

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

## **User Manual** Version 1.0

**Ausgabe Juli 2010**

Dokument-Nr.: L-1254d\_1

SYS TEC electronic GmbH August-Bebel-Straße 29 D-07973 Greiz  
Telefon: +49 (3661) 6279-0 Telefax: +49 (3661) 6279-99  
Web: <http://www.systec-electronic.com> Mail: [info@systec-electronic.com](mailto:info@systec-electronic.com)

## Status/Änderungen

Status: Freigegeben

<b>Datum/Version</b>	<b>Abschnitt</b>	<b>Änderung</b>	<b>Bearbeiter</b>
2010/07/15 1.0	alle	Erstellung	R. Sieber

Im Buch verwendete Bezeichnungen für Erzeugnisse, die zugleich ein eingetragenes Warenzeichen darstellen, wurden nicht besonders gekennzeichnet. Das Fehlen der © Markierung ist demzufolge nicht gleichbedeutend mit der Tatsache, dass die Bezeichnung als freier Warenname gilt. Ebenso wenig kann anhand der verwendeten Bezeichnung auf eventuell vorliegende Patente oder einen Gebrauchsmusterschutz geschlossen werden.

Die Informationen in diesem Handbuch wurden sorgfältig überprüft und können als zutreffend angenommen werden. Dennoch sei ausdrücklich darauf verwiesen, dass die Firma SYS TEC electronic GmbH weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf den Gebrauch oder den Inhalt dieses Handbuchs zurückzuführen sind. Die in diesem Handbuch enthaltenen Angaben können ohne vorherige Ankündigung geändert werden. Die Firma SYS TEC electronic GmbH geht damit keinerlei Verpflichtungen ein.

Ferner sei ausdrücklich darauf verwiesen, dass SYS TEC electronic GmbH weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf falschen Gebrauch oder falschen Einsatz der Hard- bzw. Software zurückzuführen sind. Ebenso können ohne vorherige Ankündigung Layout oder Design der Hardware geändert werden. SYS TEC electronic GmbH geht damit keinerlei Verpflichtungen ein.

© Copyright 2010 SYS TEC electronic GmbH, D-07973 Greiz.  
 Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form ohne schriftliche Genehmigung der Firma SYS TEC electronic GmbH unter Einsatz entsprechender Systeme reproduziert, verarbeitet, vervielfältigt oder verbreitet werden.

Informieren Sie sich:

Kontakt	Direkt	Ihr Lokaler Distributor
Adresse:	SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz GERMANY	Sie finden eine Liste unserer Distributoren unter  <a href="http://www.systec-electronic.com/distributors">http://www.systec-electronic.com/distributors</a>
Angebots-Hotline:	+49 (0) 36 61 / 62 79-0 <a href="mailto:info@systec-electronic.com">info@systec-electronic.com</a>	
Technische Hotline:	+49 (0) 36 61 / 62 79-0 <a href="mailto:support@systec-electronic.com">support@systec-electronic.com</a>	
Fax:	+49 (0) 36 61 / 6 79 99	
Webseite:	<a href="http://www.systec-electronic.com">http://www.systec-electronic.com</a>	

1. Auflage Juli 2010

Inhalt

<b>1</b>	<b>Einleitung</b> .....	<b>5</b>
<b>2</b>	<b>Übersicht / Wo finde ich was?</b> .....	<b>6</b>
<b>3</b>	<b>Produktbeschreibung</b> .....	<b>8</b>
<b>4</b>	<b>Development Kit PLCcore-9G20</b> .....	<b>10</b>
4.1	Übersicht.....	10
4.2	Elektrische Inbetriebnahme des Development Kit PLCcore-9G20 .....	11
4.3	Bedienelemente des Development Kit PLCcore-9G20 .....	12
4.4	Optionales Zubehör .....	13
4.4.1	USB-RS232 Adapter Kabel .....	13
4.4.2	Driver Development Kit (DDK).....	13
<b>5</b>	<b>Anschlussbelegung des PLCcore-9G20</b> .....	<b>14</b>
<b>6</b>	<b>SPS-Funktionalität des PLCcore-9G20</b> .....	<b>17</b>
6.1	Übersicht.....	17
6.2	Systemstart des PLCcore-9G20.....	17
6.3	Programmierung des PLCcore-9G20 .....	18
6.4	Prozessabbild des PLCcore-9G20 .....	19
6.4.1	Lokale Ein- und Ausgänge.....	19
6.4.2	Ein- und Ausgänge anwenderspezifischer Baseboards.....	20
6.5	Kommunikationsschnittstellen .....	21
6.5.1	Serielle Schnittstellen .....	21
6.5.2	CAN-Schnittstellen.....	21
6.5.3	Ethernet-Schnittstellen.....	21
6.6	Spezifische Peripherieschnittstellen .....	22
6.6.1	Zählereingänge.....	22
6.6.2	Pulsausgänge .....	22
6.7	Bedien- und Anzeigeelemente .....	23
6.7.1	Run/Stop-Schalter .....	23
6.7.2	Run-LED (grün) .....	23
6.7.3	Error-LED (rot).....	24
6.8	Lokales Löschen des SPS-Programms.....	24
6.9	Nutzung der CAN-Schnittstellen mit CANopen .....	25
6.9.1	CAN-Schnittstelle CAN0 .....	26
6.9.2	Zusätzliche CAN-Schnittstellen .....	26
<b>7</b>	<b>Konfiguration und Administration des PLCcore-9G20</b> .....	<b>28</b>
7.1	Systemvoraussetzungen und erforderliche Softwaretools .....	28
7.2	Linux-Autostart aktivieren bzw. deaktivieren .....	29
7.3	Ethernet-Konfiguration des PLCcore-9G20.....	32
7.4	SPS-Konfiguration des PLCcore-9G20 .....	33
7.4.1	SPS-Konfiguration über WEB-Frontend .....	33
7.4.2	SPS-Konfiguration über Bedienelemente des Development Kit PLCcore-9G20 ...	35
7.4.3	Aufbau der Konfigurationsdatei "plccore-9g20.cfg" .....	36
7.5	Boot-Konfiguration des PLCcore-9G20.....	37
7.6	Auswahl der zu startenden Firmware-Variante .....	38
7.7	Vordefinierte Nutzerkonten.....	39
7.8	Anmeldung am PLCcore-9G20 .....	40
7.8.1	Anmeldung an der Kommando-Shell.....	40
7.8.2	Anmeldung am FTP-Server .....	41
7.9	Anlegen und Löschen von Nutzerkonten .....	43
7.10	Passwort eines Nutzerkontos ändern.....	43
7.11	Setzen der Systemzeit.....	44
7.12	Dateisystem des PLCcore-9G20 .....	45

7.13	Software-Update des PLCcore-9G20.....	46
7.13.1	Update der SPS-Firmware.....	46
7.13.2	Update des Linux-Images.....	48
<b>8</b>	<b>Adaption von Ein-/Ausgängen sowie Prozessabbild.....</b>	<b>51</b>
8.1	Datenaustausch über Shared Prozessabbild .....	51
8.1.1	Übersicht zum Shared Prozessabbild .....	51
8.1.2	API des Shared Prozessabbild Client.....	55
8.1.3	Erstellen einer anwenderspezifischen Client Applikation .....	58
8.1.4	Beispiel zur Nutzung des Shared Prozessabbild .....	61
8.2	Driver Development Kit (DDK) für das PLCcore-9G20 .....	64
8.3	Testen der Hardwareanschaltung .....	66
<b>Anhang A: Firmware-Funktionsumfang des PLCcore-9G20 .....</b>		<b>68</b>
<b>Anhang B: Referenzdesigns zum PLCcore-9G20 .....</b>		<b>71</b>
<b>Anhang C: GNU GENERAL PUBLIC LICENSE.....</b>		<b>74</b>
<b>Index.....</b>		<b>79</b>

# 1 Einleitung

Vielen Dank, dass Sie sich für das SYS TEC PLCcore-9G20 entschieden haben. Mit diesem Produkt verfügen Sie über einen innovativen und leistungsfähigen SPS-Kern. Aufgrund seiner hohen Performance auf besonders kleiner Baugröße sowie wegen seiner geringen Leistungsaufnahme eignet er sich besonders gut als Kommunikations- und Steuerrechner für Embedded Anwendungen.

Bitte nehmen Sie sich etwas Zeit, dieses Manual aufmerksam zu lesen. Es beinhaltet wichtige Informationen zur Inbetriebnahme, Konfiguration und Programmierung des PLCcore-9G20. Es wird Ihnen helfen, sich mit dem Funktionsumfang und der Anwendung des PLCcore-9G20 vertraut zu machen. Dieses Dokument wird ergänzt durch weitere Manuals, wie beispielsweise zum IEC 61131-Programmiersystem *OpenPCS* und zur CANopen-Erweiterung für IEC 61131-3. Eine Auflistung der relevanten Manuals zum PLCcore-9G20 beinhaltet Tabelle 1 im Abschnitt 2. Bitte beachten Sie auch diese ergänzenden Dokumentationen.

Für weiter führende Informationen, Zusatzprodukte, Updates usw. empfehlen wir den Besuch unserer Website unter: <http://www.systec-electronic.com>. Der Inhalt dieser Webseite wird periodisch aktualisiert und stellt Ihnen stets die neuesten Software-Releases und Manual-Versionen zum Download bereit.

## Anmerkungen zum EMV-Gesetz für das PLCcore-9G20



Das PLCcore-9G20 ist als Zulieferteil für den Einbau in ein Gerät (Weiterverarbeitung durch Industrie) bzw. als Entwicklungsboard für den Laborbetrieb (zur Hardware- und Softwareentwicklung) bestimmt.

Nach dem Einbau in ein Gerät oder bei Änderungen/Erweiterungen an diesem Produkt muss die Konformität nach dem EMV-Gesetz neu festgestellt und bescheinigt werden. Erst danach dürfen solche Geräte in Verkehr gebracht werden.

Die CE-Konformität gilt nur für den hier beschriebenen Anwendungsbereich unter Einhaltung der im folgenden Handbuch gegebenen Hinweise zur Inbetriebnahme! Das PLCcore-9G20 ist ESD empfindlich und darf nur an ESD geschützten Arbeitsplätzen von geschultem Fachpersonal ausgepackt und gehandhabt bzw. betrieben werden.

Das PLCcore-9G20 ist ein Modul für den Bereich Automatisierungstechnik. Durch die Programmierbarkeit nach IEC 61131-3 und die Verwendung von CAN-Bus und Ethernet-Standard-Netzwerkschnittstelle für verschiedenste Automatisierungslösungen, sowie dem standardisierten Netzwerkprotokoll CANopen ergeben sich geringere Entwicklungszeiten bei günstigen Kosten der Hardware. Durch die on-board Realisierung der SPS-Funktionalität mit optionaler CANopen-Netzwerkschicht ist die Erstellung einer Firmware durch den Anwender überflüssig geworden.

## 2 Übersicht / Wo finde ich was?

Das PLCcore-9G20 basiert auf der Hardware des ECUcore-9G20 und erweitert dieses um SPS-spezifische Funktionalitäten (FPGA-Software, SPS-Firmware). Für die Hardwarekomponenten wie das ECUcore-9G20 bzw. das PLCcore-9G20 selbst (die Hardware beider Module ist identisch), die Developmentboards sowie Referenzschaltungen existieren eigene Hardware-Manuals. Softwareseitig wird das PLCcore-9G20 mit der IEC 61131-3 konformen Programmierumgebung *OpenPCS* programmiert. Zu *OpenPCS* existieren ebenfalls eigene Handbücher, die den Umgang mit dem Programmierwerkzeug und SYS TEC-spezifische Erweiterungen dazu beschreiben. Diese sind Bestandteil des Software-Paketes "*OpenPCS*". Tabelle 1 listet die für das PLCcore-9G20 relevanten Manuals auf.

Tabelle 1: Übersicht relevanter Manuals zum PLCcore-9G20

Informationen über...	In welchem Manual?
Grundlegende Informationen zum PLCcore-9G20 (Konfiguration, Administration, Prozessabbild, Anschlussbelegung, Firmwareupdate, Referenzdesigns usw.)	In diesem Manual
Entwicklung anwenderspezifischer C/C++ Applikationen für das ECUcore-9G20 / PLCcore-9G20, VMware-Image des Linux-Entwicklungssystems	System Manual ECUcore-9G20 (Manual-Nr.: L-1253)
Hardware-Beschreibung zum ECUcore-9G20 / PLCcore-9G20, Referenzdesigns usw.	Hardware Manual ECUcore-9G20 (Manual-Nr.: L-1255)
Developmentboard zum ECUcore-9G20 / PLCcore-9G20, Referenzdesigns usw.	Hardware Manual Developmentboard 9G20 (Manual-Nr.: L-1256)
Driver Development Kit (DDK) für das ECUcore-9G20	Software Manual Driver Development Kit (DDK) für ECUcore-9G20 (Manual-Nr.: L-1257)
Grundlagen zum IEC 61131-Programmiersystem <i>OpenPCS</i>	Kurzanleitung Programmiersystem (Eintrag " <i>OpenPCS Dokumentation</i> " in der <i>OpenPCS</i> -Programmgruppe des Startmenüs) (Manual-Nr.: L-1005)
Vollständige Beschreibung zum IEC 61131-Programmiersystem <i>OpenPCS</i> , Grundlagen der SPS-Programmierung nach IEC 61131-3	Online-Hilfe zum <i>OpenPCS</i> -Programmiersystem
Befehlsübersicht und Beschreibung der Standard-Funktionsbausteine nach IEC 61131-3	Online-Hilfe zum <i>OpenPCS</i> -Programmiersystem
SYS TEC-Erweiterung für IEC 61131-3: - Stringfunktionen - UDP-Funktionsbausteine - SIO-Funktionsbausteine - FB für RTC, Counter, EEPROM, PWM/PTO	User Manual " <i>SYS TEC spezifische Erweiterungen für OpenPCS / IEC 61131-3</i> " (Manual-Nr.: L-1054)

CANopen-Erweiterung für IEC 61131-3 (Netzwerkvariablen, CANopen-Funktionsbausteine)	User Manual "CANopen-Erweiterung für IEC 61131-3" (Manual-Nr.: L-1008)
Lehrbuch zur SPS-Programmierung nach IEC 61131-3	SPS-Programmierung mit IEC 61131-3 John/Tiegelkamp Springer-Verlag ISBN: 3-540-66445-9 (in gekürzter Form auch als PDF auf <i>OpenPCS</i> Installations-CD enthalten)

- Abschnitt 4** dieses Manuals erläutert die **Inbetriebnahme des PLCcore-9G20** auf Basis des Development Kit für das PLCcore-9G20.
- Abschnitt 5** beschreibt die **Anschlussbelegung** des PLCcore-9G20.
- Abschnitt 6** erklärt Details zur **Anwendung des PLCcore-9G20**, so z.B. den **Aufbau des Prozessabbildes**, die **Bedeutung der Bedienelemente** und vermittelt grundlegende Informationen zur Programmierung des Moduls. Weiterhin werden hier Informationen zur Nutzung der CAN-Schnittstellen in Verbindung mit **CANopen** gegeben.
- Abschnitt 7** beschreibt **Details zur Konfiguration des PLCcore-9G20**, so z.B. die Konfiguration von Ethernet- und CAN-Schnittstellen, den Linux-Autostartvorgang sowie die Auswahl der Firmwarevariante. Weiterhin wird hier die **Administration des PLCcore-9G20** erläutert, so z.B. die Anmeldung am System, die Nutzerverwaltung und die Durchführung von Softwareupdates.
- Abschnitt 8** erläutert die **Adaption von Ein- und Ausgängen** sowie des **Prozessabbildes** und behandelt schwerpunktmäßig des Datenaustausches zwischen einem SPS-Programm und einer anwenderspezifischen C/C++ Applikation über das **Shared Prozessabbild**.

### 3 Produktbeschreibung

Das PLCcore-9G20 erweitert die Produktpalette der Firma SYS TEC electronic GmbH im Steuerungsbereich um ein weiteres innovatives Produkt. In Form eines Aufsteckmoduls ("Core") stellt es dem Anwender eine vollständige Kompakt-SPS zur Verfügung. Dank der CAN- und Ethernet-Schnittstellen ist das PLCcore-9G20 optimal zur Realisierung dezentraler Steuerungsaufgaben geeignet.

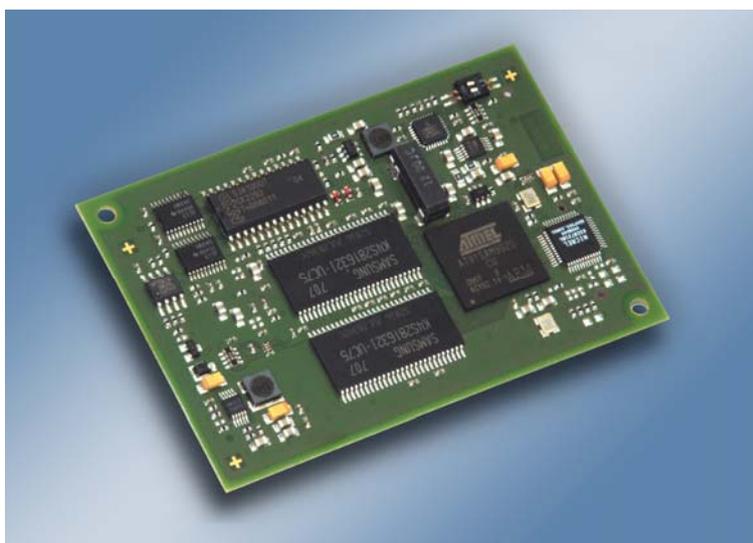


Bild 1: Ansicht des PLCcore-9G20

Einige herausragende Merkmale des PLCcore-9G20 sind:

- leistungsfähiger CPU-Kern (Atmel 32-Bit AT91SAM9G20, 400 MHz CPU Takt, 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 (mit on-board PHY)
- 1x CAN 2.0B Interface, nutzbar als CANopen-Manager (CiA 302 konform)
- 5x Asynchronous Serial Ports (UART)
- 19 digitale Eingänge, 8 digitale Ausgänge (Standardkonfiguration, modifizierbar über DDK)
- 3 analoge Eingänge (ADC)
- 4 High-speed Counter (Pulse/Dir oder A/B)
- 4 PWM-/PTO-Ausgang (Pulse/Dir)
- SPI und I<sup>2</sup>C extern nutzbar
- On-board Peripherie: RTC, Temperatursensor
- On-board Software: Linux, SPS-Firmware, CANopen-Master, HTTP- und FTP-Server
- Programmierbar nach IEC 61131-3 und in C/C++
- Funktionsbausteinbibliotheken für Kommunikation (CANopen, Ethernet und UART)
- Funktionsbausteinbibliotheken für Hardwarekomponenten (RTC, Counter, PWM/PTO)
- Support SPS-typischer Bedienelemente (z.B. Run/Stop-Schalter, Run-LED, Error-LED)
- Linux-basiert, dadurch weitere beliebige Anwenderprogramme parallel ausführbar
- Einfache, HTML-basierte Konfiguration über WEB-Browser
- Remote Login über Telnet
- Geringe Abmaße (78 \* 54 mm),

Für das PLCcore-9G20 sind verschiedene Firmware-Varianten verfügbar, die sich in dem zur Kommunikation zwischen Programmier-PC und PLCcore-9G20 verwendeten Protokoll unterscheiden:

- Art.-Nr.: 3390024: PLCcore-9G20/Z4 (CANopen)  
Kommunikation mit Programmier-PC via CANopen-Protokoll  
(Interface CAN0)
- Art.-Nr.: 3390025: PLCcore-9G20/Z5 (Ethernet)  
Kommunikation mit Programmier-PC via UDP-Protokoll  
(Interface ETH0)

Die Verfügbarkeit einer SPS als aufsteckbares "Core" und die damit verbundenen geringen Abmessungen reduzieren den Aufwand und die Kosten bei der Entwicklung anwenderspezifischer Steuerungen enorm. Gleichzeitig eignet sich das PLCcore-9G20 besonders als intelligenter Netzwerkknoten zur dezentralen Verarbeitung von Prozesssignalen (CANopen und UDP), als Basiskomponente von Spezialbaugruppen und als SPS in schwer zugänglichen Bereichen.

