | 0: Only on-board I/Os of the PLCcore-9G20 are used for the process image (except Temperature Sensor)<br>1: All I/Os supported by driver are used for the process image (incl. Temperature Sensor and external ADC of Developmentboard)<br>(for adaptation of process image see section 8.2) |
| | EnableSharing | 0, 1 | 0: No sharing of process image<br>1: Sharing of process image is enabled<br>(see section 8.1) |

| | | | |
|---|---|---|---|
| [Login] | Authorization | 0, 1 | 0: Configuration via WEB-Frontend is possible without user login<br>1: Configuration via WEB-Frontend requires user login |
| | User | Default Name: PlcAdmin | If entry *"User="* is available, only the user name defined is accepted for the login to configure via WEB-Frontend.<br>If the entry is not available, any user registered on the PLCcore-9G20 (see section 7.9) may login via WEB-Frontend. |

The configuration file ***"/home/plc/plccore-9g20.cfg"*** includes the following factory settings:

```
[Login]
Authorization=1
User=PlcAdmin

[CAN0]
Enabled=1
NodeID=0x20
Baudrate=125
MasterMode=1

[CAN1]
Enabled=0
NodeID=0
Baudrate=0
MasterMode=0

[ETH0]
PortNum=8888

[ProcImg]
EnableExtIo=1
EnableSharing=0
```

## 7.5 Boot configuration of the PLCcore-9G20

The PLCcore-9G20 is configured so that after Reset the PLC firmware starts automatically. Therefore, all necessary commands are provided by the start script ***"/home/etc/autostart"***. Hence, the required environment variables are set and drivers are booted.

If required, the start script ***"/home/etc/autostart"*** may be complemented by further entries. For example, by entering command *"pureftp"*, the FTP server is called automatically when the PLCcore-9G20 is booted. The script can be edited directly on the PLCcore-9G20 in the FTP client *"WinSCP"* (compare section 7.1) using pushbutton *"F4"* or *"F4 Edit"*.

## 7.6 Selecting the appropriate firmware version

The PLCcore-9G20 is delivered with different firmware versions. Those vary in the communication protocol for the data exchange with the programming PC and they differ from each other regarding the availability of FB communication classes (see section 6.3). The selection of the appropriate firmware version takes place in the start script ***"/home/etc/autostart"***. By default, the *"BoardID"* of the module

as set in the bootloader "U-Boot" is analyzed. Table 17 lists up the assignments of firmware versions and BoardIDs.

*Table 17: Assignment of BoardIDs and firmware versions for the PLCcore-9G20*

| BoardID | Firmware Version | Remark |
|---|---|---|
| 1008004 | plccore-9g20-z4 | **PLCcore-9G20/Z4 (CANopen)** communication with the programming PC via CANopen protocol (Interface CAN0) |
| 1008005 | plccore-9g20-z5 | **PLCcore-9G20/Z5 (Ethernet)** communication with the programming PC via UDP protocol (Interface ETH0) |

The configuration of BoardIDs takes place via the serial interface COM0. **Therefore, the "U-Boot" command prompt must be activated as described in section 7.2.** Setting BoardIDs is carried out via the "U-Boot" command *"set boardid"* by entering the corresponding number listed in Table 17, e.g.:

```
setenv  boardid  1008005
```

The modified setting can be verified by entering "*printenv*" at the "U-Boot" command prompt. Command

***saveenv***

persistently saves the current selection in the Flash of the PLCcore-9G20. Figure 12 visualizes the configuration of the BoardID.

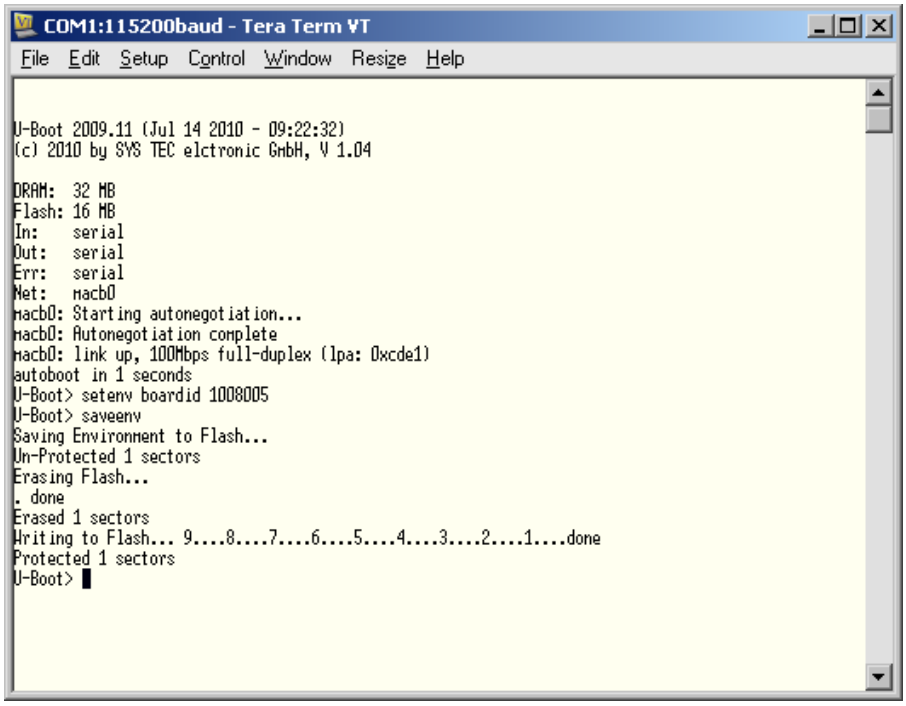



*Figure 12: Selecting the appropriate firmware version for the PLCcore-9G20*

**After completing the configuration, all preconditions for a Linux Autostart must be reestablished according to section 7.2.**

Alternatively, the appropriate firmware version may be selected directly in the start script ***"/home/etc/autostart"***. Therefore, delete part "Select PLC Type" and insert the appropriate firmware instead, e.g.:

```
PLC_FIRMWARE=$PLC_DIR/plccore-9g20-z5
```

## 7.7 Predefined user accounts

All user accounts listed in Table 18 are predefined upon delivery of the PLCcore-9G20. Those allow for a login to the command shell (serial RS232 connection or Telnet) and at the FTP server of the PLCcore-9G20.

*Table 18: Predefined user accounts of the PLCcore-9G20*

| User name | Password | Remark |
|---|---|---|
| PlcAdmin | Plc123 | Predefined user account for the administration of the PLCcore-9G20 (configuration, user administration, software updates etc.) |
| root | Sys123 | Main user account ("root") of the PLCcore-9G20 |

## 7.8 Login to the PLCcore-9G20

### 7.8.1 Login to the command shell

In some cases the administration of the PLCcore-9G20 requires the entry of Linux commands in the command shell. Therefore, the user must be directly logged in at the module. There are two different possibilities:

- Logging in is possible with the help of a **Terminal program** (e.g. HyperTerminal or TeraTerm, see section 7.1) via the serial interface **COM0** of the PLCcore-9G20 – analog to the procedure described for the Ethernet configuration in section 7.2. **For the configuration of the terminal settings pay attention to only use "CR" (carriage return) as end-of-line character.** Login with user name and password is not possible for "CR+LF" (carriage return + line feed)!

- Alternatively, the login is possible using a **Telnet client** (e.g. Telnet or also TeraTerm) via the Ethernet interface **ETH0** of the PLCcore-9G20.

For logging in to the PLCcore-9G20 via the Windows standard Telnet client, the command "*telnet*" must be called by using the IP address provided in section 7.2, e.g.

```
telnet  192.168.10.248
```

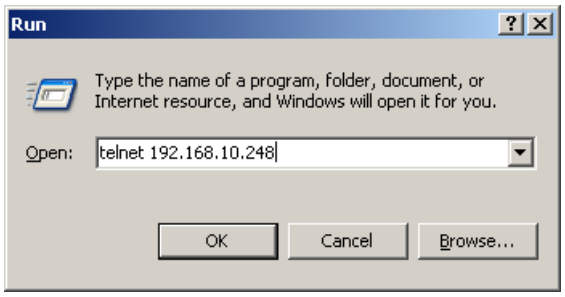



*Figure 13: Calling the Telnet client in Windows*

Logging in to the PLCcore-9G20 is possible in the Terminal window (if connected via COM0) or in the Telnet window (if connected via ETH0). The following user account is preconfigured for the administration of the module upon delivery of the PLCcore-9G20 (also compare section 7.7):

User: *PlcAdmin*
Password: *Plc123*

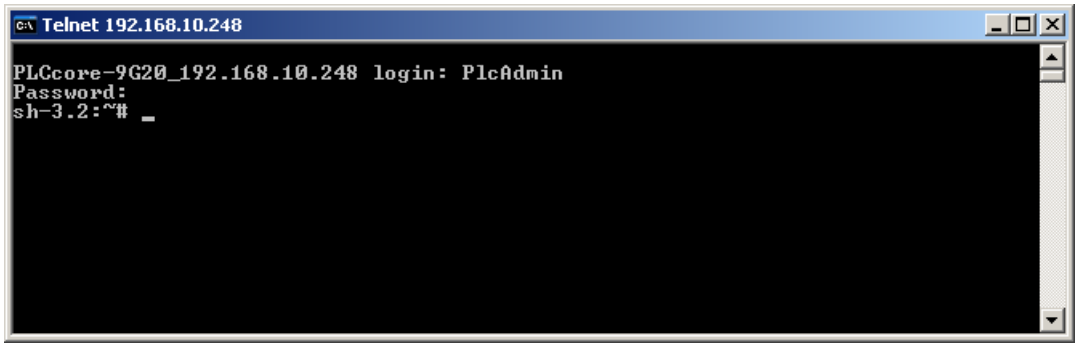



*Figure 14: Login to the PLCcore-9G20*

Figure 14 exemplifies the login to the PLCcore-9G20 using a Windows standard Telnet client.

### 7.8.2 Login to the FTP server

The PLCcore-9G20 has available a FTP server (FTP Daemon) that allows file exchange with any computer (up- and download of files). Due to security and performance reasons, the FTP server is deactivated by default and must be started manually if required. Therefore, the user must first be logged in to the command shell of the PLCcore-9G20 following the procedures described in section 7.8.1. Afterwards, the following command must be entered in the Telnet or Terminal window:

```
pureftp
```

Figure 15 illustrates an example for starting the FTP server.

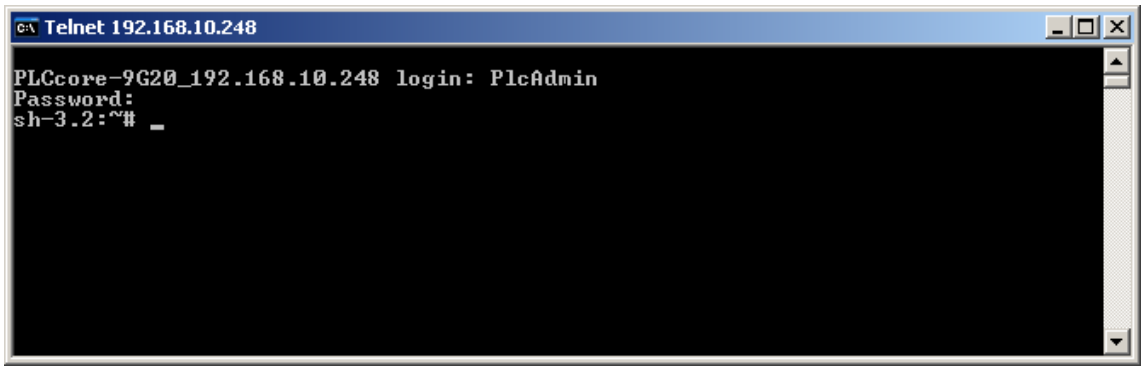



*Figure 15: Starting the FTP server*

**Advice:** By entering command *"pureftp"* in the start script ***"/home/etc/autostart"***, the FTP server may be called automatically upon boot of the PLCcore-9G20 (see section 7.5).

*"WinSCP"* - which is available as open source - would be suitable as FTP client for the computer (see section 7.1). It consists of only one EXE file, needs no installation and may be started immediately. After program start, dialog *"WinSCP Login"* appears (see Figure 16) and must be adjusted according to the following configurations:

File protocol: FTP
Host name: IP address for the PLCcore-9G20 as set in section 7.3
User name: *PlcAdmin* (for predefined user account, see section 7.7)
Password: *Plc123* (for predefined user account, see section 7.7)

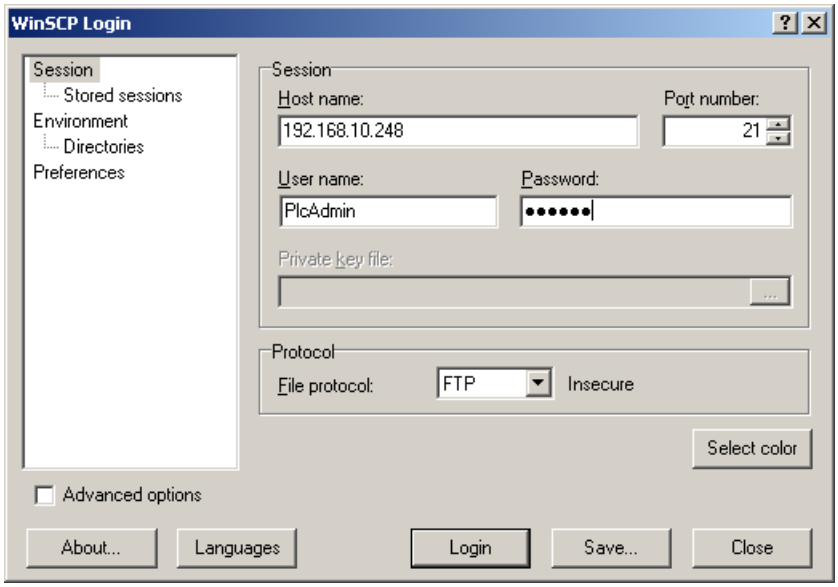



*Figure 16: Login settings for WinSCP*

After using pushbutton *"Login"*, the FTP client logs in to the PLCcore-9G20 and lists up the active content of directory *"/home"* in the right window. Figure 17 shows FTP client *"WinSCP"* after successful login to the PLCcore-9G20.