Die On-Board-Firmware des PLCcore-9G20 beinhaltet die gesamte SPS-Laufzeitumgebung einschließlich der CANopen-Anbindung mit CANopen-Masterfunktionalität. Dadurch ist das Modul in der Lage, Steuerungsaufgaben wie die Verknüpfung von Ein- und Ausgängen oder die Umsetzung von Regelalgorithmen vor Ort durchzuführen. Über das CANopen-Netzwerk, über Ethernet (UDP-Protokoll) sowie über die seriellen Schnittstellen (UART) können Daten und Ereignisse mit anderen Knoten (z.B. übergeordnete Zentralsteuerung, I/O-Slaves usw.) ausgetauscht werden. Darüber hinaus ist die Anzahl der Ein- und Ausgänge sowohl lokal als auch dezentral mit Hilfe von CANopen-Geräten erweiterbar. Ideal geeignet ist hierfür der CANopen-Chip, der ebenfalls als Aufsteckmodul für den Einsatz in anwenderspezifischen Applikationen konzipiert wurde.

Das PLCcore-9G20 bietet 19 digitale Eingänge (DI0...DI18, 3.3V-Pegel), 8 digitale Ausgänge (DO0...DO7, 3.3V-Pegel), 4 High-Speed Counter-Eingänge sowie 4 PWM/PTO-Ausgänge. Mit Hilfe des Driver Development Kit (SO-1106) kann diese Standard-I/O-Konfiguration an die spezifischen Applikationsbedingungen adaptiert werden (siehe Abschnitte 4.4.2 und 8.2). Die Ablage des SPS-Programms in der On-board Flash-Disk des Moduls ermöglicht den autarken Wiederanlauf nach einer Spannungsunterbrechung.

Die Programmierung des PLCcore-9G20 erfolgt nach IEC 61131-3 mit dem Programmiersystem *OpenPCS* der Firma infoteam Software GmbH (<http://www.infoteam.de>). Dieses Programmiersystem wurde von der Firma SYS TEC electronic GmbH für das PLCcore-9G20 erweitert und angepasst. Damit kann das PLCcore-9G20 sowohl grafisch in KOP/FUB, AS und CFC als auch textuell in AWL oder ST programmiert werden. Der Download des SPS-Programms auf das Modul erfolgt in Abhängigkeit von der verwendeten Firmware über Ethernet oder CANopen. Die Adressierung der Ein- und Ausgänge – und damit der Aufbau des Prozessabbildes – wurde an das Schema der SYS TEC-Kompaktsteuerungen angelehnt. Analog zu allen anderen SYS TEC-Steuerungen unterstützt auch das PLCcore-9G20 die Rückdokumentation des SPS-Programms aus der Steuerung sowie als Debugfunktionalität das Beobachten und Setzen von Variablen, Einzelzyklus, Breakpoints und Einzelschritt.

Das PLCcore-9G20 basiert auf einem Embedded Linux als Betriebssystem. Dadurch ist es möglich, simultan zur SPS-Firmware noch weitere, anwenderspezifische Programme abzuarbeiten. Falls erforderlich, können diese anwenderspezifischen Programme unter Nutzung des Prozessabbildes Daten mit dem SPS-Programm austauschen. Weitere Informationen hierzu beinhaltet der Abschnitt 8.

Das auf dem PLCcore-9G20 eingesetzte Embedded Linux ist unter der GNU General Public License, Version 2 lizenziert. Der entsprechende Lizenztext ist im Anhang C aufgeführt. Die vollständigen Quellen des verwendeten LinuxBSP sind im Softwarepaket **SO-1105** ("VMware-Image des Linux-Entwicklungssystems für das ECUCore-9G20") enthalten. Sollten Sie die Quellen des LinuxBSP unabhängig vom VMware-Image des Linux-Entwicklungssystems benötigen, setzen Sie sich bitte mit unserem Support in Verbindung:

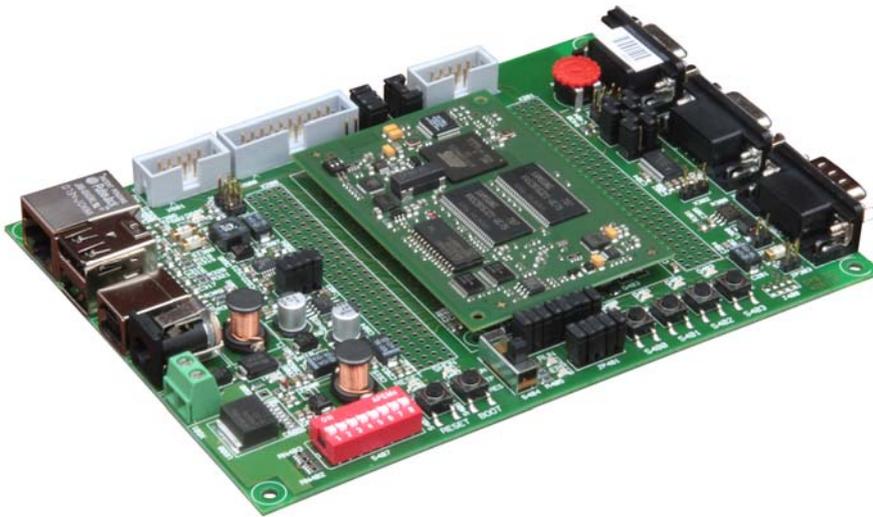
[support@systemec-electronic.com](mailto:support@systemec-electronic.com)

Das verwendete SPS-System sowie die vom Anwender entwickelten SPS- und C/C++ Programme unterliegen **nicht** der GNU General Public License!

## 4 Development Kit PLCcore-9G20

### 4.1 Übersicht

Das Development Kit PLCcore-9G20 ist ein leistungsfähiges Komplettpaket zu einem besonders günstigen Preis, um auf Basis einer Kompakt-SPS dezentrale, netzwerkfähige Automatisierungsprojekte zu realisieren oder um die zahlreichen Vorzüge der grafischen und textuellen SPS-Programmierung nach IEC 61131-3 gegenüber konventionellen Programmiersprachen kennen zu lernen.



*Bild 2: Development Kit PLCcore-9G20*

Das Development Kit PLCcore-9G20 gewährleistet eine rasche und problemlose Inbetriebnahme des PLCcore-9G20. Es vereint dazu alle notwendigen Hard- und Software-Komponenten, die zur Erstellung eigener Applikationen erforderlich sind: als Kernkomponente das PLCcore-9G20 selbst, das zugehörige Developmentboard mit I/O-Peripherie und zahlreichen Schnittstellen, das IEC 61131-Programmiersystem *OpenPCS* sowie weiteres Zubehör. Damit bildet das Developmentkit die ideale Plattform zur Entwicklung anwenderspezifischer Applikationen auf Basis des PLCcore-9G20. Ebenso ermöglicht das Kit den kostengünstigen Einstieg in die Welt der dezentralen Automatisierungstechnik. Bereits die im Kit enthaltenen Komponenten ermöglichen die Erweiterung der Ein- und Ausgänge des PLCcore-9G20 durch CANopen-I/O-Baugruppen. Damit kann das Development Kit auch für Projekte eingesetzt werden, die eine SPS mit Netzwerkanbindung erfordern.

Das Development Kit PLCcore-9G20 beinhaltet folgende Komponenten:

- PLCcore-9G20
- Developmentboard für PLCcore-9G20
- 24V Steckernetzteil
- Ethernet-Kabel
- RS232-Kabel
- CD mit Programmiersoftware, Beispielen, Dokumentation und weiteren Tools

Das im Kit enthaltene Developmentboard ermöglicht eine schnelle Inbetriebnahme des PLCcore-9G20 und vereinfacht den Aufbau von Prototypen anwenderspezifischer Applikationen, die auf diesem Modul basieren. Das Developmentboard besitzt u.a. verschiedene Möglichkeiten der Spannungszufuhr, Ethernet-Schnittstelle, CAN-Schnittstelle, 4 Taster und 4 LEDs als Bedienelemente für die digitalen Ein- und Ausgänge sowie ein Potentiometer für den analogen Eingang. Die an den Steckverbindern des PLCcore-9G20 verfügbaren Signale sind auf Stiftleisten geführt und ermöglichen

so einen einfachen Anschluss eigener Peripherie-Schaltungen. Damit bildet das Developmentboard gleichzeitig eine ideale Experimentier- und Testplattform für das PLCcore-9G20.

Als Software-Entwicklungsplattform sowie Debugumgebung für das PLCcore-9G20 dient das im Kit enthaltene IEC 61131-Programmiersystem *OpenPCS*. Dadurch kann das Modul sowohl grafisch in KOP/FUB, AS und CFC als auch textuell in AWL oder ST programmiert werden. Der Download des SPS-Programms auf das PLCcore-9G20 erfolgt in Abhängigkeit von der verwendeten Firmware über Ethernet oder CANopen. Leistungsfähige Debug-Funktionalitäten wie das Beobachten und Setzen von Variablen, Einzelzyklus, Breakpoints und Einzelschritt erleichtern die Entwicklung und Inbetriebnahme von Anwendersoftware für das Modul.

## 4.2 Elektrische Inbetriebnahme des Development Kit PLCcore-9G20

Das für den Betrieb des Development Kit PLCcore-9G20 notwendige Steckernetzteil sowie die erforderlichen Ethernet- und RS232-Kabel sind bereits im Lieferumfang des Kit enthalten. Für die Inbetriebnahme des Kit sind mindestens die Anschlüsse für Versorgungsspannung (X600/X601), COM0 (X301) und ETH0 (X304) erforderlich. Darüber hinaus wird der Anschluss von CAN0 (X303) empfohlen. Einen vollständigen Überblick über die Anschlüsse des Development Kit PLCcore-9G20 vermittelt Tabelle 2.

Tabelle 2: Anschlüsse des Development Kit PLCcore-9G20

Anschluss	Bezeichnung auf Developmentboard	Bemerkung
Versorgungsspannung	X600 oder X601	Das im Lieferumfang enthaltene Steckernetzteil ist zum direkten Anschluss an X700 vorgesehen
ETH0 (Ethernet)	X304	Schnittstelle dient zur Kommunikation mit Programmier-PC sowie zum Programmdownload (PLCcore-9G20/Z5, Bestellnummer 3390025), darüber hinaus für das Anwenderprogramm frei verfügbar
COM0 (RS232)	X301	Schnittstelle wird zur Konfiguration der Baugruppe (z.B. Setzen der IP-Adresse) benötigt und ist im normalen Betrieb für das Anwenderprogramm frei verfügbar
COM1 (RS232)	X300	Schnittstelle ist für das Anwenderprogramm frei verfügbar
COM2 (RS232)	X302n	Schnittstelle ist für das Anwenderprogramm frei verfügbar
CAN0 (CAN)	X303	Schnittstelle dient zur Kommunikation mit Programmier-PC sowie zum Programmdownload (PLCcore-9G20/Z4, Bestellnummer 3390024), darüber hinaus für das Anwenderprogramm frei verfügbar

Bild 3 zeigt die Lage der wichtigsten Anschlüsse auf dem Developmentboard für das PLCcore-9G20. Anstelle des mitgelieferten Steckernetzteiles kann die Stromversorgung des Kit optional auch über X601 mit einer externen Quelle von 24V/1A erfolgen.

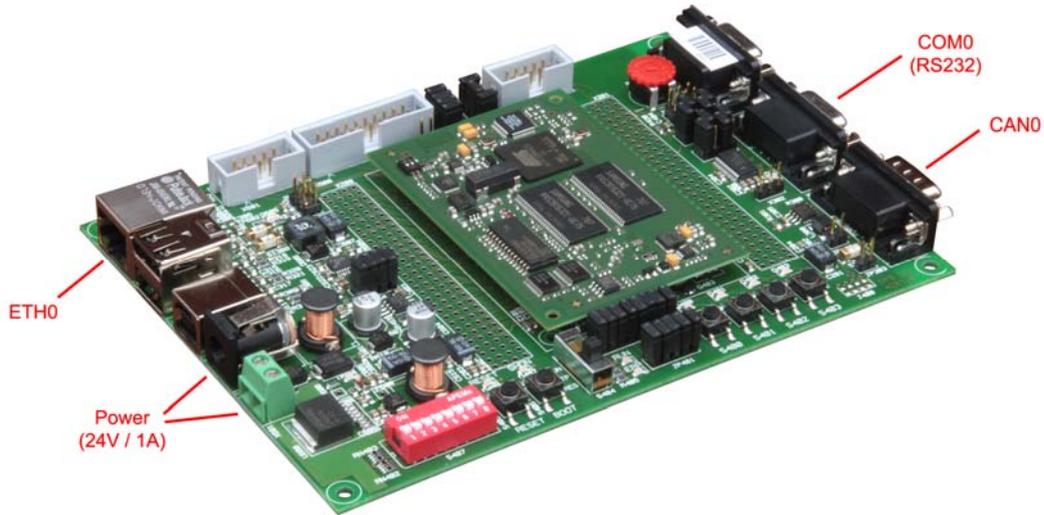


Bild 3: Lage der wichtigsten Anschlüsse auf dem Developmentboard für das PLCcore-9G20

**Hinweis:** Bei der Inbetriebnahme sind zunächst die Kabel für Ethernet (ETH0, X304) und RS232 (COM0, X301) anzuschließen, erst danach ist die Versorgungsspannung (X600 / X601) zu aktivieren.

### 4.3 Bedienelemente des Development Kit PLCcore-9G20

Das Development Kit PLCcore-9G20 ermöglicht eine einfache Inbetriebnahme des PLCcore-9G20. Es verfügt über verschiedene Bedienelemente sowohl zur Konfiguration des Moduls als auch zur Simulation von Ein- und Ausgängen für den Einsatz des PLCcore-9G20 als SPS-Kern. Tabelle 3 listet die einzelnen Bedienelemente des Developmentboard auf und beschreibt deren Bedeutung.

Tabelle 3: Bedienelemente des Developmentboard für das PLCcore-9G20

Bedienelement	Bezeichnung	Bedeutung
Taster 0	S400	Digitaler Eingang DI0 (Prozessabbild: %IX0.0)
Taster 1	S401	Digitaler Eingang DI1 (Prozessabbild: %IX0.1)
Taster 2	S402	Digitaler Eingang DI2 (Prozessabbild: %IX0.2)
Taster 3	S403	Digitaler Eingang DI3 (Prozessabbild: %IX0.3)
LED 0	D400	Digitaler Ausgang DO0 (Prozessabbild: %QX0.0)
LED 1	D401	Digitaler Ausgang DO1 (Prozessabbild: %QX0.1)
LED 2	D402	Digitaler Ausgang DO2 (Prozessabbild: %QX0.2)
LED 3	D403	Digitaler Ausgang DO3 (Prozessabbild: %QX0.3)
Poti (ADC)	R429	Analoger Eingang AI0 (Prozessabbild: %IW8.0)
Run/Stop-Schalter	S404	Run / Stop für Abarbeitung des SPS-Programms, Rücksetzen der Steuerung (siehe Abschnitt 6.7.1)

Run-LED	D405	Anzeige Aktivitätszustand der SPS (siehe Abschnitt 6.7.2)
Error-LED	D406	Anzeige Fehlerzustand der SPS (siehe Abschnitt 6.7.3)
DIP-Schalter	S407	Konfiguration Bitrate und Master-Modus CAN0, siehe Abschnitt 7.4.2

Eine vollständige Auflistung des Prozessabbildes enthält Tabelle 8 im Abschnitt 6.4.1.

## 4.4 Optionales Zubehör

### 4.4.1 USB-RS232 Adapter Kabel

Das SYS TEC USB-RS232 Adapter Kabel (Bestellnummer 3234000) stellt eine RS232-Schnittstelle über einen USB-Port des PC zur Verfügung. In Verbindung mit einem Terminalprogramm ermöglicht es die Konfiguration des PLCcore-9G20 von PCs, wie z.B. Laptops, die selber keine physikalische RS232-Schnittstelle mehr besitzen (siehe Abschnitt 7.1).



*Bild 4: SYS TEC USB-RS232 Adapter Kabel*

### 4.4.2 Driver Development Kit (DDK)

Das Driver Development Kit für das ECUcore-9G20 (Bestellnummer SO-1106) ermöglicht dem Anwender die eigenständige Anpassung der I/O-Ebene an ein selbst entwickeltes Baseboard. Einen Überblick zum Driver Development Kit vermittelt Abschnitt 8.2.

## 5 Anschlussbelegung des PLCcore-9G20

Die Anschlüsse des PLCcore-9G20 sind über zwei auf der Modulunterseite montierte, doppelreihige Buchsenleisten nach außen geführt (X500, siehe Bild 5). Die entsprechenden Steckverbinder als Gegenstücke zur Aufnahme des PLCcore-9G20 sind bei der Firma "W + P" unter folgender Bezeichnung lieferbar:

W+P-Bezeichnung: SMT-Stiftleisten RM 1,27mm, stehend, 2-reihig - 1,00mm  
 W+P-Bestellnummer: 7072-100-10-00-10-PPST (auch in anderen Höhen lieferbar)

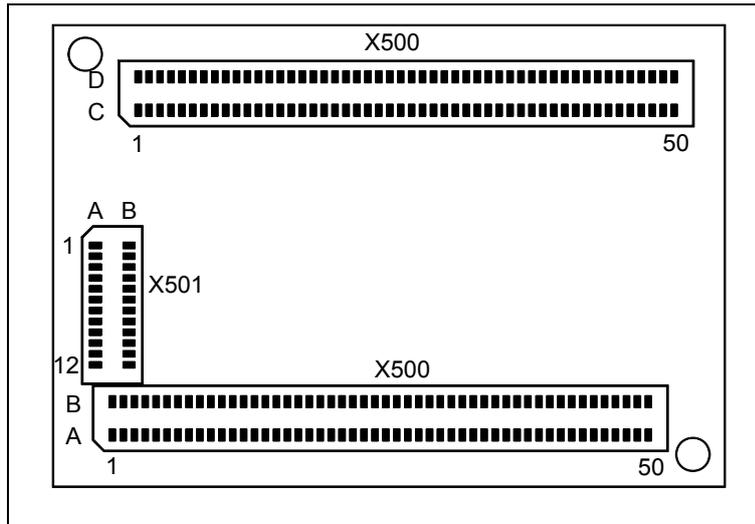


Bild 5: Pinout des PLCcore-9G20 - top view

Bild 5 verdeutlicht die Lage der Buchsenleisten (X500) auf dem PLCcore-9G20, Tabelle 4 listet die vollständige Anschlussbelegung des Moduls auf. Die im Bild 5 zusätzlich aufgeführte Buchsenleiste X501 dient zum Anschluss eines JTAG-Interfaces und ist nur auf speziellen Entwicklungsmodulen bestückt. Für die Verwendung des PLCcore-9G20 als SPS-Kern ist das JTAG-Interface ohne Bedeutung. Eine ausführliche Beschreibung der Modulanschlüsse beinhaltet das Hardware Manual ECUCore-9G20 (Manual-Nr.: L-1255). Referenzdesigns zur Anwendung des PLCcore-9G20 in anwenderspezifischen Applikationen enthält Anhang B.