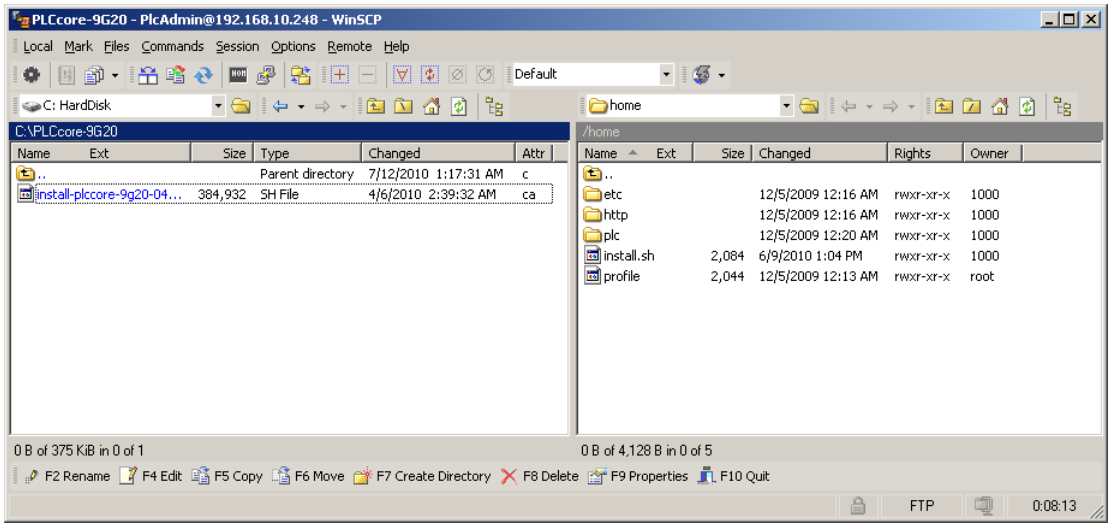



*Figure 17: FTP client for Windows "WinSCP"*

After successful login, configuration files on the PLCcore-9G20 may be edited by using pushbuttons *"F4"* or *"F4 Edit"* within the FTP client *"WinSCP"* (select transfer mode *"Text"*). With the help of pushbutton *"F5"* or *"F5 Copy"*, files may be transferred between the computer and the PLCcore-9G20, e.g. for data backups of the PLCcore-9G20 or to transfer installation files for firmware updates (select transfer mode *"Binary"*).

## 7.9 Adding and deleting user accounts

Adding and deleting user accounts requires the login to the PLCcore-9G20 as described in section 7.8.1.

**Adding** a new user account takes place via Linux command *"adduser"*. In embedded systems such as the PLCcore-9G20, it does not make sense to open a directory for every user. Hence, parameter *"-H"* disables the opening of new directories. By using parameter *"-h /home"* instead, the given directory *"/home"* is rather assigned to the new user. To open a new user account on the PLCcore-9G20, Linux command *"adduser"* is to be used as follows:

```
adduser -h /home -H -G <group> <username>
```

Figure 18 exemplifies adding a new account on the PLCcore-9G20 for user *"admin2"*.

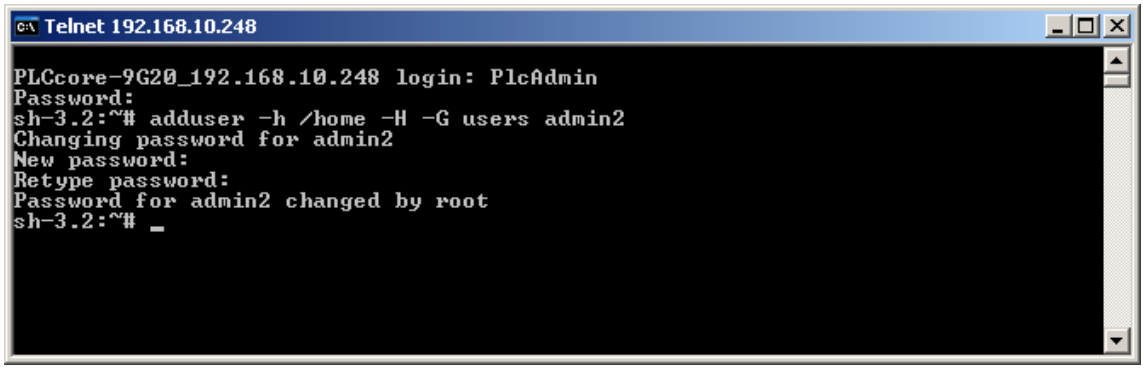



*Figure 18: Adding a new user account*

| Advice: | **If the new user account shall be used to access WEB-Frontend, the user name must be entered into the configuration file *"plccore-9g20.cfg"* (for details about logging in to WEB-Frontend please compare section 7.4.1 and 7.4.3).** |
|---|---|

To **delete** an existing user account from the PLCcore-9G20, Linux command *"deluser"* plus the respective user name must be used:

```
deluser <username>
```

## 7.10 How to change the password for user accounts

Changing the password for user accounts requires login to the PLCcore-9G20 as described in section 7.8.1.

To change the password for an existing user account on the PLCcore-9G20, Linux command *"passwd"* plus the respective user name must be entered:

```
passwd <username>
```

Figure 19 exemplifies the password change for user *"PlcAdmin"*.

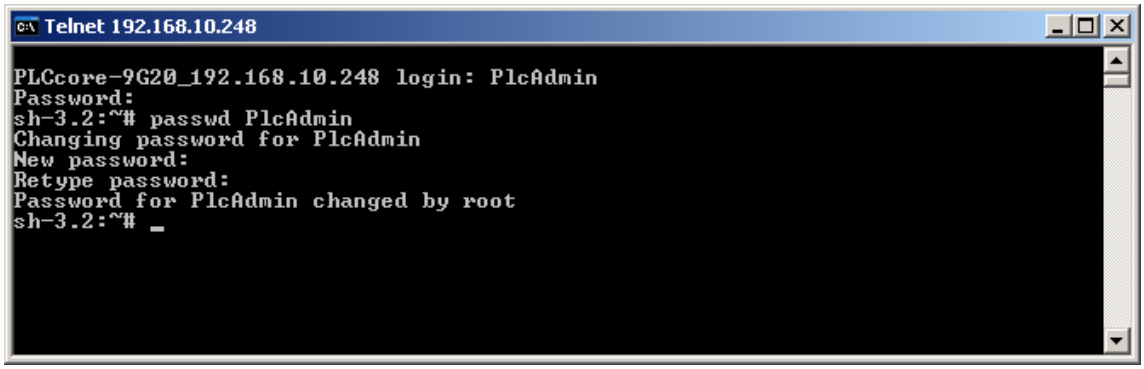



*Figure 19: Changing the password for an user account*

## 7.11 Setting the system time

Setting the system time requires login to the PLCcore-9G20 as described in section 7.8.1.

There are two steps for setting the system time of the PLCcore-9G20. At first, the current date and time must be set using Linux command *"date"*. Afterwards, by using Linux command *"hwclock -w"* the system time is taken over into RTC module of the PLCcore-9G20.

Linux command *"date"* is structured as follows:

```
date [options] [YYYY.]MM.DD-hh:mm[:ss]
```

**Example:**

```
date   2010.02.25-11:34:55
          |  |  |  |  |  |
          |  |  |  |  |  +--- Second
          |  |  |  |  +------ Minute
          |  |  |  +--------- Hour
          |  |  +------------ Day
          |  +--------------- Month
          +------------------ Year
```

To set the system time of the PLCcore-9G20 to 2010/02/25 and 11:34:55 (as shown in the example above), the following commands are necessary:

```
date  2010.02.25-11:34:55
hwclock  -w
```

The current system time is displayed by entering Linux command *"date"* (without parameter). The Linux command *"hwclock -r"* can be used to recall current values from the RTC. By using *"hwclock -s"*, the current values of the RTC are taken over as system time for Linux (synchronizing the kernel with the RTC). Figure 20 exemplifies setting and displaying the system time.

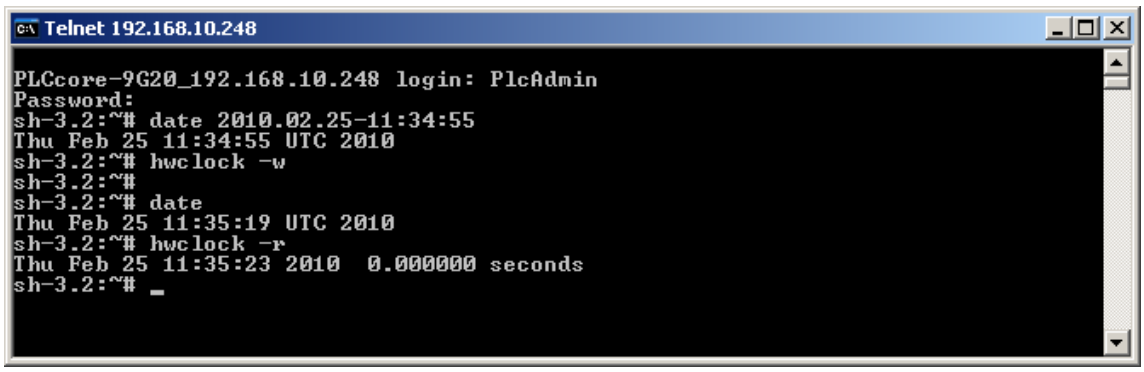



*Figure 20: Setting and displaying the system time*

Upon start of the PLCcore-9G20, date and time are taken over from the RTC and set as current system time of the module. Therefore, Linux command *"hwclock -s"* is necessary which is included in start script ***"/etc/init.d/hwclock"***.

## 7.12 File system of the PLCcore-9G20

The Embedded Linux which is pre-installed on PLCcore-9G20 provides most of the on-board memory as flash disk. The JFFS2 is used as file system. It is a well-known system in the embedded field and was specifically developed as data carrier for the usage of flash modules. The on-board flash of PLCcore-9G20 is separated into two independent JFFS2 partitions. The first partition contains the Linux kernel and the Root file system. The second partition is completely mounted as directory *"/home"*. There is a complete read and write access for both partitions.

The partition mounted to the path **"/home"** is used to store all files modifiable and updatable by the user, e.g. configuration files, PLC firmware and PLC program files that have been loaded onto the module. Directory **"/tmp"** is appropriately sized to function as temporary buffer for FTP downloads of firmware archives for PLC software updates (see section 7.13.1).

*Table 19: File system configuration of the PLCcore-9G20*

| Path | Size | Description |
|---|---|---|
| /home | 4096 kByte | Flash disk to permanently store files modifiable and updatable by the user (e.g. configuration files, PLC firmware, PLC program), data preservation in case of power breakdown |
| /tmp | 8192 kByte | RAM disk, suitable as intermediate buffer for FTP downloads, but no data preservation in case of power breakdown |
| /var | 1024 kByte | RAM disk which is used by the system to store temporary files, no data preservation in case of power breakdown |
| /mnt | | Target for integrating remote directories, it is not part of the PLCcore-9G20 standard functionality |

Sizes of file system paths that are configured or still available can be identified by using the Linux command "df" ("DiskFree") – see Figure 21.

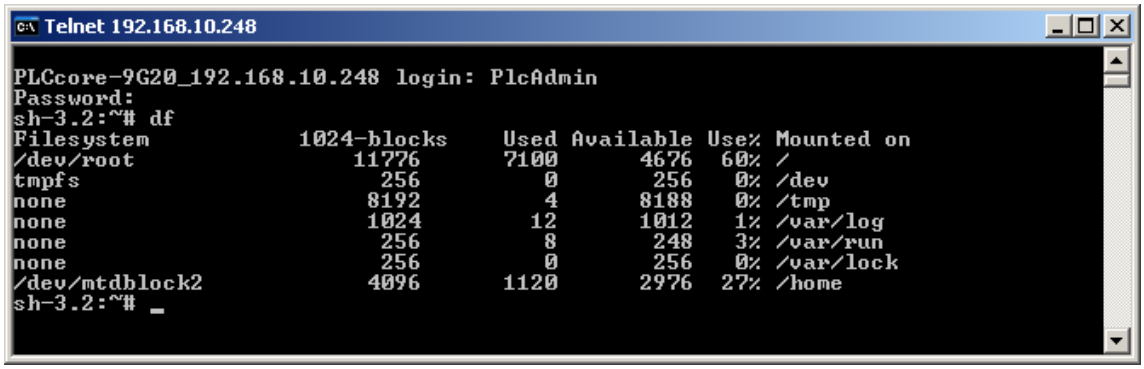



*Figure 21: Display of information about the file system*

Particular information about the system login and handling the Linux command shell of the PLCcore-9G20 is given attention in section 7.8.

## 7.13 Software update of the PLCcore-9G20

All necessary firmware components to run the PLCcore-9G20 are already installed on the module upon delivery. Hence, firmware updates should only be required in exceptional cases, e.g. to input new software that includes new functionality.

### 7.13.1 Updating the PLC firmware

PLC firmware indicates the run time environment of the PLC. ***PLC firmware*** can only be generated and modified by the producer; **it is not identical with the PLC user program** which is created by the PLC user. The PLC user program is directly transferred from the *OpenPCS* programming environment onto the module. No additional software is needed.

Updating the PLC firmware requires login to the command shell of the PLCcore-9G20 as described in section 7.8.1 and login to the FTP server as described in section 7.8.2.

Updating the PLC firmware takes place via a self-extracting firmware archive that is transferred onto the PLCcore-9G20 via FTP. After starting the FTP server on the PLCcore-9G20 (command *"pureftp"*, see section 7.8.2), the respective firmware archive can be transferred into directory *"/tmp"* of the PLCcore-9G20 (see Figure 22).

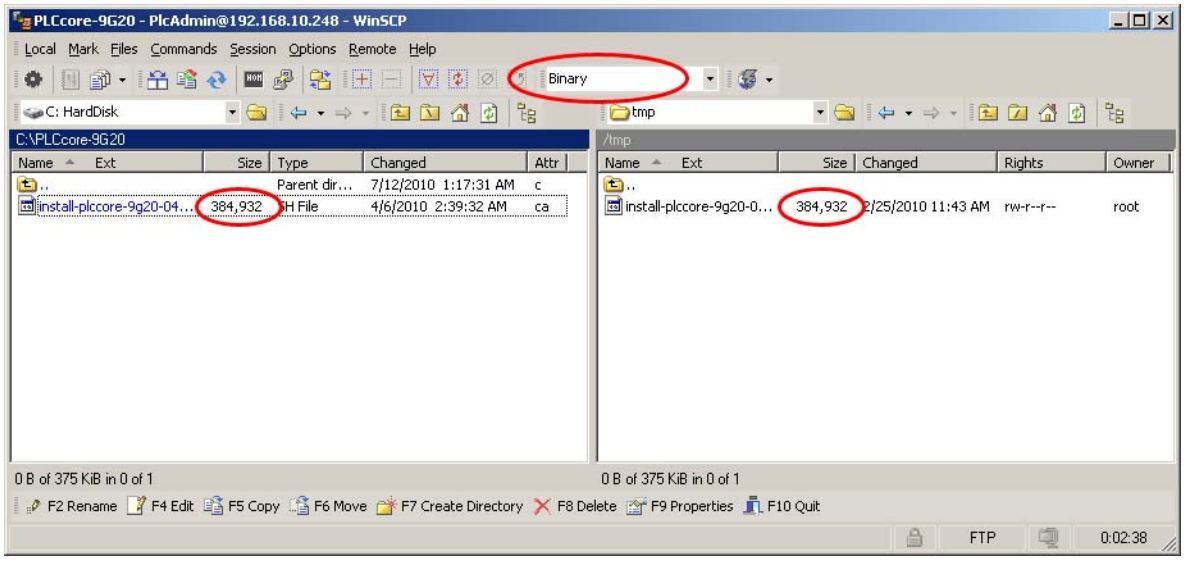



*Figure 22: File transfer in FTP client "WinSCP"*

**Important:** To transfer the firmware archive via FTP, transfer type *"Binary"* must be chosen. If FTP client *"WinSCP"* is used, the appropriate transfer mode is to be chosen from the menu bar. After downloading the firmware archive, it must be checked if the file transferred to the PLCcore-9G20 has the exact same size as the original file on the computer (compare Figure 22). Any differences in that would indicate a mistaken transfer mode (e.g. *"Text"*). In that case the transfer must be repeated using transfer type *"Binary"*.

After downloading the self-extracting archive, the PLC firmware must be installed on the PLCcore-9G20. Therefore, the following commands are to be entered in the Telnet window. It must be considered that the file name for the firmware archive is labeled with a version identifier (e.g. *"install-plccore-9g20-0412_0100.sh"* for version 4.12.01.00). This number must be adjusted when commands are entered:

```
cd  /tmp
chmod  +x  install-plccore-9g20-0412_0100.sh
./install-plccore-9g20-0412_0100.sh
```

**Advice:** The command shell of the PLCcore-9G20 is able to automatically complete names if the Tab key is used ("tab completion"). Hence, it should be sufficient to enter the first letters of each file name and the system will complement it automatically. For example, *"./ins"* is completed to *"./install-plccore-9g20-0412_0100.sh"* if the Tab key is used.

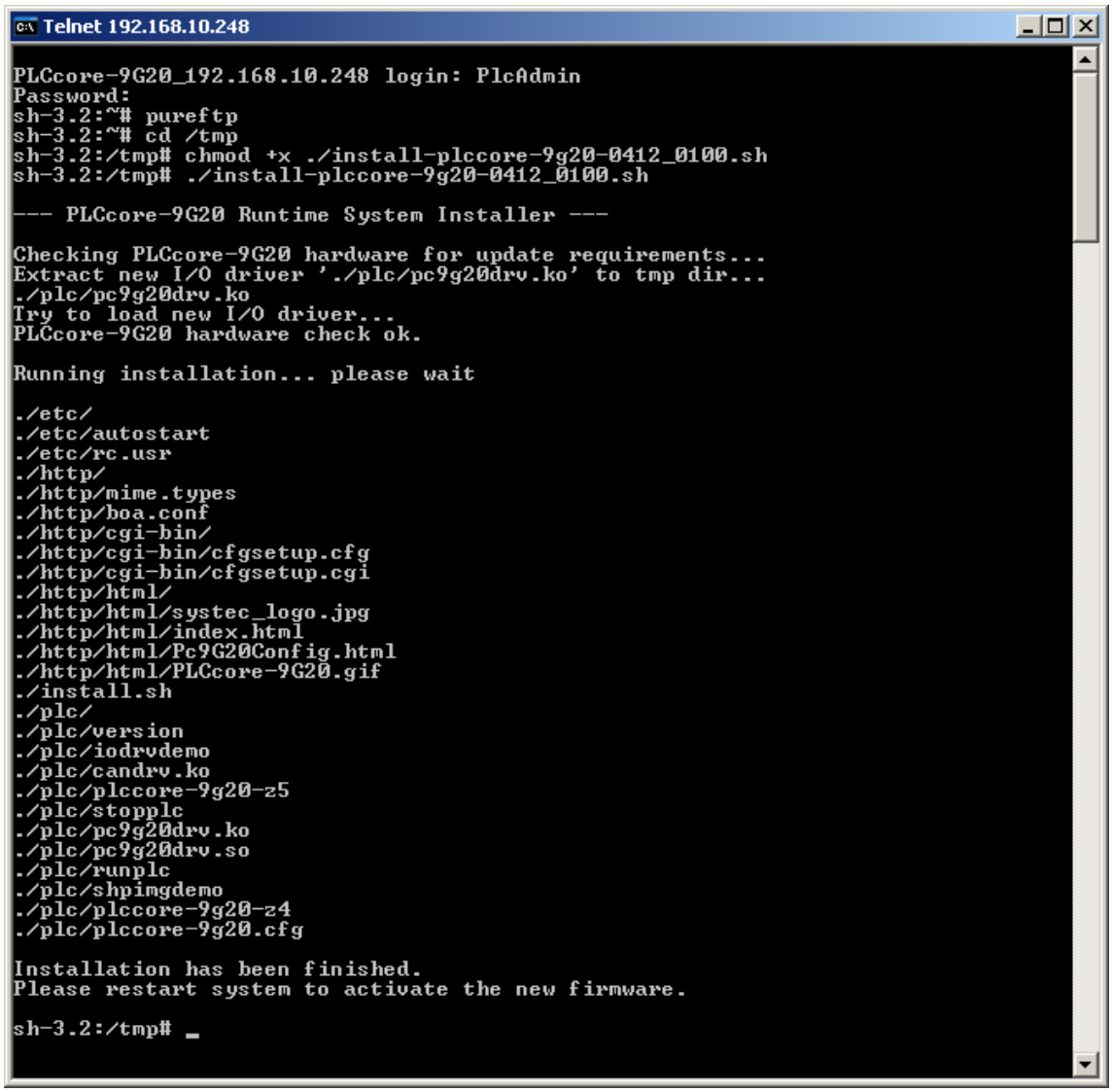



*Figure 23: Installing PLC firmware on the PLCcore-9G20*

Figure 23 exemplifies the installation of PLC firmware on the PLCcore-9G20. After Reset the module is started using the updated firmware.

| | |
|---|---|
| **Advice:** | **If the PLC firmware is updated, the configuration file *"/home/plc/plccore-9g20.cfg"* is overwritten. This results in a reset of the PLC configuration to default settings. Consequently, after an update, the configuration described in section 7.4 should be checked and if necessary it should be reset.** |

### 7.13.2 How to update the Linux-Image

Updating the Linux-Image takes place via TFTP (**T**rivial **FTP**) within Linux bootloader *"U-Boot"*. Therefore, an appropriate TFTP server is necessary on the computer, e.g. freeware *"TFTPD32"* (compare section 7.1). The program consists of only one EXE file that requires no installation and can be run immediately. After the program start, an appropriate working directory ("Current Directory") should be created by clicking on pushbutton *"Browse"* (e.g. *"C:\PLCcore-9G20"*). The Linux-Image for the PLCcore-9G20 must be located in this directory (*"root.sum.jffs2"*).

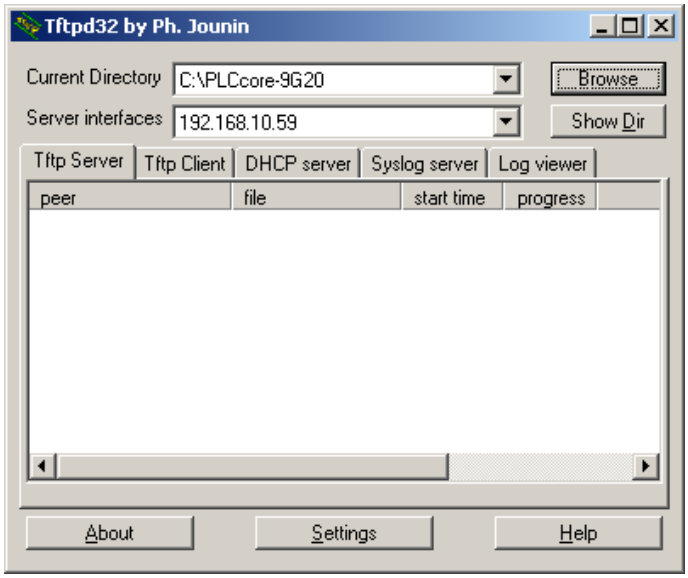



*Figure 24: TFTP server for Windows "TFTPD32"*

A TFTP download of the Linux-Image **requires** that the **Ethernet configuration** of the PLCcore-9G20 is **completed** according to procedures describes in **section 7.3**. To update the Linux-Image it is necessary to have available another serial connection to the PLCcore-9G20 in addition to the Ethernet connection. All configurations for the terminal program as described in section 7.2 apply (115200 Baud, 8 Data bit, 1 Stop bit, no parity and no flow control).

**Updating the Linux-Image of the PLCcore-9G20 is only possible if Linux is not running. Hence, Linux Autostart must be disabled prior to the updating process and "U-Boot" command prompt must be used instead. Procedures are described in section 7.2.**

After Reset (e.g. pushbutton S405 on the Development Board), the "U-Boot" command prompt answers. To update the Linux-Image the following commands must be entered according to the following sequence:

*Table 20: Command sequence to update the Linux-Image on the PLCcore-9G20*

| Command | Meaning |
|---|---|
| *setenv serverip <host_ip_addr>* | Setting the IP address of the TFTP server.<br>If *"TFTPD32"* is used, the address is shown in field "Server Interface" on the PC. |
| *tftp root.sum.jffs2* | Downloading the Linux-Image from the Development PC onto the PLCcore-9G20 |
| *erase nor0,3* | Erase the Flash area, needed by Linux-Image |
| cp.b ${fileaddr} 0x10480000 ${filesize} | Saving the Linux-Image in the Flash of the PLCcore-9G20 |

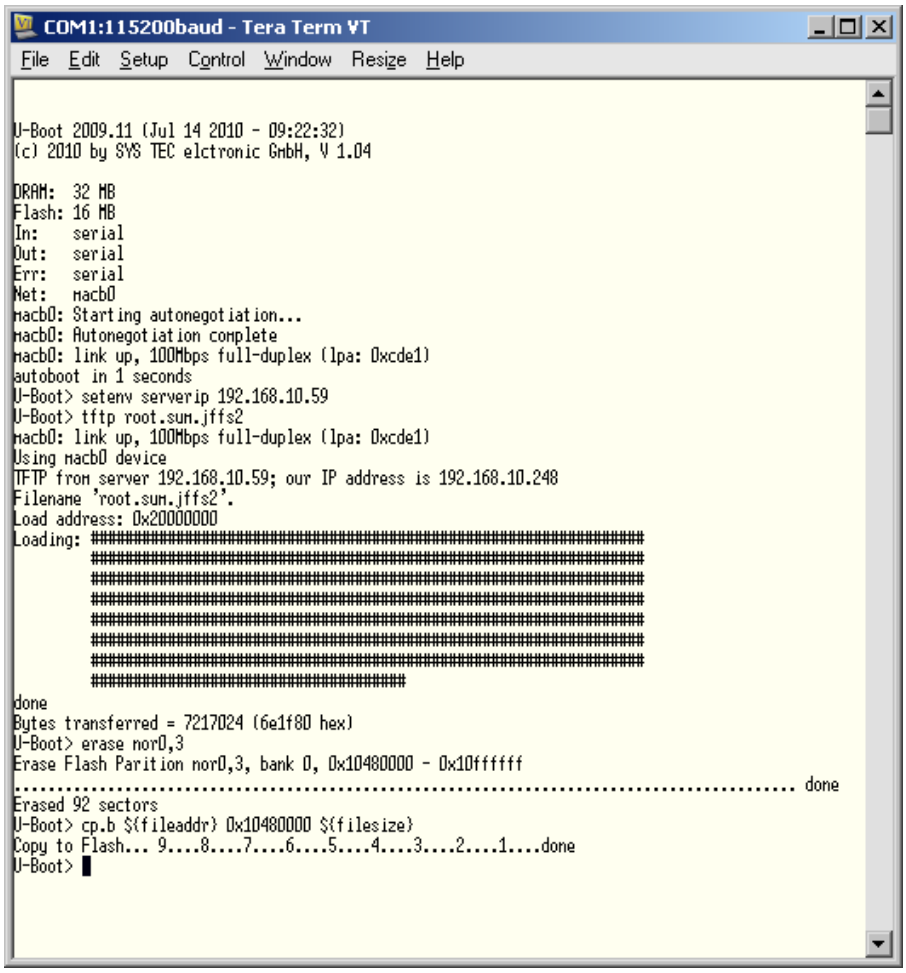



*Figure 25: Downloading the Linux-Image to the PLCcore-9G20*

**After completing the configuration, conditions for a Linux Autostart must be reestablished according to instructions in section 7.2.**

After Reset is activated (e.g. pushbutton S405 on the Development Board), the PLCcore-9G20 starts automatically using the current Linux-Image.