Tabelle 4: Anschlussbelegung des PLCcore-9G20, vollständig, sortiert nach Anschluss-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

Signal	Pin	Pin	Signal	Signal	Pin	Pin	Signal
RXD0	A14	B14	RXD1	SD_MCCK	C14	D14	SD_MCCDA
TXD2	A15	B15	TXD3	GND	C15	D15	SD_MCCDB
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

Tabelle 5 beinhaltet als Untermenge von Tabelle 4 nur alle Ein- und Ausgänge des PLCcore-9G20, sortiert nach deren Funktion.

Tabelle 5: Anschlussbelegung des PLCcore-9G20, nur I/O, sortiert nach Funktion

Connector	I/O-Pin (FPGA)	PLC Function 1	PLC Function 2 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	

Tabelle 6 definiert die Codierung des Run/Stop-Schalters. Die Funktion des Run/Stop-Schalters für die SPS-Firmware erläutert Abschnitt 6.7. 1. Ist für den Einsatz des PLCcore-9G20 auf einer anwenderspezifischen Basisplatine kein Run/Stop-Schalter vorgesehen, muss an den Modulanschlüssen die Codierung für "Run" fest verdrahtet sein (siehe dazu auch Referenzdesign im Anhang B).

Tabelle 6: Codierung des Run/Stop-Schalters

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

## 6 SPS-Funktionalität des PLCcore-9G20

### 6.1 Übersicht

Das PLCcore-9G20 realisiert eine vollständige, Linux-basierte Kompakt-SPS in Form eines Aufsteckmoduls ("Core"). Das PLCcore-9G20 basiert dabei auf der Hardware des ECUcore-9G20 und erweitert dieses um SPS-spezifische Funktionalitäten (FPGA-Software, SPS-Firmware). Beide Module, sowohl ECUcore-9G20 als auch PLCcore-9G20, benutzt dasselbe Embedded Linux als Betriebssystem. Folglich sind auch Konfiguration und C/C++ Programmierung des PLCcore-9G20 weitestgehend identisch zum ECUcore-9G20.

### 6.2 Systemstart des PLCcore-9G20

Standardmäßig lädt das PLCcore-9G20 nach Power-on bzw. Reset alle notwendigen Firmware-Komponenten und startet anschließend die Abarbeitung des SPS-Programms. Damit eignet sich das PLCcore-9G20 für den Einsatz in autarken Steuerungssystemen, die nach einer Spannungsunterbrechung selbständig und ohne Benutzeraktionen die Ausführung des SPS-Programms wieder aufnehmen. Bild 6 zeigt den Systemstart im Detail.

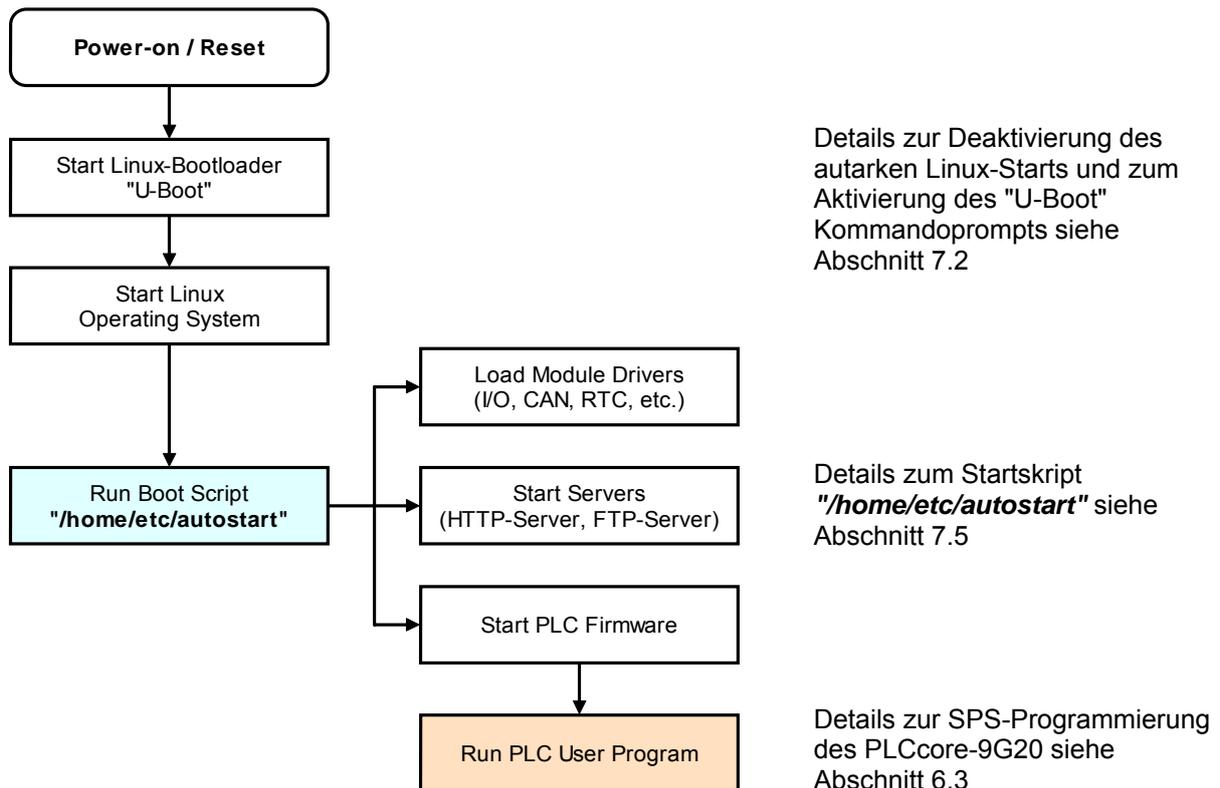


Bild 6: Systemstart des PLCcore-9G20

### 6.3 Programmierung des PLCcore-9G20

Das PLCcore-9G20 wird mit der IEC 61131-3 konformen Programmierumgebung *OpenPCS* programmiert. Zu *OpenPCS* existieren eigene Handbücher, die den Umgang mit diesem Programmierwerkzeug beschreiben. Diese sind Bestandteil des Software-Paketes "*OpenPCS*". Tabelle 1 listet die für das PLCcore-9G20 relevanten Manuals auf.

Die Firmware des PLCcore-9G20 basiert auf der Standardfirmware für SYS TEC-Kompaktsteuerungen und hat daher ein identisches Verhalten zu anderen Steuerungen der Firma SYS TEC. Dies betrifft insbesondere den Aufbau des Prozessabbildes (siehe Abschnitt 6.4) sowie die Funktionsweise der Bedienelemente (Hexcodier-Schalter, DIP-Schalter, Run/Stop-Schalter, Run-LED, Error-LED).

Die Firmware des PLCcore-9G20 stellt dem SPS-Programmierer in Abhängigkeit der eingesetzten Firmware-Variante zahlreiche Funktionsbausteine für den Zugriff auf Kommunikationsschnittstellen bereit. Tabelle 7 listet die Verfügbarkeit der Kommunikations-FB-Klassen (SIO, CAN, UDP) für die einzelnen Firmware-Typen des PLCcore-9G20 auf. Abschnitt 7.6 beschreibt die Auswahl der jeweils zu startenden Firmware-Variante.

*Tabelle 7: Unterstützung von Kommunikations-FB-Klassen für verschiedene PLCcore Typen*

Interface Typ	PLCcore-9G20/Z3 Art.-Nr.: 3390003	PLCcore-9G20/Z4 Art.-Nr.: 3390004	PLCcore-9G20/Z5 Art.-Nr.: 3390005	Bemerkung
CAN	-	x	x	FB-Beschreibung siehe Manual L-1008
UDP	-	x	x	FB-Beschreibung siehe Manual L-1054
SIO	x	x	x	FB-Beschreibung siehe Manual L-1054

Eine vollständige Auflistung der vom PLCcore-9G20 unterstützten Firmware-Funktionen und -Funktionsbausteine enthält Tabelle 22 im Anhang A.

Detaillierte Informationen zu Nutzung der CAN-Schnittstellen in Verbindung mit CANopen beschreibt Abschnitt 6.9.

## 6.4 Prozessabbild des PLCcore-9G20

### 6.4.1 Lokale Ein- und Ausgänge

Das PLCcore-9G20 besitzt ein Prozessabbild mit identischen Adressen im Vergleich zu anderen SYS TEC-Kompaktsteuerungen. Vom PLCcore-9G20 werden die in Tabelle 8 aufgelisteten Ein- und Ausgänge unterstützt.

Tabelle 8: Zuordnung der Ein- und Ausgänge zum Prozessabbild des PLCcore-9G20

I/O des PLCcore-9G20	Adresse und Datenformat im Prozessabbild
DI0 ... DI7	<b>%IB0.0</b> als Byte mit DI0 ... DI7 <b>%IX0.0 ... %IX0.7</b> als einzelne Bits für jeden Eingang
DI8 ... DI15	<b>%IB1.0</b> als Byte mit DI8 ... DI15 <b>%IX1.0 ... %IX1.7</b> als einzelne Bits für jeden Eingang
DI16 ... DI23 (DI19 ... DI23 nur als anwenderspezifische Erweiterung)	<b>%IB2.0</b> als Byte mit DI16 ... DI23 <b>%IX2.0 ... %IX2.7</b> als einzelne Bits für jeden Eingang
DI24 ... DI31 (nur als anwenderspezifische Erweiterung)	<b>%IB3.0</b> als Byte mit DI24 ... DI31 <b>%IX3.0 ... %IX3.7</b> als einzelne Bits für jeden Eingang
DI32 ... DI39 (nur als anwenderspezifische Erweiterung)	<b>%IB4.0</b> als Byte mit DI32 ... DI139 <b>%IX4.0 ... %IX4.7</b> als einzelne Bits für jeden Eingang
DI40 ... DI47 (nur als anwenderspezifische Erweiterung)	<b>%IB5.0</b> als Byte mit DI40 ... DI47 <b>%IX5.0 ... %IX5.7</b> als einzelne Bits für jeden Eingang
AI0 (externer ADC des Development-board), siehe <sup>(1)</sup>	<b>%IW8.0</b> 15Bit + Vorzeichen (0 ... +32767)
C0	<b>%ID40.0</b> 31Bit + Vorzeichen ( $-2^{31} - 2^{31} - 1$ ) Zählereingang: DI0, Richtung: DI1, siehe Abschnitt 6.6.1
C1	<b>%ID44.0</b> 31Bit + Vorzeichen ( $-2^{31} - 2^{31} - 1$ ) Zählereingang: DI2, Richtung: DI3, siehe Abschnitt 6.6.1
C2	<b>%ID48.0</b> 31Bit + Vorzeichen ( $-2^{31} - 2^{31} - 1$ ) Zählereingang: DI8, Richtung: DI9, siehe Abschnitt 6.6.1
C3	<b>%ID52.0</b> 31Bit + Vorzeichen ( $-2^{31} - 2^{31} - 1$ ) Zählereingang: DI10, Richtung: DI11, siehe Abschnitt 6.6.1
On-board Temperatursensor, siehe <sup>(1)</sup>	<b>%ID72.0</b> 31Bit + Vorzeichen als 1/10000 °C

DO0 ... DO7	<b>%QB0.0</b> %QX0.0 ... %QX0.7	als Byte mit DO0 ... DO7 als einzelne Bits für jeden Ausgang
DO8 ... DO15 (nur als anwenderspezifische Erweiterung)	<b>%QB1.0</b> %QX1.0 ... %QX1.7	als Byte mit DO8 ... DO15 als einzelne Bits für jeden Ausgang
DO16 ... DO23 (nur als anwenderspezifische Erweiterung)	<b>%QB2.0</b> %QX2.0 ... %QX2.7	als Byte mit DO16 ... DO23 als einzelne Bits für jeden Ausgang
DO24 ... DO31 (nur als anwenderspezifische Erweiterung)	<b>%QB3.0</b> %QX3.0 ... %QX3.7	als Byte mit DO24 ... DO31 als einzelne Bits für jeden Ausgang
DO32 ... DO39 (nur als anwenderspezifische Erweiterung)	<b>%QB4.0</b> %QX4.0 ... %QX4.7	als Byte mit DO32 ... DO39 als einzelne Bits für jeden Ausgang
DO40 ... DO47 (nur als anwenderspezifische Erweiterung)	<b>%QB5.0</b> %QX5.0 ... %QX5.7	als Byte mit DO40 ... DO47 als einzelne Bits für jeden Ausgang
P0	<b>%QX0.0</b> Impulsausgang: DO0, siehe Abschnitt 6.6.2	(Ausgangswert bei inaktivem Generator)
P1	<b>%QX0.1</b> Impulsausgang: DO1, siehe Abschnitt 6.6.2	(Ausgangswert bei inaktivem Generator)
P2	<b>%QX0.2</b> Impulsausgang: DO2, siehe Abschnitt 6.6.2	(Ausgangswert bei inaktivem Generator)
P3	<b>%QX0.3</b> Impulsausgang: DO3, siehe Abschnitt 6.6.2	(Ausgangswert bei inaktivem Generator)

- (1) Die entsprechend gekennzeichneten Komponenten sind nur im Prozessabbild verfügbar, wenn die Option **"Enable extended I/O's"** in der SPS-Konfiguration aktiviert ist (siehe Abschnitt 7.4.1). Alternativ kann auch der Eintrag **"EnableExtIo="** in der Sektion **"[Proclmg]"** innerhalb der Konfigurationsdatei **"/home/plc/plccore-9g20.cfg"** direkt gesetzt werden (siehe Abschnitt 7.4.3). Die entsprechende Konfigurationseinstellung wird beim Starten der SPS-Firmware ausgewertet.

Die Ein- und Ausgänge des PLCcore-9G20 werden nicht negiert im Prozessabbild verwaltet, d.h. ein H-Pegel an einem Eingang führt zum Wert "1" an der entsprechenden Adresse im Prozessabbild und umgekehrt führt der Wert "1" im Prozessabbild zu einem H-Pegel am zugehörigen Ausgang.

## 6.4.2 Ein- und Ausgänge anwenderspezifischer Baseboards

Die nach außen geführten Anschlussleitungen des FPGA bietet dem Anwender größtmögliche Freiheitsgrade bei der Gestaltung der Ein-/Ausgangsbeschaltung des PLCcore-9G20. Damit können sämtliche Ein- und Ausgänge des PLCcore-9G20 flexibel an die jeweiligen Erfordernisse angepasst werden. Dies wiederum hat zur Folge, dass das Prozessabbild des PLCcore-9G20 maßgeblich von der konkreten Realisierung der anwenderspezifischen Außenbeschaltung definiert wird. Die softwareseitige Einbindung der Ein-/Ausgangskomponenten in das Prozessabbild erfordert das **"Driver Development Kit für das ECUcore-9G20"** (Bestellnummer SO-1106).

## 6.5 Kommunikationsschnittstellen

### 6.5.1 Serielle Schnittstellen

Das PLCcore-9G20 besitzt 5 serielle Schnittstellen (COM0 ... COM4), die als RS-232 ausgeführt sind. Details zur Hardwareanschaltung beschreibt das *"Hardware Manual Development Board ECUcore-9G20"* (Manual-Nr.: L-1256).

**COM0:** Die Schnittstelle COM0 dient primär als Serviceschnittstelle zur Administration des PLCcore-9G20. Sie wird standardmäßig im Boot-Skript *"etc/inittab"* dem Linux-Prozess *"getty"* zugeordnet und als Linux-Konsole zur Administration des PLCcore-9G20 benutzt. Die Schnittstelle COM0 ist zwar prinzipiell aus einem SPS-Programm über die Funktionsbausteine vom Typ *"SIO\_Xxx"* nutzbar (siehe Manual *"SYS TEC spezifische Erweiterungen für OpenPCS / IEC 61131-3"*, Manual-Nr.: L-1054), allerdings sollten hier nur Zeichen ausgegeben werden. Empfangene Zeichen versucht das Modul stets als Linux-Kommandos zu interpretieren und auszuführen.

Um die Schnittstelle aus einem SPS-Programm frei verwenden zu können, ist das Boot-Skript *"etc/inittab"* entsprechend anzupassen, was nur durch eine Modifikation des Linux-Images möglich ist. Voraussetzung hierfür ist das Softwarepaket SO-1105 (*"VMware-Image des Linux-Entwicklungssystems für das ECUcore-9G20"*).

**COM1..4:** Die Schnittstellen COM1 ... COM4 sind frei verfügbar und dienen in der Regel zum Datenaustausch zwischen PLCcore-9G20 und anderen Feldgeräten unter Kontrolle des SPS-Programms.

Die Schnittstellen COM1 ... COM4 sind aus einem SPS-Programm über die Funktionsbausteine vom Typ *"SIO\_Xxx"* nutzbar (siehe Manual *"SYS TEC spezifische Erweiterungen für OpenPCS / IEC 61131-3"*, Manual-Nr.: L-1054).

### 6.5.2 CAN-Schnittstellen

Das PLCcore9G20 besitzt 1 CAN-Schnittstelle (CAN0). Details zur Hardwareanschaltung beschreibt das *"Hardware Manual Development Board ECUcore-9G20"* (Manual-Nr.: L-1256).

Die CAN-Schnittstelle ermöglicht den Datenaustausch mit anderen Geräten über Netzwerkvariablen und ist zudem aus einem SPS-Programm über die Funktionsbausteine vom Typ *"CAN\_Xxx"* nutzbar (siehe Abschnitt 6.9 sowie *"User Manual CANopen-Erweiterung für IEC 61131-3"*, Manual-Nr.: L-1008).

Detaillierte Informationen zu Nutzung der CAN-Schnittstelle in Verbindung mit CANopen beschreibt Abschnitt 6.9.

### 6.5.3 Ethernet-Schnittstellen

Das PLCcore-9G20 besitzt 1 Ethernet-Schnittstelle (ETH0). Details zur Hardwareanschaltung beschreibt das *"Hardware Manual Development Board ECUcore-9G20"* (Manual-Nr.: L-1256).