**Advice:** After the configuration is finished, the serial connection between the computer and the PLCcore-9G20 is no longer necessary.

# 8 Adaption of In-/Outputs and Process Image

## 8.1 Data exchange via shared process image

### 8.1.1 Overview of the shared process image

The PLCcore-9G20 is based on the operating system Embedded Linux. Thus, it is possible to execute other user-specific programs simultaneously to running the PLC firmware. The PLC program and a user-specific C/C++ application can exchange data by using the same process image (shared process image). Implementing user-specific C/C++ applications **is based on the Software package SO-1105 ("VMware-Image of the Linux development system for the ECUcore-9G20")**.

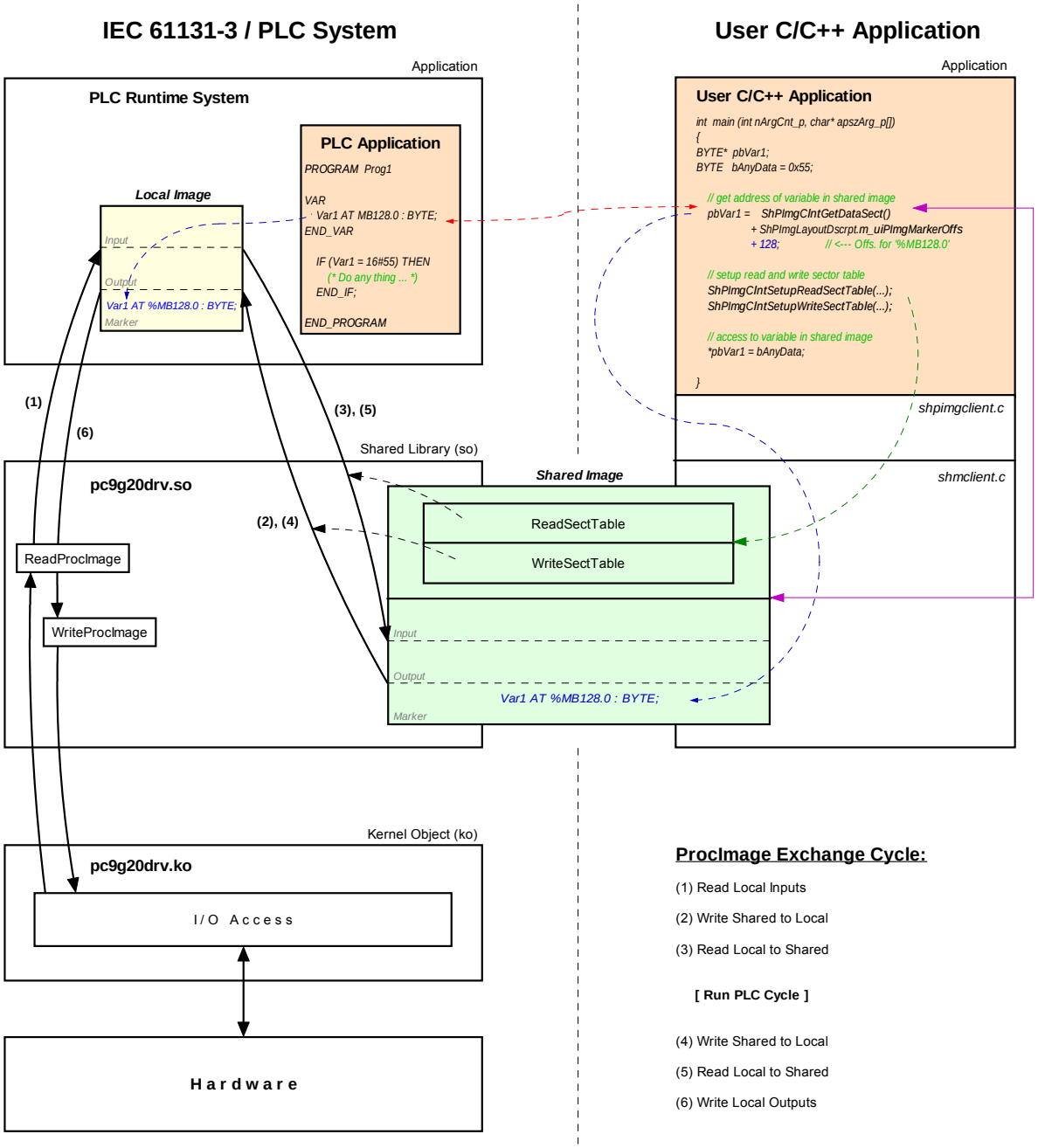



*Figure 26: Overview of the shared process image*

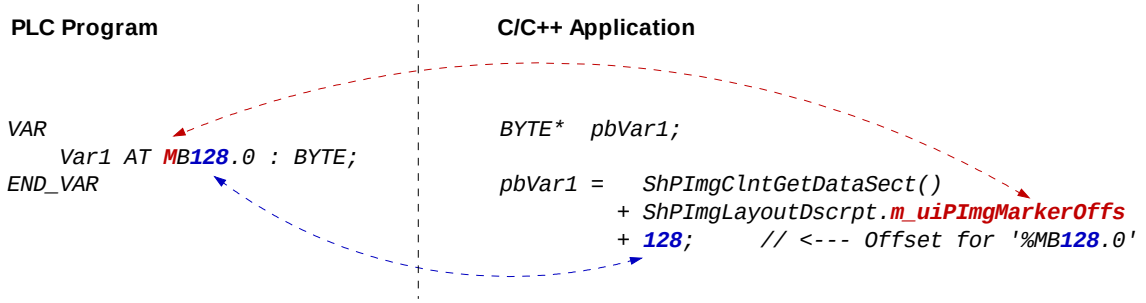
Not all variables are utilizable via the shared process image within a C/C++ application. Only those directly addressed variables that the PLC program generates within the process image. As shown in Figure 26, two separate process images are used for the data exchange with an external application inside of the PLC runtime system. This is necessary to meet the IEC 61131-3 requirement that the initial PLC process image may not be modified during the entire execution of one PLC program cycle. Thereby, the PLC program always operates with the internal process image that is locally generated within the PLC runtime system ("Local Image" in Figure 26). This is integrated within the PLC runtime system and is protected against direct accesses from the outside. On the contrary, the user-specific, external C/C++ application always uses the shared process image ("Shared Image" in Figure 26). This separation of two process images enables isolation between accesses to the PLC program and the external application. Those two in parallel and independently running processes now must only be synchronized for a short period of time to copy the process data.

An activation of **option *"Share PLC process image"*** within the PLC configuration enables data exchange with external applications (see section 7.4.1). Alternatively, entry *"EnableSharing="* can directly be set within section *"[ProcImg]"* of the configuration file ***"/home/plc/plccore-9g20.cfg"*** (see section 7.4.3). The appropriate configuration setting is evaluated upon start of the PLC firmware. By activating option *"Share PLC process image"*, the PLC firmware creates a second process image as Shared Memory ("Shared Image" in Figure 26). Its task is to exchange data with external applications. Hereby, the PLC firmware functions as Server and the external, user-specific C/C++ application functions as Client.

***ReadSectorTable*** and ***WriteSectorTable*** both control the copying of data between the two process images. Both tables are filled by the Client (external, user-specific C/C++ application) and are executed by Server (PLC runtime system). The Client defines ranges of the PLC process image from which it will read data (*ReadSectorTable*) or in which it will write data (*WriteSectorTable*). Hence, the terms *"Read"* and *"Write"* refer to data tranfer directions from the viewpoint of the Client.

Sections to read and write may comprise all sections of the entire process image – input, output as well as marker sections. This allows for example that a Client application writes data into the input section of the PLC process image and reads data from the output section. Figure 26 shows the sequence of single read and write operations. Prior to the execution of a PLC program cycle, the physical inputs are imported into the local process image of the PLC (1). Afterwards, all sections defined in *WriteSectorTable* are taken over from the shared process image into the local process image (2). By following this sequence, a Client application for example is able to overwrite the value of a physical input. This may be used for simulation purposes as well as for setting input data to constant values (*"Forcen"*). Similarly, prior to writing the process image onto the physical outputs (6), sections defined in *WriteSectorTable* are taken over from the shared process image into the local process image. (4). Thus, a Client application is able to overwrite output information generated by the PLC program.

The PLC firmware provides the **setup of the process image**. The Client application receives information about the setup of the process image via function ***ShPImgCIntSetup()***. This function enters start offsets and values of the input, output and marker sections into the structure of type *tShPImgLayoutDscrpt*. Function ***ShPImgCIntGetDataSect()*** provides the start address of the shared process image. Upon defining a variable within the PLC program, its absolute position within the process image is determined through sections (%I = Input, %Q = Output, %M = Marker) and offset (e.g. %MB**128**.0). In each section the offset starts at zero, so that for example creating a new variable in the marker section would be independent of values in the input and output section. Creating a corresponding **pair of variables** in the PLC program as well as in the C/C++ application allows for data exchange between the PLC program and the external application. Therefore, both sides must refer to the same address. Structure *tShPImgLayoutDscrpt* reflects the physical setup of the process image in the PLC firmware including input, output and marker sections. This is to use an addressing procedure for defining appropriate variables in the C/C++ application that is comparable to the PLC program. Hence, also in the C/C++ program a variable is defined in the shared process image by indicating the respective section and its offset. The following example illustrates the creation of a corresponding variable pair in the PLC program and C/C++ application:




As described above, ***ReadSectorTable*** and ***WriteSectorTable*** manage the copy process to exchange variable contents between the PLC and the C/C++ program. Following the example illustrated, the Client (C/C++ application) must enter an appropriate value into the *WriteSectorTable* to transfer the value of a variable from the C/C++ application to the PLC program (*WriteSectorTable*, because the Client "writes" the variable to the Server):

```
// specify offset and size of 'Var1' and define sync type (always or on demand?)
WriteSectTab[0].m_uiPImgDataSectOffs = ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 128;
WriteSectTab[0].m_uiPImgDataSectSize = sizeof(BYTE);
WriteSectTab[0].m_SyncType           = kShPImgSyncOnDemand;

// define the WriteSectorTable with the size of 1 entry
ShPImgClntSetupWriteSectTable (WriteSectTab, 1);
```

If several variable pairs are generated within the same transfer direction for the data exchange between the PLC program and the C/C++ application, they should possibly all be defined in one coherent address range. Thus, it is possible to list them as one entry in the appropriate SectorTable. The address of the first variable must be set as the *SectorOffset* and the sum of the variable sizes as *SectorSize*. Combining the variables improves the efficiency and the performance of the copy processes.

For each entry of the *WriteSectorTable* an appropriate *SyncType* must be defined. It determines whether the section is generally taken over from the shared process image into the local image whenever there are two successive PLC cycles (***kShPImgSyncAlways***) or whether it is taken over on demand (***kShPImgSyncOnDemand***). If classified as *SyncOnDemand*, the data only is copied if the respective section before was explicitly marked as updated. This takes places by calling function ***ShPImgClntWriteSectMarkNewData()*** and entering the corresponding *WriteSectorTable*-Index (e.g. 0 for *WriteSectTab[0]* etc.).

*kShPImgSyncAlways* is provided as *SyncType* for the *ReadSectorTable* (the value of the member element *m_SyncType* is ignored). The PLC firmware is not able to identify which variables were changed by the PLC program of the cycle before. Hence, all sections defined in *ReadSectorTable* are always taken over from the local image into the shared process image. Thus, the respective variables in the shared process image always hold the actual values.

The PLC firmware and the C/C++ application both use the shared process image. To prevent conflicts due to accesses from both of those in parallel running processes at the same time, the shared process image is internally protected by a semaphore. If one process requires access to the shared process image, this process enters a critical section by setting the semaphore first and receiving exclusive access to the shared process image second. If the other process requires access to the shared process image at the same time, it also must enter a critical section by trying to set the semaphore. In this case, the operating system identifies that the shared process image is already being used. It blocks the second process until the first process leaves the critical section and releases the semaphore. Thereby, the operating system assures that only one of the two in parallel running processes (PLC runtime system and C/C++ application) may enter the critical section and receives access to the shared process image. To ensure that both processes do not interfere with each other too much, they should enter the critical section as less as possible and only as long as necessary. Otherwise, the PLC cycle time may be extended and runtime variations (Jitter) may occur.

The client application has available two functions to set the semaphore and to block exclusive access to the shared process image. Function ***ShPImgClntLockSegment()*** is necessary to enter the critical section and function ***ShPImgClntUnlockSegment()*** to leave it. The segment between both functions is called protected section, because in this segment the client application holds access to the shared process image without competition. The consistency of read or written data is only guaranteed within such a protected section. Outside the protected section, the shared process image may anytime be manipulated by the PLC runtime system. The following example shows the exclusive access to the shared process image in the C/C++ application:

```
ShPImgClntLockSegment();
{
    // write new data value into Var1
    *pbVar1 = bAnyData;

    // mark new data for WriteSectorTable entry number 0
    ShPImgClntWriteSectMarkNewData (0);
}
ShPImgClntUnlockSegment();
```

For the example above, *kShPImgSyncOnDemand* was defined as *SyncType* upon generating entry *WriteSectorTable*. Hence, taking over variable *Var1* from the shared process image into the local image can only take place if the respective section was beforehand explicitly marked as updated. Therefore, it is necessary to call function ***ShPImgClntWriteSectMarkNewData()***. Since function *ShPImgClntWriteSectMarkNewData()* does not modify the semaphore, it may only be used within a protected section (see example) – such as the code section between *ShPImgClntLockSegment()* and *ShPImgClntUnlockSegment()*.

The synchronization between local image and shared process image by the PLC runtime system only takes place in-between two successive PLC cycles. A client application (user-specific C/C++ program) is not directly informed about this point of time, but it can get information about the update of the shared process image from the PLC runtime system. Therefore, the client application must define a callback handler of the type *tShPImgAppNewDataSigHandler*, e.g.:

```
static  void  AppSigHandlerNewData (void)
{
    fNewDataSignaled_l = TRUE;
}
```

This callback handler must be registered with the help of function ***ShPImgClntSetNewDataSigHandler()***. The handler is selected subsequent to a synchronization of the two images.

The **callback handler of the client application is called within the context of a Linux signal handler** (the PLC runtime system informs the client using Linux function *kill()*). Accordingly, all common **restrictions** for the Linux signal handler also apply to the callback handler of the client application. In particular, it is only allowed to call a few operating system functions that are explicitly marked as reentrant-proof. Please pay attention to not make reentrant calls of local functions within the client application. As shown in the example, only a global flag should be set for the signaling within the callback handler. This flag will later on be evaluated and processed in the main loop of the client application.

### 8.1.2 API of the shared process image client

As illustrated in Figure 26, the user-specific C/C++ application exclusively uses the API (Application Programming Interface) provided by the s*hared process image client*. This API is declared in the

header file *shpimgclient.h* and implemented in the source file *shpimgclient.c*. It contains the following types (partly defined in *shpimg.h*) and functions:

**Structure *tShPImgLayoutDscrpt***

```
typedef struct
{
    // definition of process image sections
    unsigned int        m_uiPImgInputOffs;      // start offset of input section
    unsigned int        m_uiPImgInputSize;      // size of input section
    unsigned int        m_uiPImgOutputOffs;     // start offset of output section
    unsigned int        m_uiPImgOutputSize;     // size of output section
    unsigned int        m_uiPImgMarkerOffs;     // start offset of marker section
    unsigned int        m_uiPImgMarkerSize;     // size of marker section

} tShPImgLayoutDscrpt;
```

Structure ***tShPImgLayoutDscrpt*** describes the setup of the process image given by the PLC firmware. The client application receives the information about the setup of the process image via function *ShPImgClntSetup()*. This function enters start offsets and values of input, output and marker sections into the structure provided upon function calling.

**Structure *tShPImgSectDscrpt***

```
typedef struct
{
    // definition of data exchange section
    unsigned int        m_uiPImgDataSectOffs;
    unsigned int        m_uiPImgDataSectSize;
    tShPImgSyncType     m_SyncType;              // only used for WriteSectTab
    BOOL                m_fNewData;

} tShPImgSectDscrpt;
```

Structure ***tShPImgSectDscrpt*** describes the setup of a *ReadSectorTable* or *WriteSectorTable* entry that must be defined by the client. Both tables support the synchronization between the local image of the PLC runtime system and the shared process image (see section 8.1.1). Member element *m_uiPImgDataSectOffs* defines the absolute start offset of the section within the shared process images. The respective start offsets of the input, output and marker sections can be determined through structure *tShPImgLayoutDscrpt*. Member element *m_uiPImgDataSectSize* determines the size of the section which may include one or more variables. Member element *m_SyncType* only applies to entries of the WriteSectorTable. It determines whether the section is generally taken over from the shared process image into the local image whenever there are two successive PLC cycles (***kShPImgSyncAlways***) or whether it is taken over on demand (***kShPImgSyncOnDemand***). If classified as *SyncOnDemand*, the data must be marked as modified by calling function *ShPImgClntWriteSectMarkNewData()*. It sets the member element *m_fNewData* to TRUE. The client application should never directly modify this member element.

**Function *ShPImgClntSetup***

```
BOOL  ShPImgClntSetup  (tShPImgLayoutDscrpt* pShPImgLayoutDscrpt_p);
```

Function ***ShPImgClntSetup()*** initializes the *shared process image client* and connects itself with the storage segment for the shared process image which is generated by the PLC runtime system. Afterwards, it enters the start offsets and values of the input, output and marker sections into the structure of type *tShPImgLayoutDscrpt* provided upon function call. Hence, the

client application receives notice about the process image setup managed by the PLC firmware.

If the PLC runtime system is not active when the function is called or if it has not generated a shared process image (option *"Share PLC process image"* in the PLC configuration deactivated, see section 8.1.1), the function will return with the return value FALSE. If the initialization was successful, the return value will be TRUE.

### Function *ShPImgClntRelease*

```
BOOL  ShPImgClntRelease  (void);
```

Function ***ShPImgClntRelease()*** shuts down the s*hared process image client* and disconnects the connection to the storage segment generated for the shared process image by the PLC runtime system.

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

### Function *ShPImgClntSetNewDataSigHandler*

```
BOOL  ShPImgClntSetNewDataSigHandler  (
         tShPImgAppNewDataSigHandler pfnShPImgAppNewDataSigHandler_p);
```

Function ***ShPImgClntSetNewDataSigHandler()*** registers a user-specific callback handler. This callback handler is called after a synchronization of both images. Registered callback handlers are cleared by the parameter NULL.

The **callback handler is called within the context of a Linux signal handler**. Accordingly, all common **restrictions** for the Linux signal handler also apply to the callback handler (see section 8.1.1).

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

### Function *ShPImgClntGetHeader*

```
tShPImgHeader*  ShPImgClntGetHeader  (void);
```

Function ***ShPImgClntGetHeader()*** provides a pointer to the internally used structure type *tShPImgHeader* to manage the shared process image. The client application does usually not need this structure, because all data that it includes can be read and written through functions of the API provided by the *shared process image client*.

### Function *ShPImgClntGetDataSect*

```
BYTE*  ShPImgClntGetDataSect  (void);
```

Function ***ShPImgClntGetDataSect()*** provides a pointer to the beginning of the shared process image. This pointer represents the basic address for all accesses to the shared process image; including the definition of sections *ReadSectorTable* and *WriteSectorTable* (see section 8.1.1).

### Functions *ShPImgClntLockSegment* and *ShPImgClntUnlockSegment*

```
BOOL  ShPImgClntLockSegment  (void);
BOOL  ShPImgClntUnlockSegment  (void);
```

To exclusively access the shared process image, the client application has available two functions - function ***ShPImgClntLockSegment()*** to enter the critical section and function ***ShPImgClntUnlockSegment()*** to leave it. The segment between both functions is called protected section, because in this segment the client application holds unrivaled access to the shared process image (see section 8.1.1). The consistency of read or written data is only guaranteed within such a protected section. Outside the protected section, the shared process image may anytime be manipulated by the PLC runtime system. To ensure that the client application does not interfere with the PLC runtime system too much, the critical sections should be set as less as possible and only as long as necessary. Otherwise, the PLC cycle time may be extended and runtime variations (Jitter) may occur.

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

### Function *ShPImgClntSetupReadSectTable*

```
BOOL  ShPImgClntSetupReadSectTable  (
         tShPImgSectDscrpt* paShPImgReadSectTab_p,
         unsigned int uiNumOfReadDscrptUsed_p);
```

Function ***ShPImgClntSetupReadSectTable()*** initializes the *ReadSectorTable* with the values defined by the client. The client hereby determines those sections of the PLC process image from which it wants to read data (see section 8.1.1). Parameter *paShPImgReadSectTab_p* holds elements of the structure *tShPImgSectDscrpt* and must be transferred as start address of a section. Parameter *uiNumOfReadDscrptUsed_p* indicates how many elements the section has.

*kShPImgSyncAlways* is provided as *SyncType* for the *ReadSectorTable*.

The maximum amount of possible elements for the *ReadSectorTable* is defined by the constant *SHPIMG_READ_SECT_TAB_ENTRIES* and can only be modified if the shared library *"pc9g20drv.so"* is generated again and at the time (this requires SO-1106 - "Driver Development Kit for the ECUcore-9G20", see section 8.2).

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

### Function *ShPImgClntSetupWriteSectTable*

```
BOOL  ShPImgClntSetupWriteSectTable  (
         tShPImgSectDscrpt* paShPImgWriteSectTab_p,
         unsigned int uiNumOfWriteDscrptUsed_p);
```

Function ***ShPImgClntSetupWriteSectTable***() initializes the *WriteSectorTable* with the values defined by the client. The client hereby determines those sections of the PLC process image from which it wants to write data (see section 8.1.1). Parameter *paShPImgWriteSectTab_p* holds elements of structure *tShPImgSectDscrpt* and must be transferred as start address of a section. Parameter *uiNumOfWriteDscrptUsed_p* indicates how many elements the section has.

For each entry in the *WriteSectorTable* the *SyncType* must be defined. This *SyncType* defines whether the section is always taken over into the local image between two PLC cycles (***kShPImgSyncAlways***) or only on demand (***kShPImgSyncOnDemand***). If taken over on demand, the respective section is explicitly marked as updated by calling

*ShPImgClntWriteSectMarkNewData()*.

The maximum amount of possible elements for the *WriteSectorTable* is defined by the constant *SHPIMG_WRITE_SECT_TAB_ENTRIES* and can only be modified if the shared library *"pc9g20drv.so"* is generated again and at the same time (this requires SO-1106 - "Driver Development Kit for the ECUcore-9G20", see section 8.2).

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

**Function *ShPImgClntWriteSectMarkNewData***

```
BOOL  ShPImgClntWriteSectMarkNewData  (unsigned int uiWriteDscrptIdx_p);
```

For the content of a section that is held by the *WriteSectorTable*, function ***ShPImgClntWriteSectMarkNewData()*** marks this content as modified. This function is used (for sections with *SyncType* ***kShPImgSyncOnDemand***) to initiate the copy process of data from the shared process image into the local image of the PLC.

Function *ShPImgClntWriteSectMarkNewData()* directly accesses the header of the shared process image without setting a semaphore before. Hence, it may only be used within the protected section – in the code section between *ShPImgClntLockSegment()* and *ShPImgClntUnlockSegment()*.

If executed successfully, the function delivers return value TRUE. If an error occurs, it will deliver return value FALSE.

### 8.1.3 Creating a user-specific client application

**Software package SO-1105 ("VMware image of the Linux development system")** is the precondition for the implementation of user-specific C/C++ applications. It contains a complete Linux development system in the form of a VMware image. Hence, it allows for an easy introduction into the C/C++ software development for the PLCcore-9G20. Thus, the VMware image is the ideal basis to develop Linux-based user programs on the same host PC that already has the *OpenPCS* IEC 61131 programming system installed on it. The VMware image of the Linux development system includes the GNU-Crosscompiler Toolchain for ARM9 processors. Additionally, it includes essential server services that are preconfigured and usable for effective software development. Details about the VMware image of the Linux development system and instructions for its usage are described in the *"System Manual ECUcore-9G20"* (Manual no: L-1253).

As illustrated in Figure 26, the user-specific C/C++ application uses the API (files *shpimgclient.c* and *shpimgclient.h*) which is provided by the *shared process image client*. The *shared process image client* is based on services provided by the *shared memory client* (files *shmclient.c* and *shmclient.c*). Both client implementations are necessary to generate a user-specific C/C++ application. The archive of the *shared process image demos* (***shpimgdemo.tar.gz***) contains the respective files. To create own user-specific client applications, it is recommended to use this demo project as the basis for own adaptations and extensions. Moreover, this demo project contains a Makefile with all relevant configuration adjustments that are necessary to create a Linux application for the PLCcore-9G20. Table 21 lists all files of the archive *"shpimgdemo.tar.gz"* and classifies those as general part of the C/C++ application or as specific component for the demo project *"shpimgdemo"*.

*Table 21: Content of the archive files "shpimgdemo.tar.gz"*

| File | Necessary for all C/C++ applications | In particular for demo ***"shpimgdemo"*** |
|---|---|---|
| shpimgclient.c | x | |
| shpimgclient.h | x | |
| shmclient.c | x | |
| shmclient.h | x | |
| shpimg.h | x | |
| global.h | x | |
| Makefile | draft, to be adjusted | |
| shpimgdemo.c | | x |
| trmio.c | | x |
| trmio.h | | x |
| trace.c | | x |

The archive file ***"shpimgdemo.tar.gz"*** including the *shared process image demo* must be unzipped into any subdirectory following the path *"/projects/ECUcore-9G20/user"* within the Linux development system. Therefore, command *"tar"* must be called:

```
tar  xzvf  shpimgdemo.tar.gz
```

During the unzipping process, command *"tar"* independently generates the subdirectory *"shpimgdemo"*. For example, if the command is called in directory *"/projects/ECUcore-9G20/user"*, all archive files will be unzipped into the path *"/projects/ECUcore-9G20/user/**shpimgdemo**"*. Figure 27 exemplifies the unzipping process of *"shpimgdemo.tar.gz"* within the Linux development system.