Die Ethernet-Schnittstelle dient sowohl als Serviceschnittstelle zur Administration des PLCcore-9G20 als auch zum Datenaustausch mit beliebigen anderen Geräten. Aus einem SPS-Programm ist die Schnittstelle über Funktionsbausteine vom Typ *"LAN\_Xxx"* nutzbar (siehe Manual *"SYS TEC spezifische Erweiterungen für OpenPCS / IEC 61131-3"*, Manual-Nr.: L-1054).

Das SPS-Programmbeispiel *"UdpRemoteCtrl"* verdeutlicht die Nutzung der Funktionsbausteine vom Typ *"LAN\_Xxx"* innerhalb eines SPS-Programms.

## 6.6 Spezifische Peripherieschnittstellen

### 6.6.1 Zählereingänge

Das PLCcore-9G20 besitzt 4 schnelle Zählereingänge (C0 ... C3). Die Zählereingänge müssen vor ihrer Nutzung zunächst mit Hilfe des Funktionsbaustein "CNT\_FUD" parametrieren werden (siehe Manual "SYS TEC spezifische Erweiterungen für OpenPCS / IEC 61131 3", Manual-Nr.: L 1054). Der aktuelle Zählerwert ist dann in einem SPS-Programm sowohl über das Prozessabbild zugänglich (siehe Tabelle 8 im Abschnitt 6.4.1), als auch über den Funktionsbaustein "CNT\_FUD". Tabelle 9 listet die Zuordnung zwischen Zählerkanälen und Eingängen auf.

Tabelle 9: Zuordnung zwischen Zählerkanälen und Eingängen

Zählerkanal	Zähleingang	Optionalere Richtungseingang	Zählerwert im Prozessabbild
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

Um die für den FPGA erforderliche Mindestflankensteilheit zu gewährleisten, sind die Zählereingänge gemäß Bild 35 in Anhang B zu beschalten. Eine zu geringe Flankensteilheit kann falsche Zählerwerte verursachen.

### 6.6.2 Pulsausgänge

Das PLCcore-9G20 besitzt 4 Pulsausgänge (P0 ... P3) zur Ausgabe von PWM- und PTO-Signalfolgen. Die Pulsausgänge müssen vor ihrer Nutzung zunächst mit Hilfe des Funktionsbaustein "PTO\_PWM" parametrieren werden (siehe Manual "SYS TEC spezifische Erweiterungen für OpenPCS / IEC 61131 3", Manual-Nr.: L 1054). Nach dem Starten des Impulsgenerators übernimmt dieser die Steuerung des zugehörigen Ausgangs. Nach Deaktivierung des Impulsgenerators nimmt der zugehörige Ausgang den im Prozessabbild für den betreffenden Ausgang abgelegten Wert an (siehe Tabelle 8 im Abschnitt 6.4.1). Tabelle 10 listet die Zuordnung zwischen Impulskanälen und Ausgängen auf.

Tabelle 10: Zuordnung zwischen Impulskanälen und Ausgängen

Impulskanal	Impulsausgang
P0	P0 (DO0) %QX0.0
P1	P1 (DO1) %QX0.1
P2	P0 (DO2) %QX0.2
P3	P1 (DO3) %QX0.3

## 6.7 Bedien- und Anzeigeelemente

### 6.7.1 Run/Stop-Schalter

Die Modulanschlüsse "IO79", "IO80" und "IO81" (siehe Tabelle 5 und Referenzdesign im Anhang B) sind für die Anbindung eines Run/Stop-Schalters vorgesehen. Mit Hilfe dieses Run/Stop-Schalters ist es möglich, die Abarbeitung eines SPS-Programms zu starten und zu unterbrechen. Der Run/Stop-Schalter bildet mit den Start- und Stop-Schaltflächen der *OpenPCS*-Programmierungsumgebung eine "logische" UND-Verknüpfung. Das bedeutet, dass die SPS erst dann mit der Programm-Abarbeitung beginnt, wenn sich der lokale Run/Stop-Schalter in Stellung "Run" befindet **UND** außerdem ein Startkommando (Kalt-, Warm- oder Heißstart) durch die *OpenPCS*-Oberfläche erteilt wurde. Die Reihenfolge ist dabei irrelevant. Ein von *OpenPCS* veranlasster Run-Befehl bei gleichzeitiger Stellung des Run/Stop-Schalters in Stellung "Stop" ist an dem kurzen Aufblinker der Run-LED (grün) erkennbar.

In der Position "MRes" ("Modul Reset") ermöglicht der Run/Stop-Schalter das lokale Löschen eines auf dem PLCcore-9G20 befindlichen SPS-Programms. Dies ist beispielsweise dann notwendig, wenn sich das SPS-Programm durch einen Programmierfehler in einer Endlosschleife "verklemmt" und ein Zugriff von der *OpenPCS*-Programmierungsumgebung dadurch nicht mehr möglich ist. Die zum Löschen des SPS-Programms notwendige Vorgehensweise beschreibt Abschnitt 6.8.

### 6.7.2 Run-LED (grün)

Der Modulanschluss "/Run-LED" (siehe Tabelle 5 und Referenzdesign im Anhang B) ist für die Anbindung einer Run-LED vorgesehen. Diese Run-LED gibt Auskunft über den Aktivitätszustand der Steuerung, die durch verschiedene Modi dargestellt werden:

Tabelle 11: Anzeigezustände Run-LED

LED-Modus	SPS-Aktivitätszustand
Aus	Die SPS befindet sich im Zustand "Stop": <ul style="list-style-type: none"> <li>die SPS besitzt kein gültiges Programm,</li> <li>die SPS hat ein Stop-Kommando von der <i>OpenPCS</i>-Programmierungsumgebung erhalten oder</li> <li>die Programm-Abarbeitung wurde aufgrund eines internen Fehlers abgebrochen</li> </ul>
Kurzes Aufblinker im Puls-Verhältnis 1:8	Die SPS befindet sich in Bereitschaft, führt jedoch noch keine Verarbeitung aus: <ul style="list-style-type: none"> <li>Die SPS hat ein Start-Kommando von der <i>OpenPCS</i>-Programmierungsumgebung erhalten, der lokale Run/Stop-Schalter befindet sich jedoch noch in Stellung "Stop"</li> </ul>
Langsames Blinken im Puls-Verhältnis 1:1	Die SPS befindet sich im Zustand "Run" und arbeitet das SPS-Programm ab
Schnelles Blinken im Puls-Verhältnis 1:1	Die SPS befindet sich im Modus "Rücksetzen", siehe Abschnitt 6.8

### 6.7.3 Error-LED (rot)

Der Modulanschluss *"/Error-LED"* (siehe Tabelle 5 und Referenzdesign im Anhang B) ist für die Anbindung einer Error-LED vorgesehen. Diese Error-LED gibt Auskunft über den Fehlerzustand der Steuerung, die durch verschiedene Modi dargestellt werden:

Tabelle 12: Anzeigezustände Error-LED

LED-Modus	SPS-Fehlerzustand
Aus	Es ist kein Fehler aufgetreten, die SPS befindet sich im Normalzustand
Dauerlicht	Es ist ein schwerwiegender Fehler aufgetreten: <ul style="list-style-type: none"> <li>Die SPS wurde mit einer ungültigen Konfiguration gestartet (z.B. CAN-Knotenadresse 0x00) und musste beendet werden oder</li> <li>Bei der Abarbeitung eines Programms trat ein schwerwiegender Fehler auf, der die SPS veranlasste, den Zustand <i>"Run"</i> selbständig zu beenden (Division durch Null, ungültiger Array-Zugriff, ...), siehe unten</li> </ul>
Langsames Blinken im Puls-Verhältnis 1:1	Bei der Kommunikation mit dem Programmiersystem trat ein Netzwerkfehler auf, ein evtl. gestartetes Programm wird jedoch weiter abgearbeitet. Dieser Fehlerzustand wird von der SPS bei der nächsten erfolgreichen Kommunikation mit dem Programmiersystem selbständig zurückgesetzt.
Schnelles Blinken im Puls-Verhältnis 1:1	Die SPS befindet sich im Modus <i>"Rücksetzen"</i> , siehe Abschnitt 6.8
Kurzes Aufblinken im Puls-Verhältnis 1:8	Die SPS befindet sich in Bereitschaft, führt jedoch noch keine Verarbeitung aus: <ul style="list-style-type: none"> <li>Die SPS hat ein Start-Kommando von der <i>OpenPCS</i>-Programmierungsumgebung erhalten, der lokale Run/Stop-Schalter befindet sich jedoch noch in Stellung <i>"Stop"</i></li> </ul>

Beim Auftreten eines schwerwiegenden Systemfehlers wie z.B. Division durch Null oder einem ungültiger Array-Zugriff geht die Steuerung selbständig vom Zustand *"Run"* in den Zustand *"Stop"* über. Erkennbar ist dies am Dauerlicht der Error-LED (rot). In einem solchen Fall wird die Fehlerursache jedoch von der SPS gespeichert und beim nächsten Anmelden des Programmiersystems zum PC übermittelt und dort angezeigt.

## 6.8 Lokales Löschen des SPS-Programms

Mit Hilfe der Schalterstellung *"MRes"* (*"Modul Reset"*) des Run/Stop-Schalters (siehe Abschnitt 6.7.1) kann ein auf dem PLCcore-9G20 befindliches Programm gelöscht werden. Dies ist beispielsweise dann notwendig, wenn sich das SPS-Programm durch einen Programmierfehler in einer Endlosschleife *"verklemmt"* und ein Zugriff von der *OpenPCS*-Programmierungsumgebung dadurch nicht mehr möglich ist. Um ein versehentliches Löschen des SPS-Programms zu vermeiden, ist folgende Bedienreihenfolge einzuhalten:

- (1) Run/Stop-Schalter in Position "MRes" stellen
- (2) Reset am PLCcore-9G20 auslösen (durch Reset-Taster des Developmentboards oder kurzzeitige Spannungsunterbrechung)  
⇒ die Run-LED (grün) blinkt schnell mit einem Puls-Verhältnis von 1:1
- (3) Run/Stop-Schalter in Position "Run" stellen  
⇒ die Error-LED (rot) blinkt schnell mit einem Puls-Verhältnis von 1:1
- (4) Run/Stop-Schalter **innerhalb von 2 Sekunden** wieder zurück in Stellung "MRes" bringen  
⇒ das PLCcore-9G20 löscht sein SPS-Programm  
⇒ Run-LED (grün) und Error-LED (rot) blinken wechselseitig
- (5) Run/Stop-Schalter wieder zurück in Position "Stop" oder "Run" bringen und einen erneuten Reset auslösen, um das PLCcore-9G20 im normalen Arbeitsmodus zu starten

Wird am PLCcore-9G20 ein Reset ausgelöst (z.B. durch kurzzeitige Spannungsunterbrechung), während sich der Run/Stop-Schalter in Position "MRes" befindet, erkennt das Modul eine Rücksetzanforderung. Signalisiert wird dies durch ein schnelles Blinken der Run-LED (grün). Dieser Modus kann jedoch wieder gefahrlos beenden werden. Dazu ist der Run/Stop-Schalter in Stellung "Run" oder "Stop" zu bringen (jetzt blinkt die Error-LED) und mindestens 2 Sekunden warten. Das PLCcore-9G20 bricht nach dieser Zeit den Rücksetzvorgang selbständig ab und startet mit dem zuletzt gespeicherten SPS-Programm im normalen Betriebsmodus.

## 6.9 Nutzung der CAN-Schnittstellen mit CANopen

Das PLCcore-9G20 besitzt 1 CAN-Schnittstelle (CAN0), die als CANopen-Manager nutzbar ist (konform zum CiA Draft Standard 302). Die Konfiguration der Schnittstelle (aktiv/inaktiv, Knotennummer, Bitrate, Master an/aus) beschreibt Abschnitt 7.4.

Die CAN-Schnittstelle ermöglicht den Datenaustausch mit anderen Geräten über Netzwerkvariablen und ist zudem aus einem SPS-Programm über die Funktionsbausteine vom Typ "CAN\_Xxx" nutzbar. Ausführliche Details hierzu beschreibt das "*User Manual CANopen-Erweiterung für IEC 61131-3*", Manual-Nr.: L-1008.

CANopen stellt mit den beiden Diensten **PDO** (**P**rocess **D**ata **O**bjects) und **SDO** (**S**ervice **D**ata **O**bjects) zwei unterschiedliche Mechanismen für den Datenaustausch zwischen den einzelnen Feldbusgeräten zur Verfügung. Die von einem Knoten gesendeten Prozessdaten (**PDO**) stehen als Broadcast gleichzeitig allen interessierten Empfängern zur Verfügung. Durch die Realisierung als quittungslose Broadcast-Nachrichten sind PDOs auf 1 CAN-Telegramm und damit auf maximal 8 Byte Nutzdaten limitiert. Im Gegensatz dazu basieren **SDO**-Transfers auf logischen Punkt-zu-Punkt-Verbindungen ("Peer to Peer") zwischen zwei Knoten und ermöglichen den quitierten Austausch von Datenpaketen, die auch größer als 8 Bytes sein können. Diese Datenpakete werden intern durch eine entsprechende Anzahl quittierter CAN-Telegramme übertragen. Beide Dienste sind sowohl für die Schnittstelle CAN0 als auch CAN1 des PLCcore-9G20 nutzbar.

Die SDO-Kommunikation erfolgt grundsätzlich immer über Funktionsbausteine vom Typ "CAN\_SDO\_Xxx" erfolgt (siehe "*User Manual CANopen-Erweiterung für IEC 61131-3*", Manual-Nr.: L-1008). Für PDOs stehen ebenfalls Funktionsbausteine bereit ("CAN\_PDO\_Xxx"), diese sollten jedoch nur in besonderen Fällen verwendet werden, um auch nicht CANopen-konforme Geräte ansprechen zu können. Für die Anwendung der PDO-Bausteine muss die CANopen-Konfiguration im Detail bekannt sein, da die Bausteine lediglich 8 Bytes als Übergabeparameter benutzen, die Zuordnung der Bytes zu den Prozessdaten jedoch Aufgabe des Anwenders ist.

Statt PDO-Bausteinen sollten vorrangig Netzwerkvariablen für den PDO-basierten Datenaustausch verwendet werden. Netzwerkvariablen stellen die einfachste Form des Datenaustausches mit anderen CANopen-Knoten dar. In einem SPS-Programm erfolgt der Zugriff auf Netzwerkvariablen in derselben

Form wie auf interne, lokale Variablen der SPS. Aus Sicht des SPS-Programmierers ist es somit völlig unbedeutend, ob z.B. eine Input-Variable einem lokalen Eingang der Steuerung zugeordnet ist oder einen Eingang eines dezentralen Erweiterungsmoduls repräsentiert. Die Verwendung von Netzwerkvariablen basiert auf der Einbindung von DCF-Dateien, die durch einen entsprechenden CANopen-Konfigurator erstellt wurden. Die DCF-Dateien beschreiben zum einen die Kommunikationsparameter eines Gerätes (CAN Identifier, usw.) und zum anderen beinhalten sie eine Zuordnung der Netzwerkvariablen auf die einzelnen Bytes eines CAN-Telegramms (Mapping). Die Anwendung von Netzwerkvariablen erfordert nur allgemeine Grundkenntnisse über CANopen.

Der Austausch von PDOs erfolgt in einem CANopen-Netzwerk nur im Zustand "OPERATIONAL". Befindet sich das PLCcore-9G20 nicht in diesem Zustand, verarbeitet es keine PDOs (weder sende- noch empfangsseitig) und aktualisiert folglich auch nicht den Inhalt der Netzwerkvariablen. Das Setzen der Betriebszustände "OPERATIONAL", "PRE-OPERATIONAL" usw. ist Aufgabe des CANopen-Managers (meist auch als "CANopen-Master" bezeichnet). In typischen CANopen-Netzwerken wird für den CANopen-Manager ein programmierbarer Knoten - meist in Form einer SPS - verwendet. Das PLCcore-9G20 kann optional die Aufgabe des CANopen-Managers übernehmen. Die Aktivierung des Managers beschreibt Abschnitt 7.4.

Als CANopen-Manager ist das PLCcore-9G20 zudem in der Lage, die am CAN-Bus angeschlossenen CANopen I/O-Geräte ("CANopen-Slaves") zu parametrieren, indem es die vom CANopen-Konfigurator erstellten DCF-Dateien beim Systemstart per SDO auf die jeweiligen Knoten überträgt.

### 6.9.1 CAN-Schnittstelle CAN0

Die Schnittstelle CAN0 besitzt ein dynamisches Object-Dictionary. Das bedeutet, dass diese Schnittstelle nach dem Einschalten der SPS zunächst überhaupt keine Kommunikationsobjekte für den Datenaustausch mit anderen Geräten zur Verfügung stellt. Erst nach dem Download eines SPS-Programms (bzw. dessen Rückladen aus dem nichtflüchtigen Speicher nach Power-on) werden die benötigten Kommunikationsobjekte anhand der in das SPS-Projekt eingebundenen DCF-Datei dynamisch angelegt. Dadurch ist die CAN-Schnittstelle CAN0 extrem flexibel auch für große Datenmengen nutzbar.

Auf Ebene des SPS-Programms werden die Netzwerkvariablen entsprechend der Norm IEC61131-3 als "VAR\_EXTERNAL" deklariert und somit als "außerhalb der Steuerung" gekennzeichnet, z.B.:

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

Die detaillierte Vorgehensweise zum Einbinden von DCF-Dateien in das SPS-Projekt und zur Deklaration von Netzwerkvariablen beschreibt das "User Manual CANopen-Erweiterung für IEC 61131-3" (Manual-Nr.: L-1008).

Bei der Nutzung der CAN-Schnittstelle CAN0 ist zu beachten, dass aufgrund des dynamischen Objekt-Verzeichnisses das Anlegen der benötigten Objekte nach jedem Systemstart erneut erfolgt. Die "Aufbauvorschrift" dazu beinhaltet die in das SPS-Projekt eingebundene DCF-Datei. **Änderungen an der Konfiguration können daher nur durch entsprechende Modifikationen der DCF-Datei erfolgen.** Das bedeutet, dass nach einer Änderung der Netzwerkkonfiguration (Modifikation der DCF-Datei) das SPS-Projekt neu übersetzt und auf das PLCcore-9G20 geladen werden muss.

### 6.9.2 Zusätzliche CAN-Schnittstellen

Die auf dem PLCcore-9G20 eingesetzte SPS-Firmware ist prinzipiell in der Lage, mehrere CAN-Schnittstellen simultan zu bedienen (analog zu anderen SPS-Typen wie beispielsweise PLCcore-5484 oder PLCmodule-C32).

Bei Bedarf können weitere CAN-Schnittstellen extern an das Modul angeschlossen werden. Bitte setzen Sie sich bei Interesse dazu mit unserem Support in Verbindung:

[support@sys-tec-electronic.com](mailto:support@sys-tec-electronic.com)

## 7 Konfiguration und Administration des PLCcore-9G20

### 7.1 Systemvoraussetzungen und erforderliche Softwaretools

Zur Administration des PLCcore-9G20 ist ein beliebiger Windows- oder Linux-PC erforderlich, der über eine Ethernet-Schnittstelle sowie eine serielle Schnittstelle (RS232) verfügt. Als Alternative zur seriellen on-board Schnittstelle eignet sich auch das von SYS TEC angebotenen USB-RS232 Adapter Kabel (Bestellnummer 3234000, siehe Abschnitt 4.4.1), das eine entsprechende RS232-Schnittstelle über einen USB-Port zur Verfügung stellt.