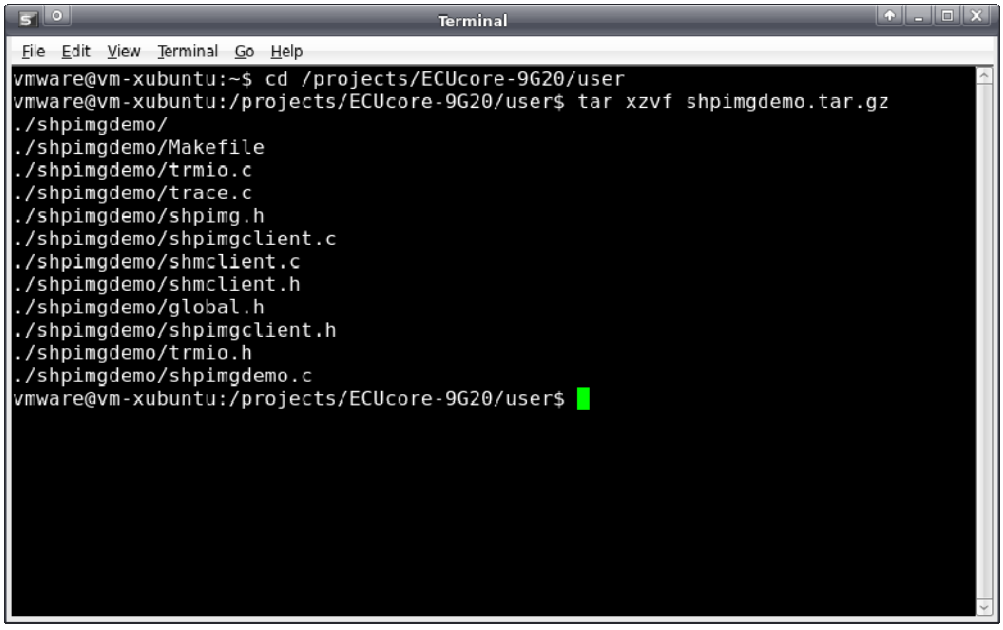



*Figure 27: Unzipping the archive files shpimgdemo.tar.gz in the Linux development system*

After unzipping and switching into subdirectory *"shpimgdemo"*, the demo project can be created by calling command *"make"*:

```
cd  shpimgdemo
make
```

Figure 28 shows how the demo project *"shpimgdemo"* is generated in the Linux development system.

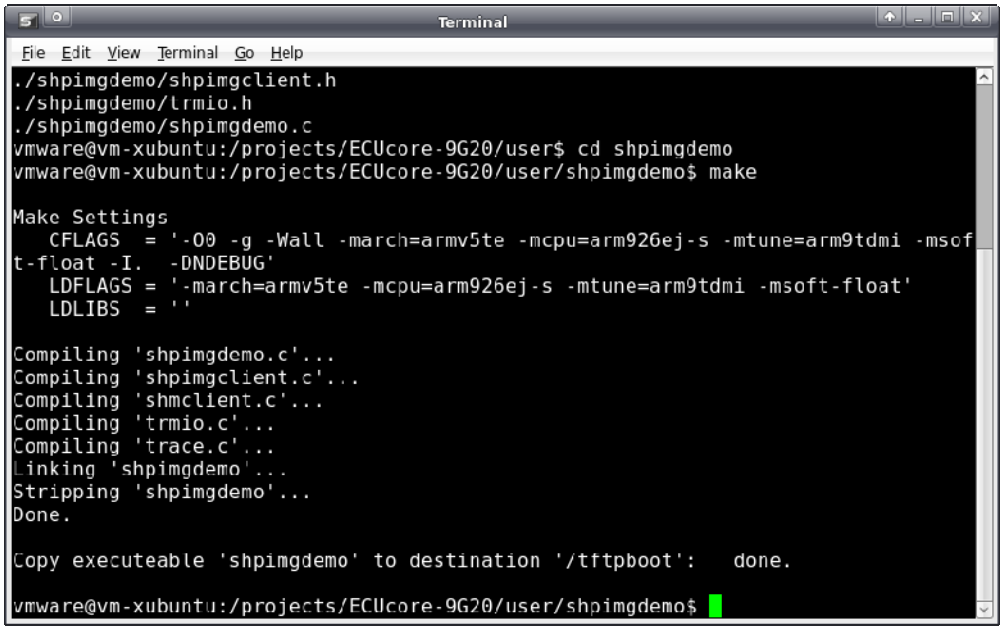



*Figure 28: Generating the demo project "shpimgdemo" in the Linux development system*

Section 8.1.4 describes the usage and handling of the demo project *"shpimgdemo"* on the PLCcore-9G20.

### 8.1.4 Example for using the shared process image

The demo project *"shpimgdemo"* (described in section 8.1.3) in connection with the PLC program example *"RunLight"* both exemplify the data exchange between a PLC program and a user-specific C/C++ application.

**Technical background**

The PLC program generates some variables in the process image as directly addressable variables. In a C/C++ application, all those variables are usable via the shared process image. For the PLC program example *"RunLight"* those are the following variables:

```
(* variables for local control via on-board I/Os *)
bButtonGroup        AT %IB0.0   : BYTE;
iAnalogValue        AT %IW8.0   : INT;
bLEDGroup0          AT %QB0.0   : BYTE;
bLEDGroup1          AT %QB1.0   : BYTE;

(* variables for remote control via shared process image  *)
uiRemoteSlidbarLen  AT %MW512.0 : UINT;     (* out: length of slidebar         *)
bRemoteStatus       AT %MB514.0 : BYTE;     (* out: Bit0: RemoteControl=on/off *)
bRemoteDirCtrl      AT %MB515.0 : BYTE;     (* in:  direction left/right       *)
iRemoteSpeedCtrl    AT %MW516.0 : INT;      (* in:  speed                      *)
```

Variables of the PLC program are accessible from a C/C++ application via the shared process image. Therefore, sections must be generated for the *ReadSectorTable* and *WriteSectorTable* on the one hand and on the other hand, pointers must be defined for accessing the variables. The following program extract shows this using the example *"shpimgdemo.c"*. Function *ShPImgClntSetup()* inserts the start offsets of input, output and marker sections into the structure *ShPImgLayoutDscrpt*. Hence, on the basis of the initial address provided by *ShPImgClntGetDataSect()*, the absolute initial addresses of each section in the shared process image can be determined. To identify the address of a variable, the variable's offset within the particular section must be added. For example, the absolute address to access the variable *"bRemoteDirCtrl AT %MB515.0 : BYTE;"* results from the sum of the initial address of the shared process image (pabShPImgDataSect), the start offset of the marker section (*ShPImgLayoutDscrpt.m_uiPImgMarkerOffs für "%M..."*) as well as the direct address within the marker section which was defined in the PLC program (515 for "%MB515.0"):

```
    pbPImgVar_61131_bDirCtrl = (BYTE*) (pabShPImgDataSect
               + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 515);
```

The following code extract shows the complete definition of all variables in the demo project used for exchanging data with the PLC program:

```
// ---- Setup shared process image client ----
fRes = ShPImgClntSetup (&ShPImgLayoutDscrpt);
if ( !fRes )
{
    printf ("\n*** ERROR *** Init of shared process image client failed");
}

pabShPImgDataSect = ShPImgClntGetDataSect();


// ---- Read Sector Table ----
// Input Section:       bButtonGroup AT %IB0.0
{
    ShPImgReadSectTab[0].m_uiPImgDataSectOffs =
                ShPImgLayoutDscrpt.m_uiPImgInputOffs + 0;
    ShPImgReadSectTab[0].m_uiPImgDataSectSize = sizeof(BYTE);
    ShPImgReadSectTab[0].m_SyncType           = kShPImgSyncAlways;

    pbPImgVar_61131_bButtonGroup = (BYTE*) (pabShPImgDataSect
               + ShPImgLayoutDscrpt.m_uiPImgInputOffs + 0);
}

// Output Section:      bLEDGroup0 AT %QB0.0
//                      bLEDGroup1 AT %QB1.0
{
    ShPImgReadSectTab[1].m_uiPImgDataSectOffs =
                ShPImgLayoutDscrpt.m_uiPImgOutputOffs + 0;
    ShPImgReadSectTab[1].m_uiPImgDataSectSize = sizeof(BYTE) + sizeof(BYTE);
    ShPImgReadSectTab[1].m_SyncType           = kShPImgSyncAlways;

    pbPImgVar_61131_bLEDGroup0 = (BYTE*) (pabShPImgDataSect
               + ShPImgLayoutDscrpt.m_uiPImgOutputOffs + 0);
    pbPImgVar_61131_bLEDGroup1 = (BYTE*) (pabShPImgDataSect
               + ShPImgLayoutDscrpt.m_uiPImgOutputOffs + 1);
}
```

```
// Marker Section:      uiSlidbarLen AT %MW512.0
//                      bStatus      AT %MB514.0
{
    ShPImgReadSectTab[2].m_uiPImgDataSectOffs =
                ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 512;
    ShPImgReadSectTab[2].m_uiPImgDataSectSize =   sizeof(unsigned short int)
                                                + sizeof(BYTE);
    ShPImgReadSectTab[2].m_SyncType           = kShPImgSyncAlways;

    pbPImgVar_61131_usiSlidbarLen = (unsigned short int*) (pabShPImgDataSect
              + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 512);
    pbPImgVar_61131_bStatus = (BYTE*) (pabShPImgDataSect
              + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 514);
}

fRes = ShPImgClntSetupReadSectTable (ShPImgReadSectTab, 3);
if ( !fRes )
{
    printf ("\n*** ERROR *** Initialization of read sector table failed");
}

// ---- Write Sector Table ----
// Marker Section:      bDirCtrl   AT %MB513.0
//                      iSpeedCtrl AT %MB514.0
{
    ShPImgWriteSectTab[0].m_uiPImgDataSectOffs =
                ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 515;
    ShPImgWriteSectTab[0].m_uiPImgDataSectSize = sizeof(BYTE) + sizeof(WORD);
    ShPImgWriteSectTab[0].m_SyncType           = kShPImgSyncOnDemand;

    pbPImgVar_61131_bDirCtrl = (BYTE*) (pabShPImgDataSect
              + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 515);
    psiPImgVar_61131_iSpeedCtrl = (short int*) (pabShPImgDataSect
                  + ShPImgLayoutDscrpt.m_uiPImgMarkerOffs + 516);
}

fRes = ShPImgClntSetupWriteSectTable (ShPImgWriteSectTab, 1);
if ( !fRes )
{
    printf ("\n*** ERROR *** Initialization of write sector table failed");
}
```

## Realization on the PLCcore-9G20

To enable the execution of the *shared process image demo* without previous introduction into the Linux-based C/C++ programming for the PLCcore-9G20, the module comes with a preinstalled, translated and ready-to-run program version and PLC firmware (*"/home/plc/shpimgdemo"*). The following description refers to this program version. Alternatively, the demo project can be newly-generated from the corresponding source files (see section 8.1.3) and can be started afterwards.

The following steps are necessary to run the *shared process image demo* on the PLCcore-9G20:

1. **Activate option *"Share PLC process image"*** in the PLC configuration (see sections 8.1.1, 7.4.1 and 7.4.3).

2. Open the PLC program example *"RunLight"* in the *OpenPCS* IEC 61131 programming system und build the project for a target hardware of the type *"SYSTEC - PLCcore-9G20"*.

3. Select the network connection to the PLCcore-9G20 und download the program.

4. Start the PLC program on the PLCcore-9G20.

5. Login to the command shell of the PLCcore-9G20 as described in section 7.8.1.

6. Switch to the directory *"/home/plc"* and call the demo program *"shpimgdemo"*:

```
cd  /home/plc
./shpimgdemo
```

The digital outputs of the PLCcore-9G20 are selected as runlight. The speed is modifiable via the analog input AI0 (Poti at the ADC of the Development Board). With the help of pushbuttons S400 (DI0) and S401 (DI1), the running direction can be changed. After starting the demo program *"shpimgdemo"* on the PLCcore-9G20, actual status information about the runlight is indicated cyclically in the terminal (see Figure 29).

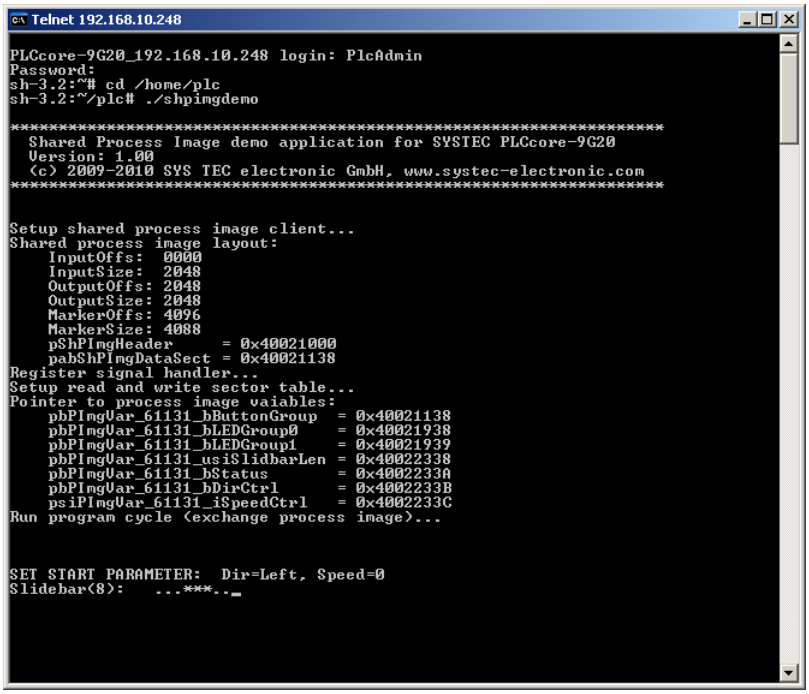



*Figure 29: Terminal outputs of the demo program "shpimgdemo" after start*

7. By pressing pushbutton S403 (DI3), the control of the runlight direction and speed is handed over to the demo program *"shpimgdemo"*. Afterwards, the running direction may be set by the C application by using the cursor pushbuttons left and right (← und →) in the terminal window and the speed may be changed by using cursor pushbuttons up and down (↑ und ↓).

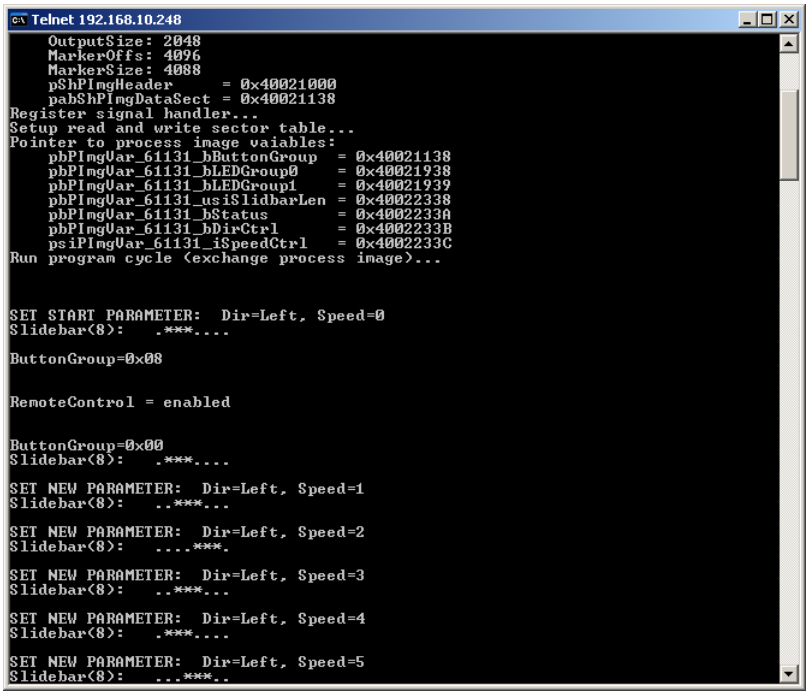



*Figure 30: Terminal outputs of the demo program "shpimgdemo" after user inputs*

Figure 30 shows the terminal outputs of the demo program *"shpimgdemo"* in answer to activating the cursor pushbuttons.

The demo program *"shpimgdemo"* may be terminated by pressing "Ctrl+C" in the terminal window.

## 8.2 Driver Development Kit (DDK) for the PLCcore-9G20

**The Driver Development Kit (DDK) for the ECUcore-9G20 (resp. PLCcore-9G20) is distributed as additional software package with the order number SO-1106. It is not included in the delivery of the PLCcore-9G20 or the Development Kit PLCcore-9G20.** The "Software Manual Driver Development Kit for the ECUcore-9G20" (Manual no.: L-1257) provides details about the DDK.

The Driver Development Kit for the ECUcore-9G20 (resp. PLCcore-9G20) enables the user to adapt an I/O level to self-developed baseboards. The Embedded Linux on the PLCcore-9G20 supports dynamic loading of drivers during runtime. Hence, it allows for a separation of the PLC runtime system and I/O drivers. Consequently, the user is able to completely adapt the I/O driver to own requirements – without having to modify the PLC runtime system.

By using the DDK, the following resources may be integrated into the I/O level:

- Periphery (usually GPIO) of the AT91SAM9G20
- on-board FPGA (Lattice ECP2-6)
- Address-/Data Bus (memory-mapped periphery)
- SPI-Bus and $I^2C$-Bus
- All other resources provided by the operating system, e.g. file system and TCP/IP

Figure 31 provides an overview of the DDK structure and its components. The DDK contains the FPGA software sources (VHDL) as well as the source code of the Linux kernel driver (*pc9g20drv.ko*) and the Linux user library (*pc9g20drv.so*). Additionally, the DDK includes a PLD Programming Tool (*pldtool* + *plddrv.ko*) which allows for a FPGA software update without extra JTAG hardware.

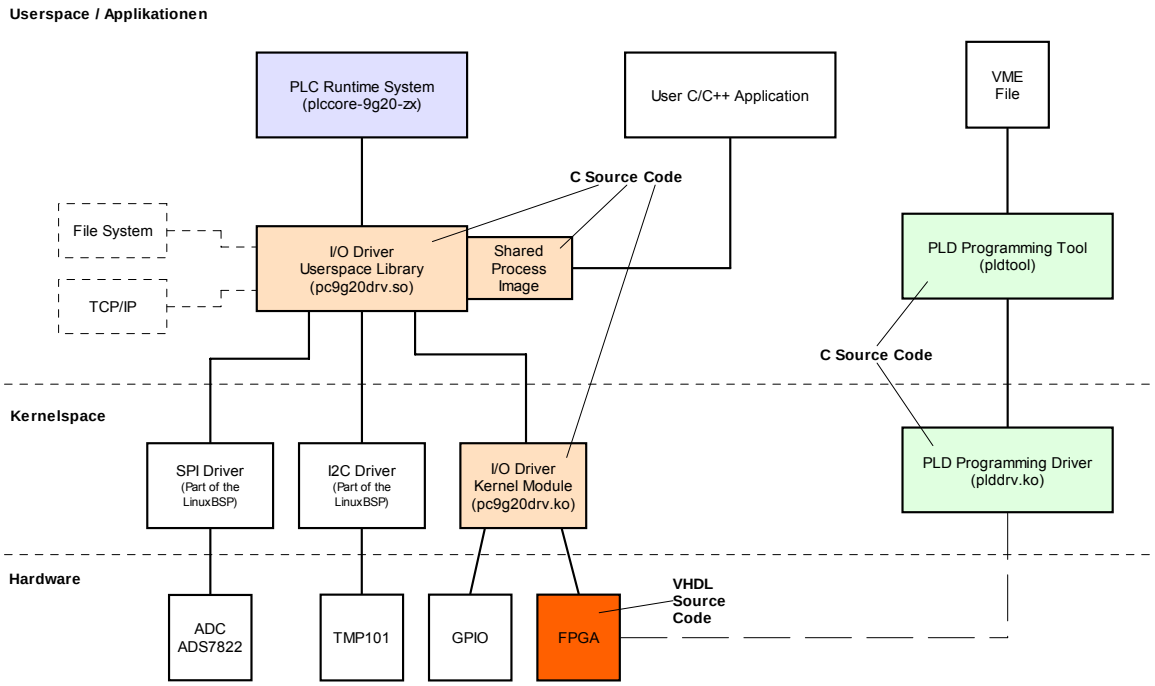



*Figure 31: Overview of the Driver Development Kit for the PLCcore-9G20*

## Scope of delivery / components of the DDK:

The DDK contains the following components:

1. VHDL project for the FPGA; comprises all files necessary to regenerate FPGA software (VHLS source files, pin assignment, timing settings, project file etc.)

2. Source code for the Linux kernel driver (*pc9g20drv.ko*, see Figure 31); includes all files necessary to regenerate kernel drivers (C and H files, Make file etc.)

3. Source code for the Linux user library (*pc9g20drv.so*, see Figure 31); contains all files (incl. implementation of Shared Process Image) necessary to regenerate a user library (C and H files, Make file etc.)

4. PLD/FPGA Programming Tool (*pldtool* + *plddrv.ko*); enables a FPGA software update using Linux without additional JTAG hardware

5. I/O driver demo application (*iodrvdemo*) in the source code; allows for a quick and trouble-free test of the I/O drivers

6. Documentation

The Driver Development Kit is based on the software package **SO-1105** ("VMware-Image of the Linux development system"). It contains sources of the LinuxBSP used and it includes the necessary GNU-Crosscompiler Toolchain for ARM9 processors.

## 8.3 Testing the hardware connections

The PLCcore-9G20 primarily is designed as vendor part for the application in industrial controls. Hence, the PLCcore-9G20 typically is integrated in a user-specific baseboard. To enable trouble-free inspection of correct I/O activation, the test program *"iodrvdemo"* is installed on the module together with the PLC firmware. This test program is directly tied in with the I/O driver and allows quick and direct access to the periphery.

At first, if a PLC runtime system is running, it must be quit. This is to ensure that the test program *"iodrvdemo"* receives exclusive access to all I/O resources. To do so, script *"stopplc"* may possibly be called:

```
cd  /home/plc
./stopplc
```

Afterwards, the I/O driver may be reloaded and the test program *"iodrvdemo"* may be started:

```
insmod  pc9g20drv.ko
./iodrvdemo
```

Figure 32 exemplifies the testing of the hardware connections using *"iodrvdemo"*.

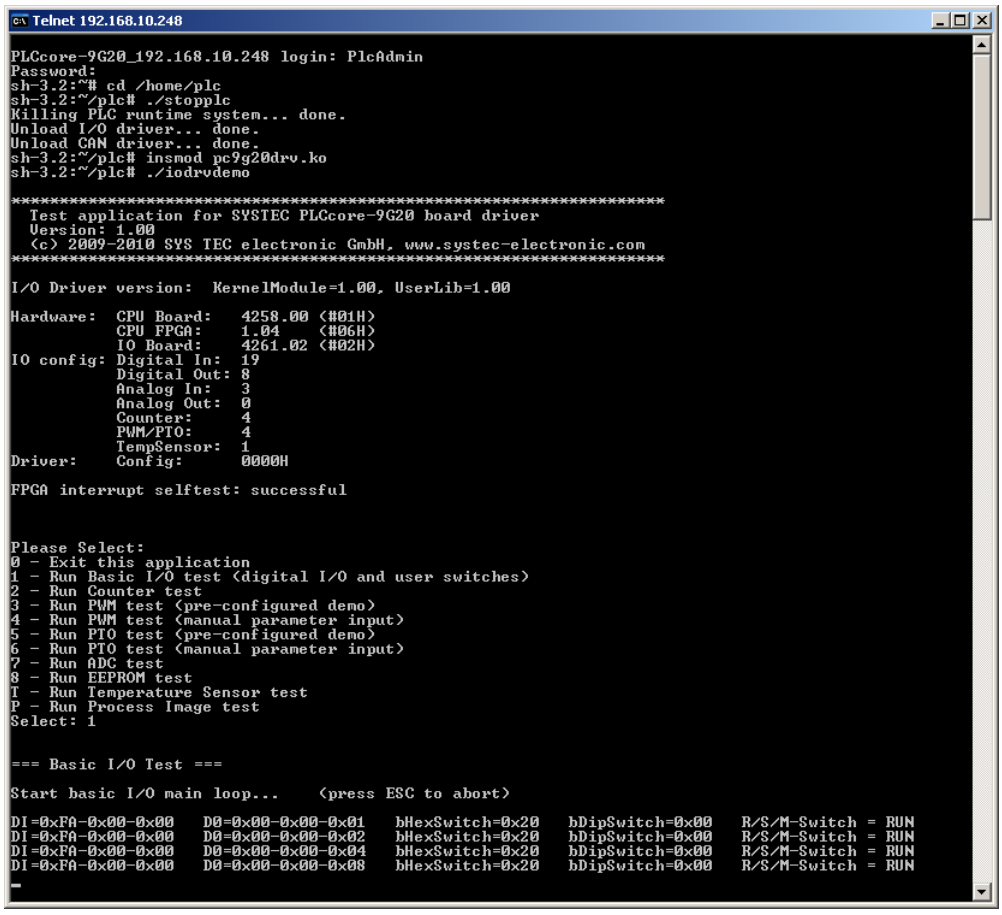



*Figure 32: Testing the hardware connections using "iodrvdemo"*

# Appendix A: Firmware function scope of PLCcore-9G20

Table 22 lists all firmware functions and function blocks available on the PLCcore-9G20.