Alle in diesem Manual aufgeführten Beispiele beziehen sich auf die Administration des PLCcore-9G20 von einem Windows-PC aus. Das Vorgehen auf einem Linux-PC ist analog.

Zur Administration des PLCcore-9G20 sind folgende Softwaretools erforderlich:

**Terminalprogramm** Ein Terminalprogramm ermöglicht die Kommunikation mit der **Kommando-Shell** des PLCcore-9G20 über eine **serielle RS232-Verbindung an COM0 des PLCcore-9G20**. Diese ist Voraussetzung für die im Abschnitt 7.3 beschriebene Ethernet-Konfiguration des PLCcore-9G20. Nach Abschluss der Ethernet-Konfiguration können alle weiteren Kommandos wahlweise entweder auch weiterhin im Terminalprogramm eingegeben werden oder alternativ dazu in einem Telnet-Client (siehe unten).

Als Terminalprogramm eignen sich z.B. das im Lieferumfang von Windows bereits enthaltenen "*HyperTerminal*" oder für gehobeneren Ansprüche das als Open-Source verfügbare "*TeraTerm*" (Download unter: <http://tssh2.sourceforge.jp>).

**Telnet-Client** Ein Telnet-Client ermöglicht die Kommunikation mit der **Kommando-Shell** des PLCcore-9G20 über eine **Ethernet-Verbindung an ETH0 des PLCcore-9G20**. Voraussetzung für die Verwendung eines Telnet-Clients ist eine abgeschlossene Ethernet-Konfiguration des PLCcore-9G20 gemäß Abschnitt 7.3. Alternativ zum Telnet-Client können auch sämtliche Kommandos über ein Terminalprogramm (an COM0 des PLCcore-9G20) eingegeben werden.

Als Telnet-Client eignet sich z.B. das im Lieferumfang von Windows bereits enthaltene "*Telnet*" oder ebenfalls "*TeraTerm*", das gleichzeitig auch als Terminalprogramm eingesetzt werden kann (siehe oben).

**FTP-Client** Ein FTP-Client ermöglicht den Austausch von Dateien zwischen dem PLCcore-9G20 (ETH0) und dem PC. Dies erlaubt beispielsweise das **Editieren von Konfigurationsdateien**, indem diese zunächst vom PLCcore-9G20 auf den PC übertragen, dort mit einem Editor bearbeitet und anschließend wieder zurück auf das PLCcore-9G20 geschrieben werden. Der Download von Dateien auf das PLCcore-9G20 ist aber auch zum **Update der SPS-Firmware** erforderlich. (Hinweis: Der Update der *SPS-Firmware* ist nicht identisch mit dem Update des *SPS-Anwenderprogramms*. Das SPS-Programm wird direkt aus der OpenPCS-Programmierungsumgebung heraus auf das Modul übertragen, hierzu ist keinerlei Zusatzsoftware erforderlich.)

Als FTP-Client für den PC eignen sich beispielsweise das als Open-Source verfügbare "*WinSCP*" (Download unter: <http://winscp.net>), das lediglich aus einer einzelnen EXE-Datei besteht, die keine Installation erfordert und sofort

gestartet werden kann. Ebenso geeignet sind aber auch die Freeware "Core FTP LE" (Download unter: <http://www.coreftp.com>) oder der bereits im Dateimanager "Total Commander" integrierte FTP-Client.

#### **TFTP-Server**

Der TFTP-Server ist lediglich zum Update des Linux-Images auf dem PLCcore-9G20 erforderlich. Als TFTP-Server eignet sich die Freeware "TFTPD32" (Download unter: <http://tftpd32.jounin.net>). Das Programm besteht lediglich aus einer einzelnen EXE-Datei, die keine Installation erfordert und sofort gestartet werden kann.

Bei Programmen die über die Ethernet-Schnittstelle kommunizieren, wie beispielsweise FTP-Client oder TFTP-Server, ist darauf zu achten, dass die entsprechenden Rechte in der Windows-Firewall freigegeben sind. In der Regel melden Firewalls, dass ein Programm Zugriff auf das Netzwerk erlangen möchte und fragen, ob dieser Zugriff erlaubt oder abgeblockt werden soll. Hier ist in jedem Fall der entsprechende Zugriff zu gestatten.

## **7.2 Linux-Autostart aktivieren bzw. deaktivieren**

Im Standardbetriebsmodus startet der Bootloader "U-Boot" bei Reset (bzw. Power-on) autark das Linux-Betriebssystem des Moduls, das dann wiederum das Laden aller weiteren Softwarekomponenten bis hin zur Ausführung des SPS-Programms übernimmt (siehe Abschnitt 6.2). Für Servicezwecke wie beispielsweise die Konfiguration der Ethernet-Schnittstelle (siehe Abschnitt 7.3) oder zum Update des Linux-Images (siehe Abschnitt 7.13.2) ist es erforderlich, diesen Autostart-Mechanismus zu unterbinden und stattdessen zum "U-Boot" Kommandoprompt zu wechseln (Konfigurationsmodus).

Der automatische Start des Linux-Betriebssystems ist an die **gleichzeitige Erfüllung** verschiedener Bedingungen geknüpft ("UND-Verknüpfung"). Dementsprechend ist es zur Unterdrückung autarken Linux-Starts ausreichend, eine dieser Bedingungen **nicht zu erfüllen**.

Im Detail werden durch den Bootloader "U-Boot" die in Tabelle 13 aufgelisteten Voraussetzungen geprüft, von denen alle Bedingungen für ein autarkes Booten des Linux-Images erfüllt sein müssen.

Tabelle 13: Voraussetzungen zum Booten von Linux

Nr.	Bedingung	Bemerkung
1	DIP1 auf PLCcore-9G20 = "Off"  UND  Anschluss "/BOOT" = High (Taster S406 am Developmentboard nicht gedrückt)	Der DIP-Schalter 1 auf dem PLCcore-9G20 und der Modulanschluss "/BOOT" sind elektrisch parallel geschaltet. Nur wenn beide Elemente nicht aktiv sind (DIP-Schalter 1 offen, Modulanschluss "/BOOT" nicht aktiv) liegt das Signal "/BOOT" am PLCcore-9G20 auf H-Pegel und gibt damit den Linux-Autostart frei. Die Lage des DIP-Schalter 1 auf dem PLCcore-9G20 zeigt Bild 7, die Position des Anschlusses "/BOOT" am Steckverbinder des Moduls ist im Hardware Manual ECUCore-9G20 (Manual-Nr.: L-1255) definiert.
2	Kein Abbruch der Autoboot-Prozedur über COM0 des PLCcore-9G20	Sind die vorangegangenen Bedingungen erfüllt, überprüft "U-Boot" nach Reset für ca. 1 Sekunde die serielle Schnittstelle COM0 des PLCcore-9G20 auf den Empfang eines SPACE-Zeichens (ASCII 20H). Wird innerhalb dieser Zeit ein entsprechendes Zeichen empfangen, unterbindet "U-Boot" den Linux-Bootvorgang und aktiviert stattdessen seinen eigenen Kommandoprompt.

**Gemäß Tabelle 13 wird unter folgenden Voraussetzungen der Linux-Bootvorgang nach Reset (z.B. Taster S405 am Developmentboard) unterbunden und stattdessen der "U-Boot"-Kommandoprompt aktiviert:**

- (1) **DIP1 = "On" oder /BOOT = "Low"**      DIP1: siehe Bild 7, /BOOT: siehe Manual L-1255  
Developmentboard: "/BOOT" = Taster S406  
DIP1 und "/BOOT" sind parallel geschaltet
- ODER -
- (2) **Empfang eines SPACE-Zeichens (ASCII 20H) innerhalb 1 Sekunde nach Reset**

Nach dem Betätigen des Reset-Tasters (z.B. Taster S405 am Developmentboard) meldet sich der "U-Boot" Kommandoprompt.

Bild 7 zeigt die Lage und Bedeutung des DIP-Schalters 1 auf dem PLCcore-9G20. Da dieser DIP-Schalter im eingebauten Zustand des Moduls unter Umständen schwer zugänglich sein kann, ist das damit verbundene Portpin des Prozessors parallel dazu auch als Anschluss "/BOOT" am Steckverbinder des PLCcore-9G20 verfügbar (siehe Tabelle 5).

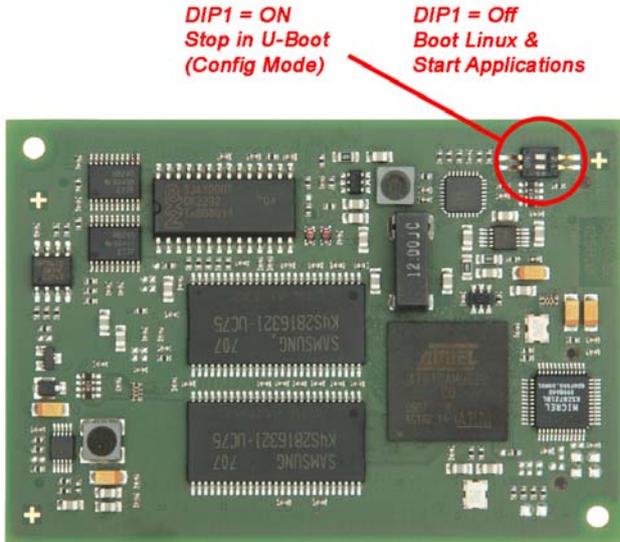


Bild 7: Lage und Bedeutung des DIP-Schalter 1 auf dem PLCcore-9G20

Die Kommunikation mit dem Bootloader "U-Boot" erfolgt ausschließlich über die serielle Schnittstelle COM0 des PLCcore-9G20. Als Gegenstelle ist auf dem PC ein beliebiges Terminalprogramm zu starten (z.B. HyperTerminal oder TeraTerm, siehe Abschnitt 7.1) und wie folgt zu konfigurieren (siehe Bild 8):

- 115200 Baud
- 8 Datenbits
- 1 Stopbit
- keine Parität
- keine Flusskontrolle

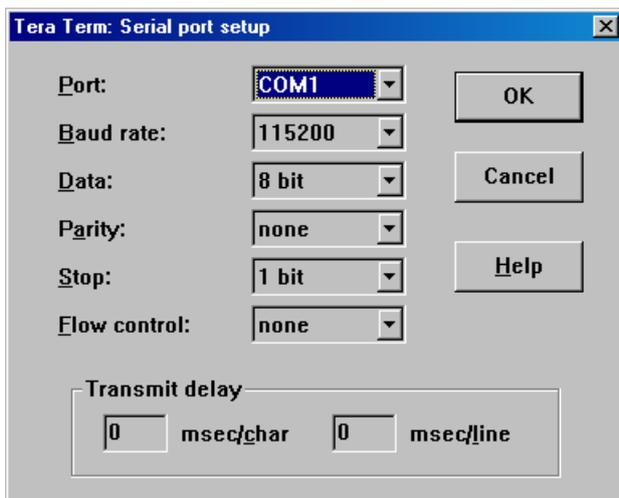


Bild 8: Terminaleinstellungen am Beispiel von "TeraTerm"

### 7.3 Ethernet-Konfiguration des PLCcore-9G20

Die zentrale Ethernet-Konfiguration des PLCcore-9G20 erfolgt im Bootloader "U-Boot" und wird von hier für alle anderen Softwarekomponenten übernommen (Linux, SPS-Firmware, HTTP-Server usw.) Die Ethernet-Konfiguration erfolgt über die serielle Schnittstelle COM0. **Dazu ist wie im Abschnitt 7.2 beschrieben der "U-Boot"-Kommandoprompt zu aktivieren.** Tabelle 14 listet die zur Ethernet-Konfiguration des PLCcore-9G20 notwendigen U-Boot-Kommandos auf.

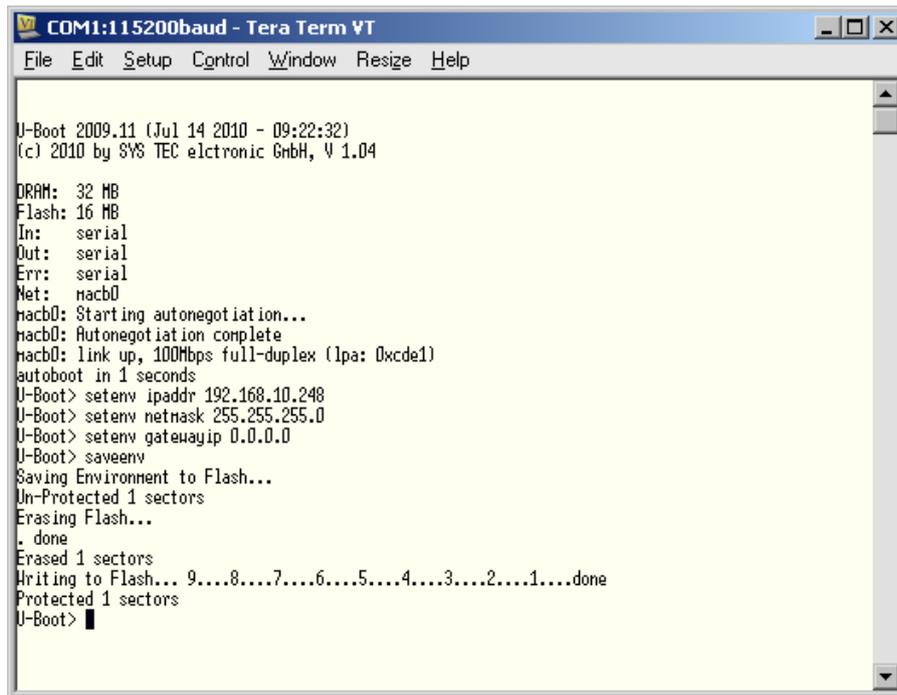
Tabelle 14: "U-Boot" Kommandos zur Konfiguration des PLCcore-9G20

Einstellung	Kommando	Bemerkung
MAC-Adresse	setenv ethaddr <xx:xx:xx:xx:xx:xx>	Die MAC-Adresse ist eine weltweit eindeutige Kennung des Moduls, die bereits vom Hersteller vergeben wird. Diese sollte vom Anwender normalerweise nicht verändert werden.
IP-Adresse	setenv ipaddr <xxx.xxx.xxx.xxx>	Dieses Kommando setzt die lokale IP-Adresse des PLCcore-9G20. Die IP-Adresse ist vom Netzwerkadministrator festzulegen.
Netzwerkmaske	setenv netmask <xxx.xxx.xxx.xxx>	Dieses Kommando setzt die Netzwerkmaske des PLCcore-9G20. Die Netzwerkmaske ist vom Netzwerkadministrator festzulegen.
Gateway-Adresse	setenv gatewayip <xxx.xxx.xxx.xxx>	Dieses Kommando definiert die IP-Adresse des vom PLCcore-9G20 zu benutzenden Gateways. Die Gateway-Adresse ist vom Netzwerkadministrator festzulegen.  Hinweis: Befinden sich PLCcore-9G20 und Programmier-PC im selben Sub-Netz, dann kann die Definition der Gateway-Adresse entfallen und stattdessen der Wert "0.0.0.0" verwendet werden.
Speichern der Konfiguration	saveenv	Dieses Kommando speichert die aktuellen Einstellungen im Flash des PLCcore-9G20.

Die modifizierten Einstellungen können durch die Eingabe von *"printenv"* am "U-Boot" Kommandoprompt nochmals überprüft werden. Die aktuellen Einstellungen werden durch das Kommando

#### **saveenv**

persistent im Flash des PLCcore-9G20 gespeichert. Die Änderungen werden mit dem nächsten Reset des PLCcore-9G20 übernommen.



```
COM1:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help

U-Boot 2009.11 (Jul 14 2010 - 09:22:32)
(c) 2010 by SYS TEC electronic GmbH, V 1.04

DRAM: 32 MB
Flash: 16 MB
In: serial
Out: serial
Err: serial
Net: macb0
macb0: Starting autonegotiation...
macb0: Autonegotiation complete
macb0: link up, 100Mbps full-duplex (lpa: 0xcde1)
autoboot in 1 seconds
U-Boot> setenv ipaddr 192.168.10.248
U-Boot> setenv netmask 255.255.255.0
U-Boot> setenv gatewayip 0.0.0.0
U-Boot> saveenv
Saving Environment to Flash...
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... 9....8....7....6....5....4....3....2....1....done
Protected 1 sectors
U-Boot> █
```

Bild 9: Ethernet-Konfiguration des PLCcore-9G20

**Nach Abschluss der Konfiguration sind gemäß Abschnitt 7.2 die Voraussetzungen für einen Linux-Autostart wieder herzustellen.**

Nach Reset (z.B. Taster S405 am Developmentboard) startet das Modul mit den aktuellen Einstellungen.

**Hinweis:** Nach Abschluss der Konfiguration ist die serielle Verbindung zwischen PC und PLCcore-9G20 nicht mehr erforderlich.

## 7.4 SPS-Konfiguration des PLCcore-9G20

### 7.4.1 SPS-Konfiguration über WEB-Frontend

Nach dem Abschluss der Ethernet-Konfiguration (siehe Abschnitt 7.3) können alle weiteren Einstellungen über das integrierte WEB-Frontend des PLCcore-9G20 erfolgen. Beim Einsatz des PLCcore-9G20 auf dem Development Kit sind grundlegende Einstellungen alternativ auch über lokale Bedienelemente möglich (siehe Abschnitt 7.4.2).

Für die Konfiguration des PLCcore-9G20 über das WEB-Frontend ist auf dem PC lediglich ein WEB-Browser erforderlich (z.B. Microsoft Internet Explorer, Mozilla Firefox usw.). Zum Aufruf der Konfigurationsseite ist in der Adressleiste des WEB-Browsers der Präfix "*http://*" gefolgt von der im Abschnitt 7.2 festgelegten IP-Adresse des PLCcore-9G20 einzugeben, z.B. "*http://192.168.10.248*". Bild 10 verdeutlicht den Aufruf der Konfigurationsseite für das PLCcore-9G20 im WEB-Browser.

In der Standardeinstellung (Werkseinstellungen) erfordert die Konfiguration des PLCcore-9G20 über das WEB-Frontend eine Benutzeranmeldung, um unbefugte Zugriffe zu unterbinden. Dazu sind Nutzernamen und Passwort in dem entsprechenden Anmeldedialog einzugeben (siehe Bild 10). Bei Auslieferung des Moduls ist folgendes Nutzerkonto vorkonfiguriert (siehe auch Abschnitt 7.7):

User: PlcAdmin  
Passwort: Plc123

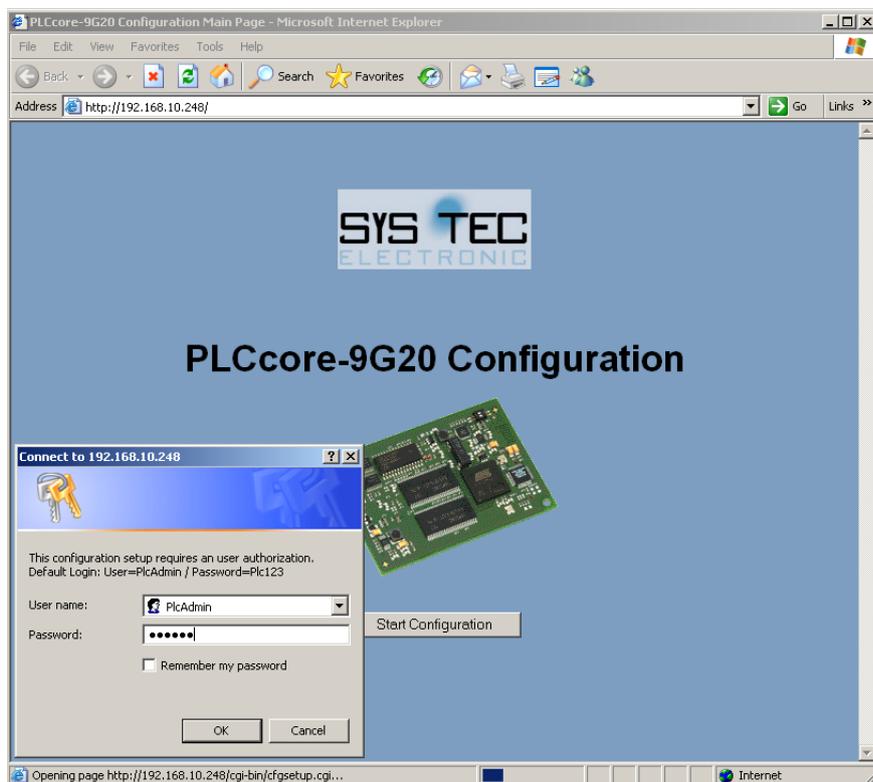


Bild 10: Anmeldedialog des WEB-Frontend

Sämtliche Konfigurationseinstellungen für das PLCcore-9G20 erfolgen dialogbasiert und werden durch Betätigen der Schaltfläche "Save Configuration" in die Datei **"/home/plc/plccore-9g20.cfg"** des PLCcore-9G20 übernommen (siehe auch Abschnitt 7.4.3). Nach Reset (z.B. Taster S405 am Developmentboard) startet das PLCcore-9G20 mit den aktuellen Einstellungen. Bild 11 zeigt die Konfiguration des PLCcore-9G20 über das WEB-Frontend.

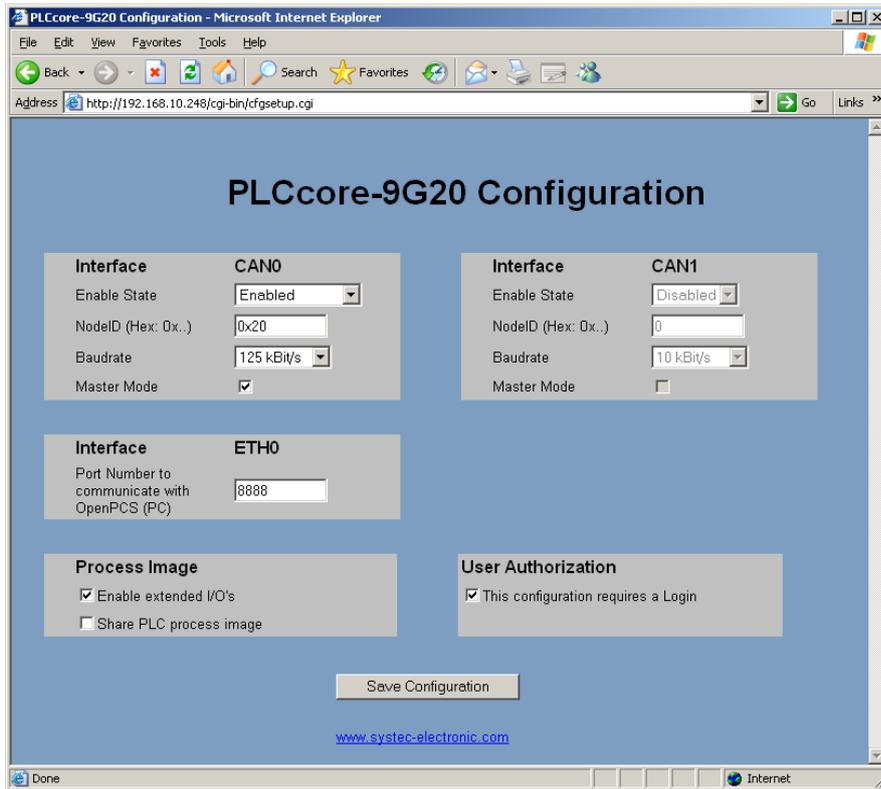


Bild 11: SPS-Konfiguration über WEB-Frontend

Bei der Auswahl von "DIP/Hex-Switch" für den Enable State des Interfaces CAN0 erfolgt die Konfiguration dieser Schnittstelle über die lokalen Bedienelemente des Development Kit PLCcore-9G20 (siehe Abschnitt 7.4.2).

In der Standardeinstellung (Werkseinstellungen) ist das PLCcore-9G20 so konfiguriert, dass für den Zugriff auf das WEB-Frontend eine Benutzeranmeldung erforderlich ist. Dabei wird nur der in der Konfigurationsdatei `"/home/plc/plccore-9g20.cfg"` angegebene Benutzername akzeptiert (Eintrag `"User="` in Sektion `"[Login]"`, siehe Abschnitt 7.4.3). Die Vorgehensweise zum Ändern des Passwortes für die Nutzeranmeldung beschreibt Abschnitt 7.10. Um einem anderen Benutzer die Konfiguration des Moduls zu ermöglichen, ist zunächst das entsprechende Nutzerkonto wie im Abschnitt 7.9 beschrieben anzulegen. Anschließend ist der neue Benutzername in der Konfigurationsdatei `"/home/plc/plccore-9g20.cfg"` einzutragen. Durch Löschen des Eintrages `"User="` in Sektion `"[Login]"` (siehe 7.4.3) wird die Limitierung auf einen Benutzer aufgehoben, so dass für die Konfiguration jeder auf dem Modul angelegte Nutzer-Account verwendet werden kann. Durch deaktivieren des Kontrollkästchens `"This configuration requires a Login"` im Feld `"User Authorization"` der Konfigurationsseite (siehe Bild 11) wird ein freier Zugriff auf die Konfiguration ohne vorherige Nutzeranmeldung ermöglicht.

## 7.4.2 SPS-Konfiguration über Bedienelemente des Development Kit PLCcore-9G20

Die SPS-Konfiguration über Bedienelemente wird durch das derzeit verfügbare Developmentboard mit der PCB-Version 4261.2 noch nicht unterstützt und bleibt einer zukünftigen Hardware-Version vorbehalten.

Die SPS-Konfiguration ist entweder über das WEB-Frontend (siehe Abschnitt 7.4.1) oder alternativ durch direktes Editieren der Konfigurationsdatei `"/home/plc/plccore-9g20.cfg"` möglich (siehe Abschnitt 7.4.3).

### 7.4.3 Aufbau der Konfigurationsdatei "plccore-9g20.cfg"

Die Konfigurationsdatei **"/home/plc/plccore-9G20.cfg"** ermöglicht eine umfassende Konfiguration des PLCcore-9G20. Ihre manuelle Bearbeitung ist jedoch nur in Ausnahmefällen sinnvoll, da die meisten der darin enthaltenen Einstellungen bequem über das WEB-Frontend editiert werden können (siehe Abschnitt 7.4.1). Der Aufbau der Konfigurationsdatei entspricht dem als "Windows INI-File" bekannten Datenformat, sie untergliedert sich in "[Sections]", die dann wiederum verschiedene "Entry=" Einträge enthalten. Tabelle 16 enthält eine Auflistung der Konfigurationseinträge. Die Einträge der Sektion "[CAN0]" haben Vorrang gegenüber den Einstellungen an den Bedienelementen (siehe Abschnitt 7.4.2).

Tabelle 16: Konfigurationseinträge der CFG-Datei

Section	Entry	Value	Bedeutung
[CAN0]	Enabled	-1, 0, 1	-1: Interface CAN0 ist aktiviert, Konfiguration erfolgt über Bedienelemente des Developmentboards (Werkseinstellung, siehe Abschnitt 7.4.2)  0: Interface CAN0 ist deaktiviert  1: Interface CAN0 ist aktiviert, Konfiguration erfolgt über nachstehende Einträge dieser Konfigurationsdatei
	NodeID	1 ... 127 bzw. 0x01 ... 0x7F	Knotennummer für Interface CAN0 (dezimal oder hexadezimal mit Präfix "0x")
	Baudrate	10, 20, 50, 125, 250, 500, 800, 1000	Bitrate für Interface CAN0
	MasterMode	0, 1	1: Master-Modus ist aktiviert 0: Master-Modus ist deaktiviert
[CAN1]	Enabled		Die Sektion "[CAN1]" wird standardmäßig nicht ausgewertet, ermöglicht aber bei Bedarf eine Erweiterung des PLCcore-9G20 um eine zusätzliche CAN-Schnittstelle (siehe Abschnitt 6.9.2)
	NodeID		
	Baudrate		
	MasterMode		
[ETH0]	PortNum	Default Portnummer: 8888	Portnummer zur Kommunikation mit dem Programmier-PC sowie zum Programmdownload (nur für PLCcore-9G20/Z5, Bestellnummer 3390025)
[Proclmg]	EnableExtIo	0, 1	0: nur on-board I/O's des PLCcore-9G20 für Prozessabbild verwenden (jedoch ohne Temperatursensor)  1: alle vom Treiber unterstützten I/O's für Prozessabbild verwenden (incl. Temperatursensor und externer ADC des Developmentboard)  (zur Anpassung des Prozessabbildes siehe Abschnitt 8.2)
	EnableSharing	0, 1	0: kein Sharing für Prozessabbild 1: Sharing für Prozessabbild ist freigegeben (siehe Abschnitt 8.1)

[Login]	Authorization	0, 1	0: Konfiguration über WEB-Frontend ist ohne Benutzeranmeldung möglich 1: Konfiguration über WEB-Frontend erfordert Benutzeranmeldung
	User	Default Name: PlcAdmin	Ist der Eintrag "User=" vorhanden, wird nur der darin definierte Benutzername bei der Anmeldung für die Konfiguration über das WEB-Frontend akzeptiert.  Ist der Eintrag nicht vorhanden, kann sich jeder auf dem PLCcore-9G20 angelegte Benutzer (siehe Abschnitt 7.9) zur Konfiguration über das WEB-Frontend anmelden.

Die Konfigurationsdatei ***"/home/plc/plccore-9g20.cfg"*** besitzt folgende Werkseinstellungen:

```
[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-Konfiguration des PLCcore-9G20

Das PLCcore-9G20 ist so konfiguriert, dass nach einem Reset die SPS-Firmware automatisch gestartet wird. Die dazu notwendigen Kommandos sind in dem Startskript ***"/home/etc/autostart"*** hinterlegt. Hier werden u.a. die notwendigen Umgebungsvariablen gesetzt sowie die erforderlichen Treiber geladen.

Bei Bedarf kann das Startskript ***"/home/etc/autostart"*** um weitere Einträge ergänzt werden. Hier kann z.B. durch Einfügen des Kommandos ***"pureftp"*** der Aufruf des FTP-Servers beim Booten des PLCcore-9G20 automatisiert werden. Das Skript lässt sich im FTP-Client ***"WinSCP"*** (siehe Abschnitt 7.1) mit Hilfe der Taste ***"F4"*** bzw. der Schaltfläche ***"F4 Edit"*** direkt auf dem PLCcore-9G20 bearbeiten.

## 7.6 Auswahl der zu startenden Firmware-Variante

Das PLCcore-9G20 wird mit verschiedenen Firmware-Varianten ausgeliefert. Diese unterscheiden sich im verwendeten Kommunikationsprotokoll für den Datenaustausch mit dem Programmier-PC und in der Verfügbarkeit der Kommunikations-FB-Klassen (siehe Abschnitt 6.3). Die Auswahl der zu verwendenden Firmware-Varianten erfolgt im Startskript *"/home/etc/autostart"*. Standardmäßig wird hierzu die im Bootloader "U-Boot" festgelegte *"BoardID"* des Moduls ausgewertet. Tabelle 17 listet die Zuordnung von Firmware-Variante und BoardID auf.

Tabelle 17: Zuordnung von BoardID und Firmware-Variante für das PLCcore-9G20

BoardID	Firmware-Variante	Bemerkung
1008004	plccore-9g20-z4	<b>PLCcore-9G20/Z4 (CANopen)</b> Kommunikation mit Programmier-PC via CANopen-Protokoll (Interface CAN0)
1008005	plccore-9g20-z5	<b>PLCcore-9G20/Z5 (Ethernet)</b> Kommunikation mit Programmier-PC via UDP-Protokoll (Interface ETH0)

Die Konfiguration der BoardID erfolgt über die serielle Schnittstelle COM0. **Dazu ist wie im Abschnitt 7.2 beschrieben der "U-Boot"-Kommandoprompt zu aktivieren.** Das Festlegen der jeweiligen BoardID erfolgt mit dem "U-Boot"-Kommando *"set boardid"* unter Angabe der in Tabelle 17 aufgeführten, zugehörigen Nummer, z.B.:

```
setenv boardid 1008005
```

Die modifizierte Einstellung kann durch die Eingabe von *"printenv"* am "U-Boot" Kommandoprompt nochmals überprüft werden. Die aktuelle Auswahl wird durch das Kommando

**saveenv**

persistent im Flash des PLCcore-9G20 gespeichert. Bild 12 veranschaulicht die Konfiguration der BoardID.

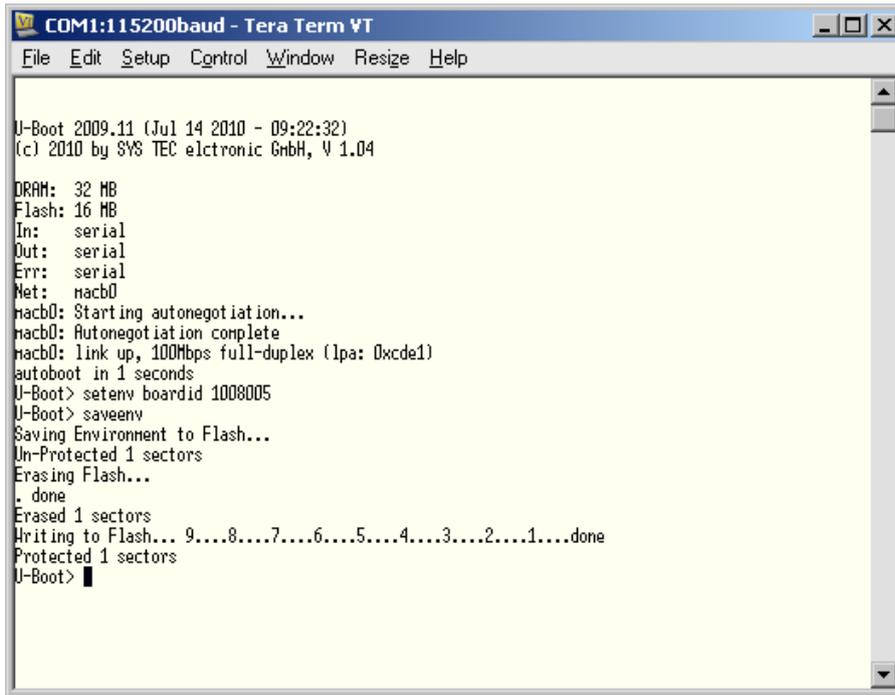


Bild 12: Auswahl der zu startenden Firmware-Variante für das PLCcore-9G20

**Nach Abschluss der Konfiguration sind gemäß Abschnitt 7.2 die Voraussetzungen für einen Linux-Autostart wieder herzustellen.**

Alternativ kann die zu startende Firmware auch direkt im Startskript ***"/home/etc/autostart"*** festgelegt werden. Dazu ist der Abschnitt "Select PLC Type" zu löschen und stattdessen die entsprechende Firmware fest vorzugeben, z.B.:

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

## 7.7 Vordefinierte Nutzerkonten

Bei der Auslieferung des PLCcore-9G20 sind die in Tabelle 18 aufgelisteten Benutzerkonten vordefiniert. Diese erlauben eine Anmeldung an der Kommando-Shell (serielle RS232-Verbindung oder Telnet) und am FTP-Server des PLCcore-9G20.

Tabelle 18: Vordefinierte Benutzerkonten des PLCcore-9G20

Benutzername	Passwort	Bemerkung
PlcAdmin	Plc123	vordefiniertes Benutzerkonto zur Administration des PLCcore-9G20 (Konfiguration, Nutzerverwaltung, Softwareupdates usw.)
root	Sys123	Haupt-Benutzerkonto ("root") des PLCcore-9G20

## 7.8 Anmeldung am PLCcore-9G20

### 7.8.1 Anmeldung an der Kommando-Shell

Die Administration des PLCcore-9G20 erfordert in einigen Fällen die Eingabe von Linux-Kommandos in der Kommando-Shell. Dazu ist eine direkte Anmeldung am Modul notwendig. Dies kann auf zwei alternativen Wegen erfolgen:

- Mit Hilfe eines **Terminalprogramms** (z.B. HyperTerminal oder TeraTerm, siehe Abschnitt 7.1) über die serielle Schnittstelle **COM0** des PLCcore-9G20, analog zum Vorgehen bei der im Abschnitt 7.2 beschriebenen Ethernet-Konfiguration. **Bei der Konfiguration der Terminaleinstellungen ist darauf zu achten, dass als Zeilenendezeichen nur "CR" (carriage return) verwendet wird.** Bei "CR+LF" (carriage return + line feed) ist keine Anmeldung mit Nutzernamen und Passwort möglich!
- Alternativ ist die Anmeldung mit Hilfe eines **Telnet-Clients** (z.B. Telnet oder ebenfalls TeraTerm) über die Ethernet-Schnittstelle **ETH0** des PLCcore-9G20 möglich.

Um sich über den in Windows standardmäßig enthaltenen Telnet-Client am PLCcore-9G20 anzumelden, ist der Befehl `"telnet"` unter Angabe der in Abschnitt 7.2 festgelegten IP-Adresse für das PLCcore-9G20 aufzurufen, z.B.