Sign explanation:

| | |
|---|---|
| FB | Function block |
| FUN | Function |
| Online Help | *OpenPCS* online help |
| L-1054 | Manual *"SYS TEC-specific extensions for OpenPCS / IEC 61131-3"*, Manual no.: L-1054) |
| PARAM:={0,1,2} | values 0, 1 and 2 are valid for the given parameter |

*Table 22: Firmware functions and function blocks of PLCcore-9G20*

| Name | Type | Reference | Remark |
|---|---|---|---|
| ***PLC standard Functions and Function Blocks*** | | | |
| SR | FB | Online Help | |
| RS | FB | Online Help | |
| R_TRIG | FB | Online Help | |
| F_TRIG | FB | Online Help | |
| CTU | FB | Online Help | |
| CTD | FB | Online Help | |
| CTUD | FB | Online Help | |
| TP | FB | Online Help | |
| TON | FB | Online Help | |
| TOF | FB | Online Help | |
| ***Functions and Function Blocks for string manipulation*** | | | |
| LEN | FUN | L-1054 | |
| LEFT | FUN | L-1054 | |
| RIGHT | FUN | L-1054 | |
| MID | FUN | L-1054 | |
| CONCAT | FUN | L-1054 | |
| INSERT | FUN | L-1054 | |
| DELETE | FUN | L-1054 | |
| REPLACE | FUN | L-1054 | |
| FIND | FUN | L-1054 | |
| GETSTRINFO | FB | L-1054 | |
| CHR | FUN | L-1054 | |
| ASC | FUN | L-1054 | |
| STR | FUN | L-1054 | |
| VAL | FUN | L-1054 | |
| ***Functions and Function Blocks for OpenPCS specific task controlling*** | | | |
| ETRC | FB | L-1054 | |
| PTRC | FB | L-1054 | |
| GETVARDATA | FB | Online Help | |
| GETVARFLATADDRESS | FB | Online Help | |
| GETTASKINFO | FB | Online Help | |

| | | | |
|---|---|---|---|
| ***Functions and Function Blocks for handling of non-volatile data*** | | | |
| NVDATA_BIT | FB | L-1054 | DEVICE:={0}, see (1) |
| NVDATA_INT | FB | L-1054 | DEVICE:={0}, see (1) |
| NVDATA_STR | FB | L-1054 | DEVICE:={0}, see (1) |
| ***Functions and Function Blocks for handling of time*** | | | |
| GetTime | FUN | Online Help | |
| GetTimeCS | FUN | Online Help | |
| DT_CLOCK | FB | L-1054 | |
| DT_ABS_TO_REL | FB | L-1054 | |
| DT_REL_TO_ABS | FB | L-1054 | |
| ***Functions and Function Blocks for counter inputs and pulse outputs*** | | | |
| CNT_FUD | FB | L-1054 | CHANNEL:={0,1,2,3} |
| PTO_PWM | FB | L-1054 | CHANNEL:={0,1,2,3} |
| PTO_TAB | FB | L-1054 | CHANNEL:={0,1,2,3} |
| ***Functions and Function Blocks for Serial interfaces*** | | | |
| SIO_INIT | FB | L-1054 | PORT:={0,1,2,3}, see (2) |
| SIO_STATE | FB | L-1054 | PORT:={0,1,2,3} see (2) |
| SIO_READ_CHR | FB | L-1054 | PORT:={0,1,2,3} see (2) |
| SIO_WRITE_CHR | FB | L-1054 | PORT:={0,1,2,3} see (2) |
| SIO_READ_STR | FB | L-1054 | PORT:={0,1,2,3} see (2) |
| SIO_WRITE_STR | FB | L-1054 | PORT:={0,1,2,3} see (2) |
| ***Functions and Function Blocks for CAN interfaces / CANopen*** | | | |
| CAN_GET_LOCALNODE_ID | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_CANOPEN_KERNEL_STATE | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_REGISTER_COBID | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_PDO_READ8 | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_PDO_WRITE8 | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_SDO_READ8 | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_SDO_WRITE8 | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_SDO_READ_STR | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_SDO_WRITE_STR | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_GET_STATE | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_NMT | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_RECV_EMCY_DEV | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_RECV_EMCY | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_WRITE_EMCY | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_RECV_BOOTUP_DEV | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_RECV_BOOTUP | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_ENABLE_CYCLIC_SYNC | FB | L-1054 | NETNUMBER:={0,1} |
| CAN_SEND_SYNC | FB | L-1054 | NETNUMBER:={0,1} |

| ***Functions and Function Blocks for Ethernet interfaces / UDP*** | | | |
|---|---|---|---|
| LAN_GET_HOST_CONFIG | FB | L-1054 | NETNUMBER:={0,1} |
| LAN_ASCII_TO_INET | FB | L-1054 | NETNUMBER:={0,1} |
| LAN_INET_TO_ASCII | FB | L-1054 | NETNUMBER:={0,1} |
| LAN_GET_HOST_BY_NAME | FB | L-1054 | NETNUMBER:={0,1} |
| LAN_GET_HOST_BY_ADDR | FB | L-1054 | NETNUMBER:={0,1} |
| LAN_UDP_CREATE_SOCKET | FB | L-1054 | NETNUMBER:={0,1} |
| LAN_UDP_CLOSE_SOCKET | FB | L-1054 | NETNUMBER:={0,1} |
| LAN_UDP_RECVFROM_STR | FB | L-1054 | NETNUMBER:={0,1} |
| LAN_UDP_SENDTO_STR | FB | L-1054 | NETNUMBER:={0,1} |

(1) All nonvolatile data is filed into directory *"/home/plc/PlcPData.bin"* on the PLCcore-9G20. This file has a fix size of 32 kByte. By calling function blocks of type *NVDATA_Xxx* in a writing mode, the modified data is directly stored into file *"/home/plc/PlcPData.bin"* ("*flush*"). Thus, unsecured data is not getting lost in case of power interruption.

(2) Interface COM0 (PORT:=0) primarily serves as service interface to administer the PLCcore-9G20. Hence, this interface should only be used for sign output. The module always tries to interpret and execute sign inputs as Linux commands (see section 6.5.1).

# Appendix B: Reference design for the PLCcore-9G20

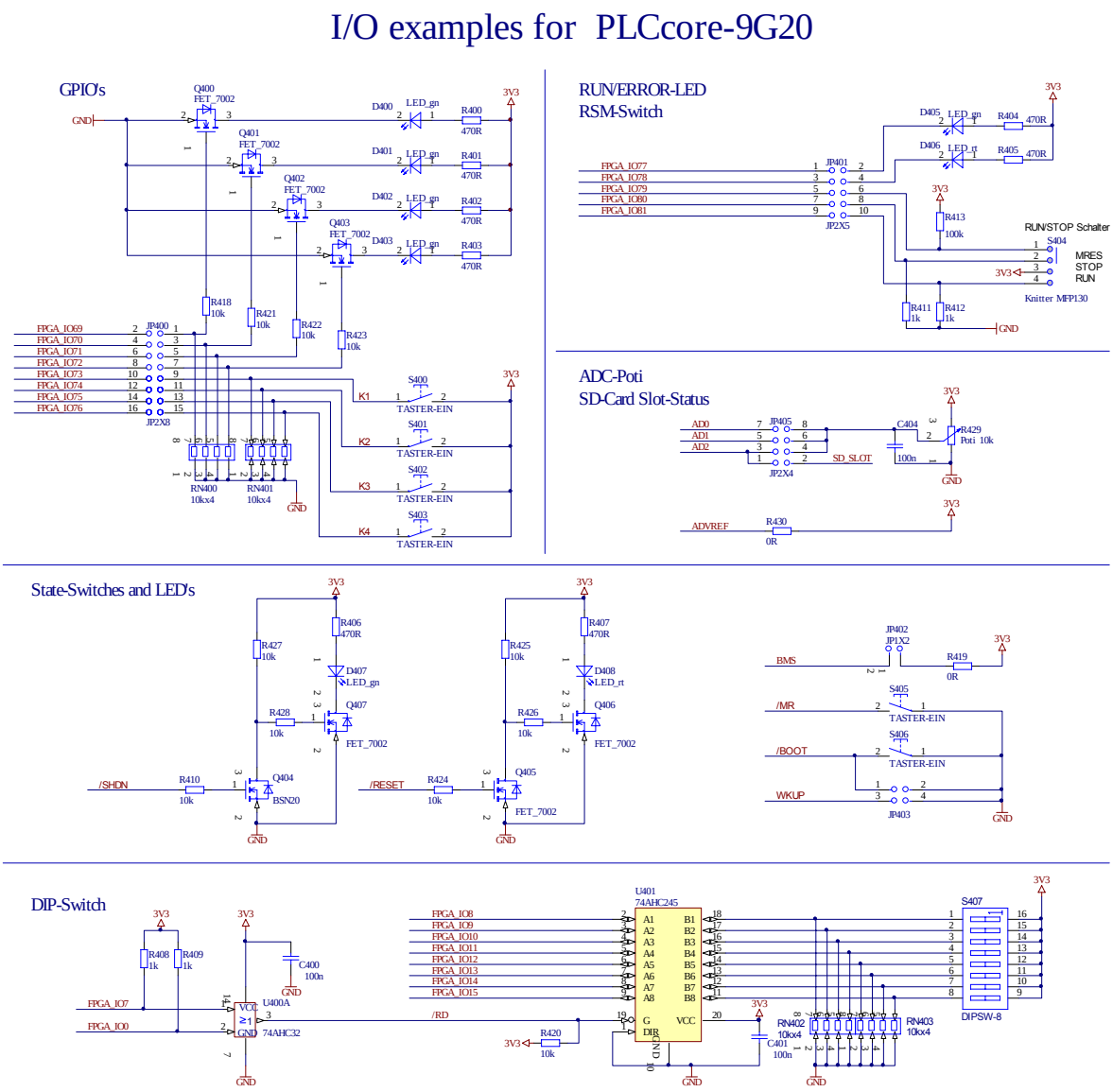



*Figure 33: Reference design for I/O interface connection*

# interface examples for PLCcore-9G20

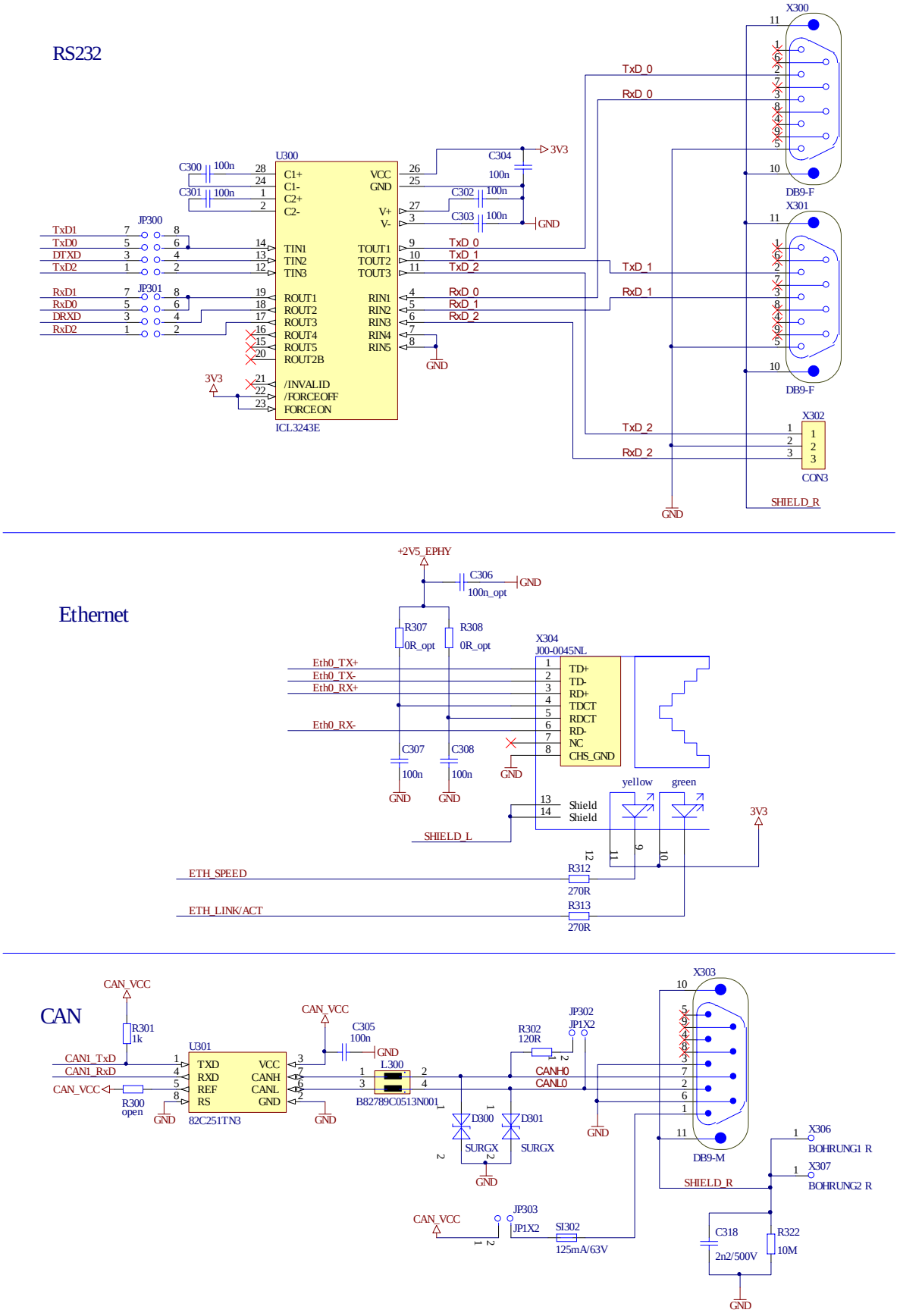



*Figure 34: Reference design for interface circuit*

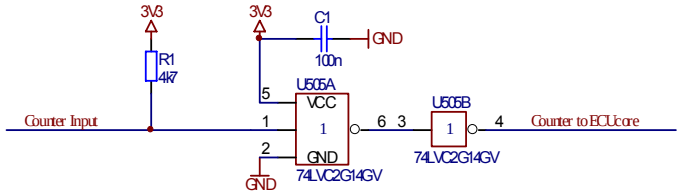



*Figure 35: Reference design for counter input connection*

# Appendix C: GNU GENERAL PUBLIC LICENSE

The Embedded Linux used on the PLCcore-9G20 is licensed under GNU General Public License, version 2. The entire license text is specified below.

The PLC system used and the PLC and C/C++ programs developed by the user are **not** subject to the GNU General Public License!

**GNU GENERAL PUBLIC LICENSE**
**Version 2, June 1991**



**Preamble**

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software -- to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

**GNU GENERAL PUBLIC LICENSE**
**TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year  name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it under certain conditions;
type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items -- whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

# Index

| | |
|---|---|
| **Document:** | System Manual PLCcore-9G20 |
| **Document number:** | L-1254e_1, 1st Edition July 2010 |

**How would you improve this manual?**

**Did you detect any mistakes in this manual?** page

**Submitted by:**

Customer number: _____

Name: _____

Company: _____

Address: _____

**Please return your suggestions to:**

SYS TEC electronic GmbH
August-Bebel-Str. 29
D - 07973 Greiz
GERMANY
Fax: +49 (0) 36 61 / 6279-99
Email: info@systec-electronic.com