```
telnet 192.168.10.248
```

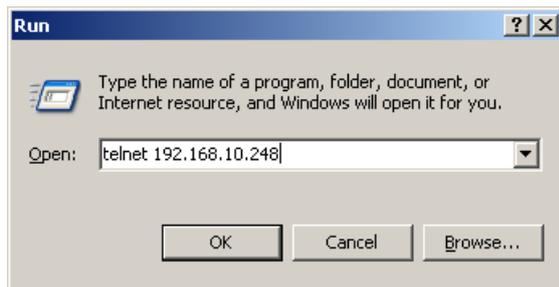


Bild 13: Aufruf des Telnet-Clients unter Windows

Innerhalb des Terminal-Fensters (bei Verbindung über COM0) bzw. des Telnet-Fensters (bei Verwendung von ETH0) ist die Anmeldung am PLCcore-9G20 möglich. Bei Auslieferung des Moduls ist zur Administration des PLCcore-9G20 folgendes Nutzerkonto vorkonfiguriert (siehe auch Abschnitt 7.7):

User: *PlcAdmin*  
Passwort: *Plc123*

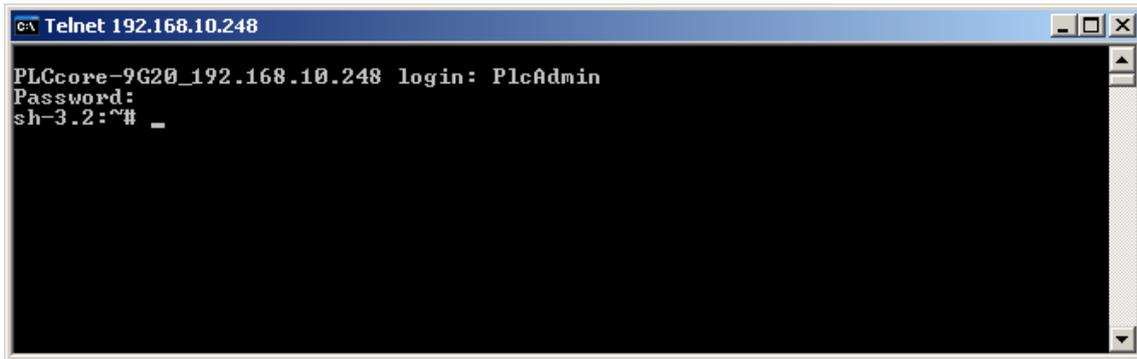


Bild 14: Anmeldung am PLCcore-9G20

Bild 14 verdeutlicht am Beispiel die Anmeldung am PLCcore-9G20 mit Hilfe des in Windows standardmäßig enthaltenen Telnet-Clients.

## 7.8.2 Anmeldung am FTP-Server

Das PLCcore-9G20 verfügt über einen FTP-Server (FTP Daemon), der den Austausch von Dateien mit einem PC ermöglicht (Up- und Download von Dateien). Aus Sicherheits- und Performance-Gründen ist dieser FTP-Server jedoch standardmäßig deaktiviert und muss bei Bedarf zunächst manuell gestartet werden. Hierzu ist zunächst die im Abschnitt 7.8.1 beschriebene Anmeldung an der Kommando-Shell des PLCcore-9G20 durchzuführen. Anschließend ist im Telnet- bzw. Terminal-Fenster folgendes Kommando einzugeben:

```
pureftp
```

Bild 15 verdeutlicht am Beispiel das Starten des FTP-Servers.

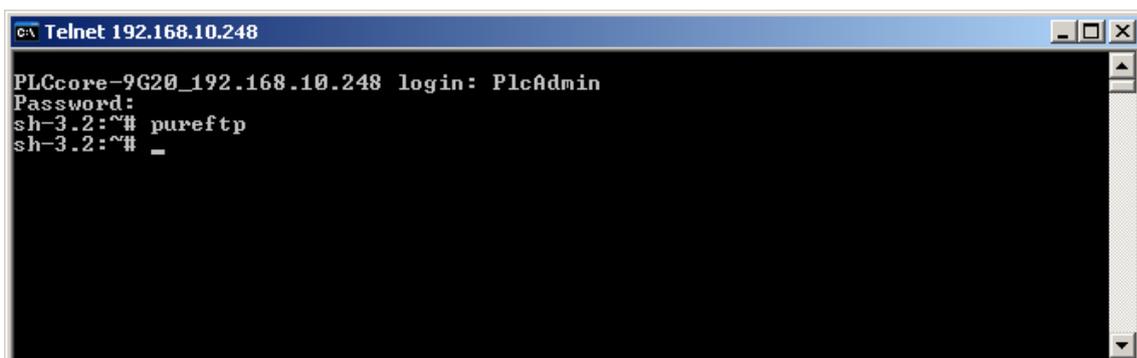


Bild 15: Starten des FTP-Servers

**Hinweis:** Durch Einfügen des Kommandos *"pureftp"* in das Startskript *"/home/etc/autostart"* lässt sich der Aufruf des FTP-Servers beim Booten des PLCcore-9G20 automatisieren (siehe dazu Abschnitt 7.5).

Als FTP-Client für den PC eignet sich beispielsweise das als Open-Source verfügbare *"WinSCP"* (siehe Abschnitt 7.1), das lediglich aus einer einzelnen EXE-Datei besteht, die keine Installation erfordert und sofort gestartet werden kann. Nach dem Programmstart erscheint zunächst der Dialog *"WinSCP Login"* (siehe Bild 16), in dem folgende Einstellungen vorzunehmen sind:

File protocol: FTP  
 Host name: die im Abschnitt 7.3 festgelegte IP-Adresse für das PLCcore-9G20  
 User name: PlcAdmin (für vorkonfiguriertes Nutzerkonto, siehe Abschnitt 7.7)  
 Password: Plc123 (für vorkonfiguriertes Nutzerkonto, siehe Abschnitt 7.7)

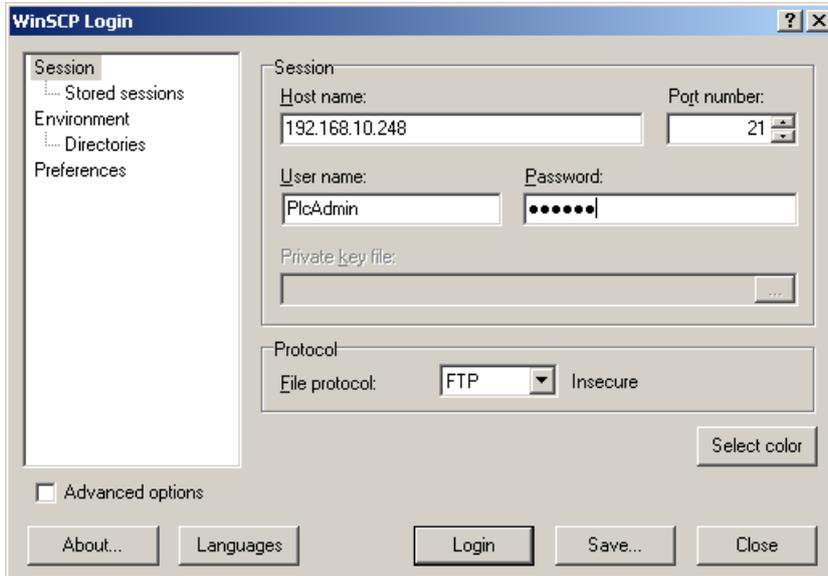


Bild 16: Login-Einstellungen für WinSCP

Nach dem Betätigen der Schaltfläche "Login" meldet sich der FTP-Client am PLCcore-9G20 an und listet im rechten Fenster den aktuellen Inhalt des Verzeichnisses "/home" auf. Bild 17 zeigt den FTP-Client "WinSCP" nach der erfolgreichen Anmeldung am PLCcore-9G20.

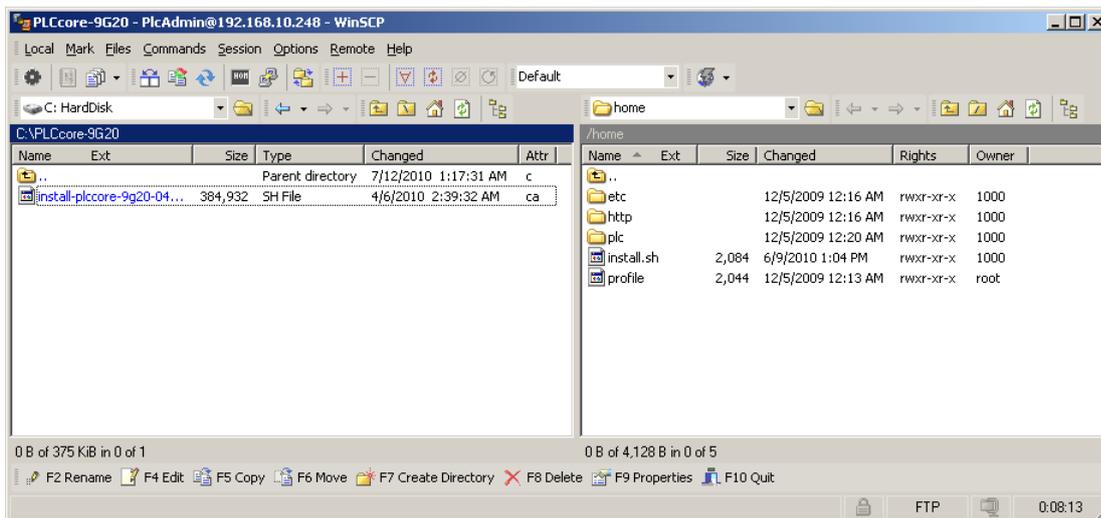


Bild 17: FTP-Client für Windows "WinSCP"

Nach einer erfolgreichen Anmeldung lassen sich innerhalb des FTP-Clients "WinSCP" mit Hilfe der Taste "F4" bzw. der Schaltfläche "F4 Edit" Konfigurationsdateien auf dem PLCcore-9G20 bearbeiten (dazu Transfermodus "Text" selektieren). Mit Hilfe der Taste "F5" bzw. der Schaltfläche "F5 Copy" können Dateien zwischen dem PC und dem PLCcore-9G20 transferieren werden, um beispielsweise

Datensicherungen des PLCcore-9G20 durchzuführen oder um Installationsdateien für Firmware-Updates auf das Modul zu übertragen (dazu Transfermodus "Binary" selektieren).

## 7.9 Anlegen und Löschen von Nutzerkonten

Das Anlegen und Löschen von Nutzerkonten erfordert zunächst die Anmeldung am PLCcore-9G20 wie in Abschnitt 7.8.1 beschrieben.

Das **Anlegen** eines neuen Nutzerkontos erfolgt mit Hilfe des Linux-Kommandos "adduser". Da es bei einem Embedded System wie dem PLCcore-9G20 nicht sinnvoll ist, für jeden Benutzer ein eigenes Verzeichnis anzulegen, ist mit dem Parameter "-H" das Erstellen eines neuen Verzeichnisses zu unterbinden. Stattdessen wird dem neuen Benutzer über den Parameter "-h /home" das bereits vorhandene Verzeichnis "/home" zugeordnet. Zum Anlegen eines neuen Nutzerkontos auf dem PLCcore-9G20 ist das Linux-Kommandos "adduser" wie folgt anzuwenden:

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

Bild 18 verdeutlicht am Beispiel das Anlegen eines neuen Kontos auf dem PLCcore-9G20 für den Nutzer "admin2".

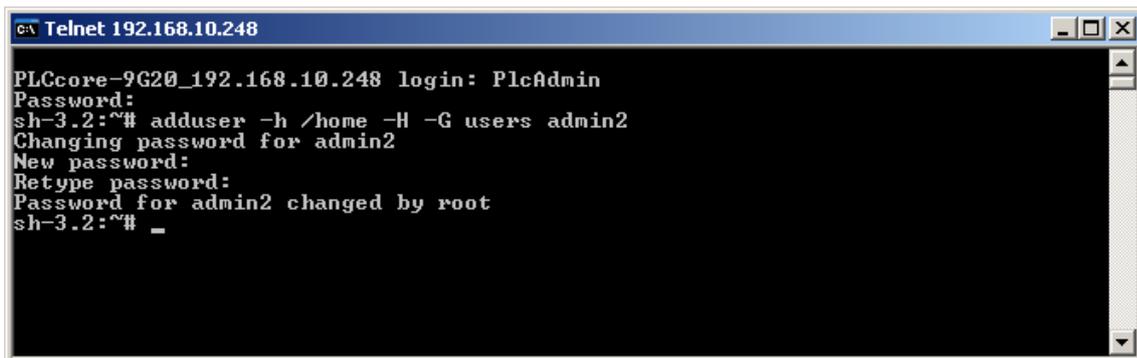


Bild 18: Anlegen eines neuen Nutzerkontos

**Hinweis:** Falls das neu angelegte Benutzerkonto für den Zugriff auf das WEB-Frontend genutzt werden soll, ist der Benutzername gegebenenfalls noch in die Konfigurationsdatei "plccore-9g20.cfg" einzutragen (für Details zur Anmeldung an das WEB-Frontend siehe Abschnitte 7.4.1 und 7.4.3).

Zum **Löschen** eines vorhandenen Nutzerkontos vom PLCcore-9G20 ist das Linux-Kommando "deluser" unter Angabe des Benutzernamens zu verwenden:

```
deluser <username>
```

## 7.10 Passwort eines Nutzerkontos ändern

Das Ändern von Passwörtern erfordert zunächst die Anmeldung am PLCcore-9G20 wie in Abschnitt 7.8.1 beschrieben.

Zum Ändern des Passwortes für ein auf dem PLCcore-9G20 vorhandenes Nutzerkonto ist das Linux-Kommando "passwd" unter Angabe des Benutzernamens zu verwenden:

```
passwd <username>
```

Bild 19 verdeutlicht am Beispiel das Ändern des Passwortes für den Nutzer "PlcAdmin".

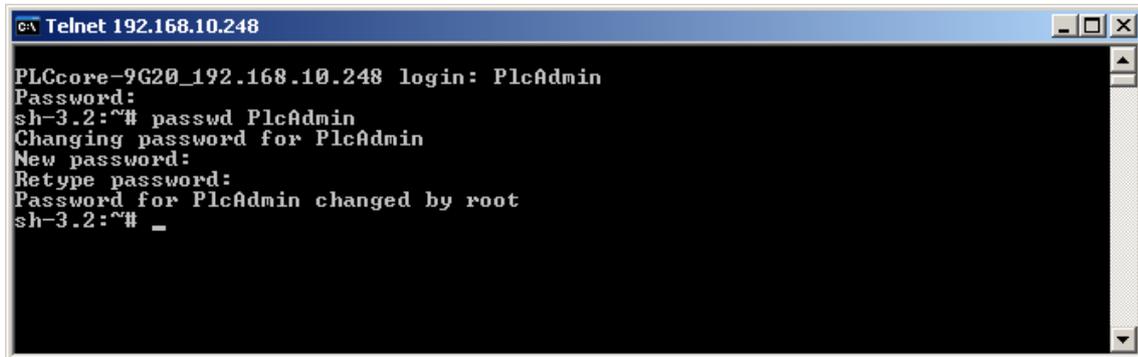


Bild 19: Passwort eines Nutzerkontos ändern

## 7.11 Setzen der Systemzeit

Das Setzen der Systemzeit erfordert zunächst die Anmeldung am PLCcore-9G20 wie in Abschnitt 7.8.1 beschrieben.

Das Setzen der Systemzeit für das PLCcore-9G20 erfolgt in zwei Schritten. Zuerst sind Datum und Uhrzeit mit Hilfe des Linux-Kommandos "date" auf die aktuellen Werte zu setzen. Anschließend wird die Systemzeit durch das Linux-Kommando "hwclock -w" in den RTC-Baustein des PLCcore-9G20 übernommen.

Das Linux-Kommando "date" besitzt folgenden Aufbau:

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

### Beispiel:

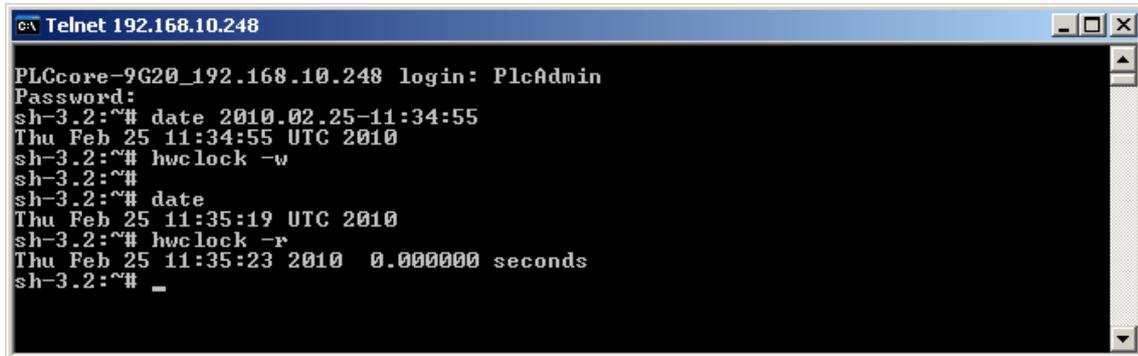
```
date 2010.02.25-11:34:55
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
+----- Sekunde
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
+----- Minute
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
+----- Stunde
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
+----- Tag
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
+----- Monat
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
+----- Jahr
```

Um die aktuelle Systemzeit des PLCcore-9G20 wie im obigen Beispiel dargestellt auf den 2010/02/25 um 11:34:55 zu setzen, ist folgende Befehlssequenz notwendig:

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

Die aktuelle Systemzeit wird durch Eingabe des Linux-Kommandos "date" (ohne Parameter) angezeigt, die aktuellen Werte der RTC lassen sich durch das Linux-Kommando "hwclock -r"

abfragen. Mit `"hwclock -s"` werden die aktuellen Werte der RTC als Systemzeit für Linux übernommen (Synchronisation des Kernels auf die RTC). Bild 20 verdeutlicht das Setzen und Anzeigen der Systemzeit am Beispiel.



```
CA Telnet 192.168.10.248
PLCcore-9G20_192.168.10.248 login: PlcAdmin
Password:
sh-3.2:~# date 2010.02.25-11:34:55
Thu Feb 25 11:34:55 UTC 2010
sh-3.2:~# hwclock -w
sh-3.2:~#
sh-3.2:~# date
Thu Feb 25 11:35:19 UTC 2010
sh-3.2:~# hwclock -r
Thu Feb 25 11:35:23 2010  0.000000 seconds
sh-3.2:~# _
```

Bild 20: Setzen und Anzeigen der Systemzeit

Beim Starten des PLCcore-9G20 werden Datum und Uhrzeit der RTC als aktuelle Systemzeit für das Modul übernommen. Das dazu notwendige Linux-Kommando `"hwclock -s"` ist bereits im Startskript `"/etc/init.d/hwclock"` enthalten.

## 7.12 Dateisystem des PLCcore-9G20

Das auf dem PLCcore-9G20 vorinstallierte Embedded Linux stellt den überwiegenden Teil des on-board Speichers in Form einer Flash-Disk zur Verfügung. Als Dateisystem wird dabei JFFS2 verwendet, ein im embedded Bereich weit verbreitetes System, das speziell für den Einsatz von Flash-Bausteinen als Datenträger entwickelt wurde. Der on-board Flash des PLCcore-9G20 ist in zwei unabhängige JFFS2-Partitionen aufgeteilt. Die erste Partition beinhaltet den Linux-Kernel und das Root-Filesystem. Die zweite Partition wird komplett als Verzeichnis `"/home"` eingebunden. Auf beide Partitionen besteht voller Schreib-/Lesezugriff.

In die als Verzeichnis `"/home"` ansprechbare Partition werden sämtliche vom Anwender modifizierbaren und updatefähigen Files wie beispielsweise Konfigurationsdateien, SPS-Firmware sowie die auf das Modul geladenen SPS-Programm-Dateien abgelegt. Das Verzeichnis `"/tmp"` ist in seiner Größe so dimensioniert, dass es als Zwischenpuffer für den FTP-Download des Firmware-Archives beim Update der SPS-Firmware benutzt werden kann (siehe Abschnitt 7.13.1).

Tabelle 19: Dateisystemkonfiguration des PLCcore-9G20

Zweig	Nutzbare Größe	Beschreibung
/home	4096 kByte	Flash-Disk, zur persistenten Ablage vom Anwender modifizierbarer und updatefähiger Files (z.B. Konfigurationsdateien, SPS-Firmware, SPS-Programm), Datenerhalt bei Spannungsausfall ist gegeben
/tmp	8192 kByte	RAM-Disk, geeignet als Zwischenpuffer für FTP-Downloads, aber kein Datenerhalt bei Spannungsausfall
/var/log	1024 kByte	RAM-Disk, wird vom System selbst zur Ablage temporärer Dateien benutzt, aber kein Datenerhalt bei Spannungsausfall
/mnt		Ziel für die Einbindung von Remote-Verzeichnissen, wird auf dem PLCcore-9G20 in der Standardfunktionalität nicht verwendet

Die konfigurierten sowie jeweils aktuell noch verfügbaren Größen der einzelnen Dateisystemzweige können mit Hilfe des Linux-Kommandos "df" ("DiskFree") ermittelt werden (siehe Bild 21).

```

c:\ Telnet 192.168.10.248
PLCcore-9G20_192.168.10.248 login: PlcAdmin
Password:
sh-3.2:~# df
Filesystem          1024-blocks    Used Available Use% Mounted on
/dev/root            11776          7100    4676    60% /
tmpfs                256             0      256     0% /dev
none                 8192             4    8188     0% /tmp
none                 1024             12   1012     1% /var/log
none                 256              8    248     3% /var/run
none                 256              0    256     0% /var/lock
/dev/mtdblock2      4096           1120   2976    27% /home
sh-3.2:~# _

```

Bild 21: Anzeige von Informationen zum Dateisystem

Einzelheiten zur Anmeldung am System und zum Umgang mit der Linux-Kommando-Shell des PLCcore-9G20 behandelt Abschnitt 7.8.

## 7.13 Software-Update des PLCcore-9G20

Bei Auslieferung ab Werk sind bereits sämtliche für den Betrieb des PLCcore-9G20 notwendigen Firmwarekomponenten auf dem Modul installiert. Ein Firmwareupdate ist daher nur in Ausnahmefällen erforderlich, um beispielsweise eine aktuellere Software mit neuen Eigenschaften einzuspielen.

### 7.13.1 Update der SPS-Firmware

Als SPS-Firmware wird die Laufzeitumgebung der SPS bezeichnet. Die **SPS-Firmware** kann nur vom Hersteller generiert und modifiziert werden, sie ist nicht identisch mit dem **SPS-Anwenderprogramm**, das vom Nutzer der SPS erstellt wird. Das SPS-Anwenderprogramm wird direkt aus der OpenPCS-Programmierungsumgebung heraus auf das Modul übertragen, hierzu ist keinerlei externe Zusatzsoftware erforderlich.

Ein Update der SPS-Firmware erfordert sowohl die Anmeldung an der Kommando-Shell des PLCcore-9G20 wie in Abschnitt 7.8.1 beschrieben als auch die Anmeldung am FTP-Server gemäß Abschnitt 7.8.2.

Das Update der SPS-Firmware erfolgt durch ein selbst-extrahierendes Firmware-Archiv, das per FTP auf das PLCcore-9G20 übertragen wird. Nach dem Start des FTP-Servers auf dem PLCcore-9G20 (Kommando *"pureftp"*, siehe Abschnitt 7.8.2) kann das entsprechende Firmware-Archiv in das Verzeichnis *"/tmp"* des PLCcore-9G20 übertragen werden (siehe Bild 22).

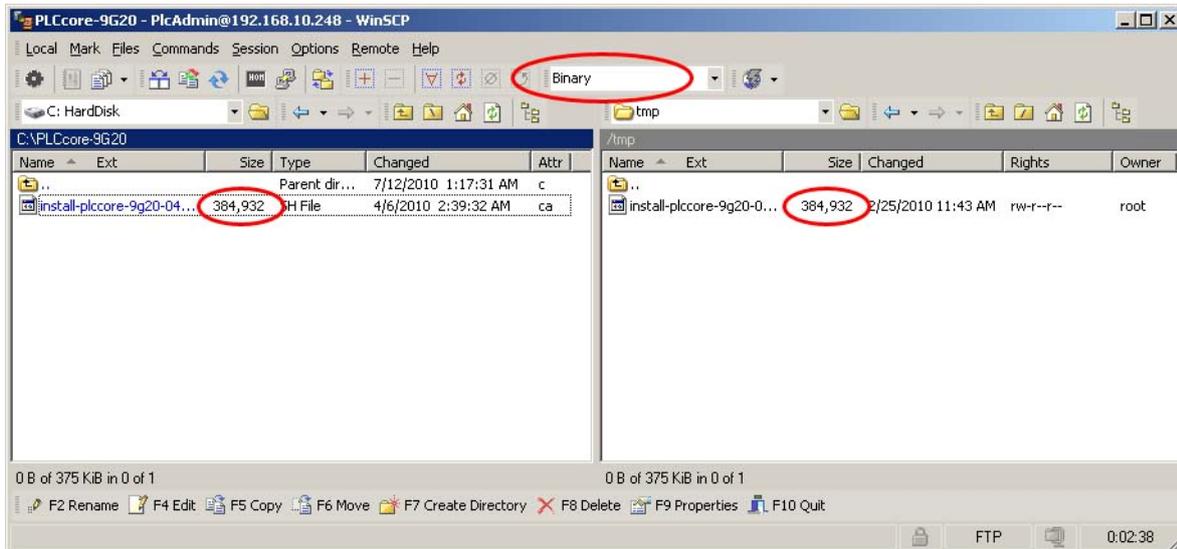


Bild 22: Dateitransfer im FTP-Client "WinSCP"

**Wichtig:** Für den FTP-Transfer des Firmware-Archives ist der Übertragungstyp *"Binary"* auszuwählen. Bei Verwendung des FTP-Clients *"WinSCP"* ist hierzu in der Menüleiste der entsprechende Transfermodus zu selektieren. Nach dem Download des Firmware-Archives ist zu kontrollieren, dass die übertragene Datei auf dem PLCcore-9G20 exakt dieselbe Größe besitzt wie die Originaldatei auf dem PC (siehe Bild 22). Eine abweichende Größe deutet auf einen falschen Übertragungsmodus hin (z.B. *"Text"*). In diesem Fall ist der Download mit dem Übertragungstyp *"Binary"* zu wiederholen.

Nach dem Download des selbst-extrahierenden Archives muss die SPS-Firmware auf dem PLCcore-9G20 installiert werden. Dazu sind im Telnet-Fenster nachfolgende Befehle einzugeben. Hier ist zu beachten, dass der Dateiname für das Firmware-Archiv eine Versionskennung beinhaltet (z.B. *"install-plccore-9G20-0412\_0100.sh"* für die Version 4.12.01.00). Diese Nummer ist bei der Befehlseingabe entsprechend anzupassen:

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

**Tip:** Die Kommando-Shell auf dem PLCcore-9G20 kann angefangene Namen durch drücken der Tab-Taste automatisch vervollständigen (*"tab completion"*), so dass es in der Regel genügt, nur den Anfang des Dateinamens einzugeben und dann vom System automatisch ergänzen zu lassen. So wird z.B. *"/ins"* durch Drücken der Tab-Taste automatisch zu *"/install-plccore-9g20-0412\_0100.sh"* erweitert.

```

CA Telnet 192.168.10.248
PLCcore-9G20_192.168.10.248 login: PlcAdmin
Password:
sh-3.2:~# pureftp
sh-3.2:~# cd /tmp
sh-3.2:/tmp# chmod +x ./install-plccore-9g20-0412_0100.sh
sh-3.2:/tmp# ./install-plccore-9g20-0412_0100.sh

--- PLCcore-9G20 Runtime System Installer ---

Checking PLCcore-9G20 hardware for update requirements...
Extract new I/O driver './plc/pc9g20drv.ko' to tmp dir...
./plc/pc9g20drv.ko
Try to load new I/O driver...
PLCcore-9G20 hardware check ok.

Running installation... please wait

./etc/
./etc/autostart
./etc/rc.usr
./http/
./http/mime.types
./http/boa.conf
./http/cgi-bin/
./http/cgi-bin/cfgsetup.cfg
./http/cgi-bin/cfgsetup.cgi
./http/html/
./http/html/systec_logo.jpg
./http/html/index.html
./http/html/Pc9G20Config.html
./http/html/PLCcore-9G20.gif
./install.sh
./plc/
./plc/version
./plc/iodrvdemo
./plc/candrv.ko
./plc/plccore-9g20-z5
./plc/stopplc
./plc/pc9g20drv.ko
./plc/pc9g20drv.so
./plc/runplc
./plc/shpingdemo
./plc/plccore-9g20-z4
./plc/plccore-9g20.cfg

Installation has been finished.
Please restart system to activate the new firmware.

sh-3.2:/tmp# _

```

Bild 23: Installation der SPS-Firmware auf dem PLCcore-9G204

Bild 23 verdeutlicht die Installation der SPS-Firmware auf dem PLCcore-9G20. Nach einem Reset startet das Modul anschließend mit der aktualisierten Firmware.

**Hinweis:** Bei einem Update der SPS-Firmware wird auch die Konfigurationsdatei `"/home/plc/plccore-9g20.cfg"` überschrieben, was zu einem Rücksetzen der SPS-Konfiguration auf Standardeinstellungen führt. Daher sollte nach einem Update die Konfiguration wie im Abschnitt 7.4 beschrieben kontrolliert und ggf. neu gesetzt werden.

### 7.13.2 Update des Linux-Images

Der Update des Linux-Images erfolgt via TFTP (Trivial FTP) innerhalb des Linux-Bootloaders "U-Boot". Auf dem PC ist hierfür ein entsprechender TFTP-Server notwendig, beispielsweise die Freeware "TFTPD32" (siehe Abschnitt 7.1). Das Programm besteht lediglich aus einer einzelnen EXE-Datei, die keine Installation erfordert und sofort gestartet werden kann. Nach dem Programmstart sollte ein geeignetes Arbeitsverzeichnis ("Current Directory") durch Anklicken der Schaltfläche

"Browse" eingestellt werden (z.B. "C:\PLCcore-9G20"). In diesem Verzeichnis muss sich Linux-Image für das PLCcore-9G20 befinden ("root.sum.jffs2").

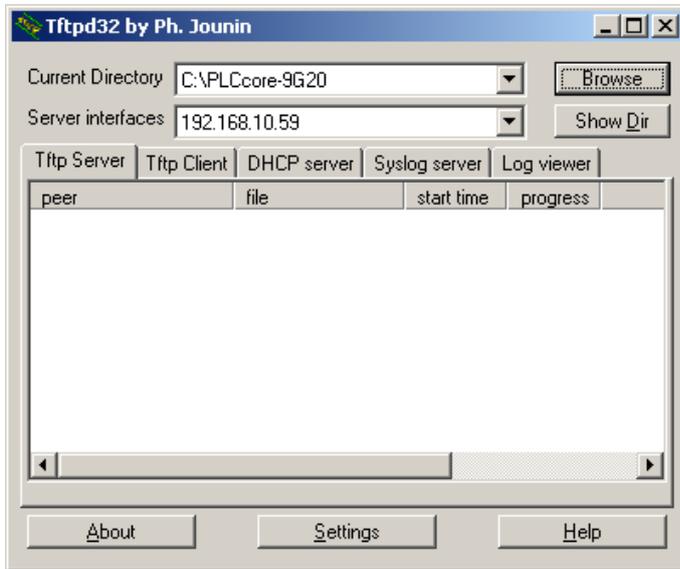


Bild 24: TFTP-Server für Windows "TFTPD32"

**Voraussetzung** für den TFTP-Download des Linux-Images **ist die abgeschlossenen Ethernet-Konfiguration** des PLCcore-9G20 **gemäß Abschnitt 7.3**. Für das Update des Linux-Images ist neben der Ethernet-Verbindung noch eine serielle Verbindung zum PLCcore-9G20 erforderlich. Hier gelten ebenfalls die im Abschnitt 7.2 beschriebenen Einstellungen für das Terminalprogramm (115200 Baud, 8 Datenbits, 1 Stopbit, keine Parität, keine Flusskontrolle).

**Ein Update des Linux-Images auf dem PLCcore-9G20 ist nur möglich, wenn Linux noch nicht gestartet ist. Daher ist vor dem Update der Linux-Autostart zu unterbinden und stattdessen zum "U-Boot" Kommandoprompt zu wechseln. Die dazu notwendigen Schritte beschreibt Abschnitt 7.2.**

Nach Reset (z.B. Taster S405 am Developmentboard) meldet sich der "U-Boot" Kommandoprompt. Hier sind zum Update des Linux-Images die im Folgenden beschriebenen Kommandos in der aufgeführten Reihenfolge einzugeben:

Tabelle 20: Kommando-Sequenz zum Update des Linux-Images auf dem PLCcore-9G20

Kommando	Bedeutung
<code>setenv serverip &lt;host_ip_addr&gt;</code>	Setzen der IP-Adresse des TFTP-Servers. Bei Verwendung von "TFTPD32" wird diese auf dem PC im Feld "Server Interface" angezeigt
<code>tftp root.sum.jffs2</code>	Download des Linux-Images vom Entwicklungs-PC auf das PLCcore-9G20
<code>erase nor0,3</code>	Löschen des vom Linux-Image benötigten Flash-Bereiches
<code>cp.b \${fileaddr} 0x10480000 \${filesize}</code>	Speichern des Linux-Images im Flash des PLCcore-9G20



## 8 Adaption von Ein-/Ausgängen sowie Prozessabbild

### 8.1 Datenaustausch über Shared Prozessabbild

#### 8.1.1 Übersicht zum Shared Prozessabbild

Das PLCcore-9G20 basiert auf Embedded Linux als Betriebssystem. Dadurch ist es möglich, simultan zur SPS-Firmware noch weitere, anwenderspezifische Programme abzuwickeln. Über ein gemeinsam genutztes Prozessabbild (Shared Prozessabbild) können das SPS-Programm und eine anwenderspezifische C/C++ Applikation Daten miteinander austauschen. Voraussetzung für die Implementierung anwenderspezifischer C/C++ Applikationen ist das **Softwarepaket SO-1105** ("VMware-Image des Linux-Entwicklungssystems für das ECUCore-9G20").

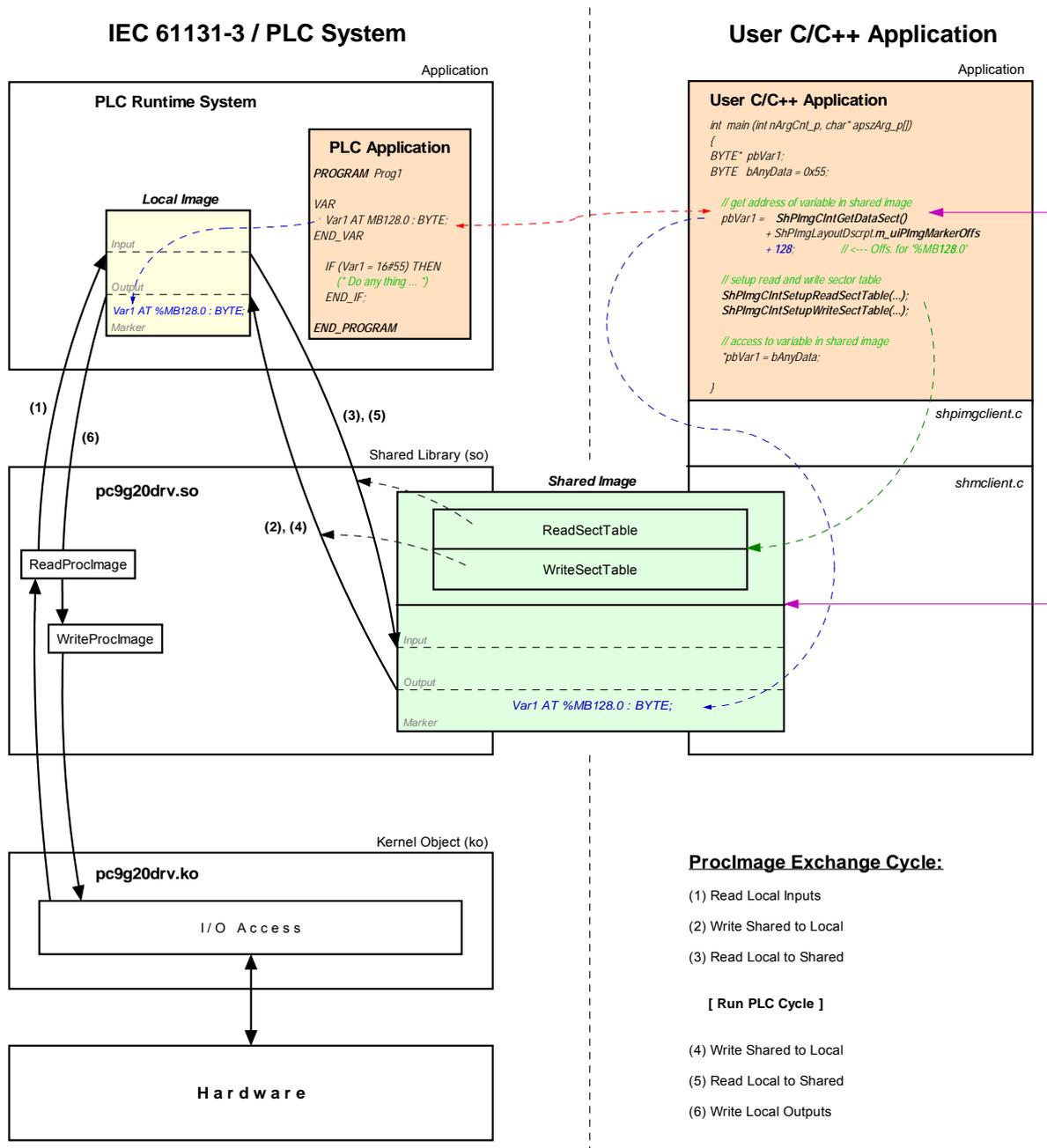


Bild 26: Übersicht Shared Prozessabbild

Für eine C/C++ Applikation sind alle Variablen über das Shared Prozessabbild nutzbar, die das SPS-Programm als direkt adressierte Variablen im Prozessabbild anlegt. Wie in Bild 26 dargestellt, werden für den Datenaustausch mit einer externen Applikation innerhalb des SPS-Laufzeitsystems zwei getrennte Prozessabbilder benutzt. Das ist notwendig, um die in der IEC 61131-3 festgelegte Forderung zu erfüllen, nach der sich das Eingangs-Prozessabbild einer SPS während der gesamten Ausführungszeit eines SPS-Programmzyklus nicht verändern darf. Das SPS-Programm operiert dabei stets mit dem internen, innerhalb des SPS-Laufzeitsystems lokal angelegten Prozessabbild ("Local Image" in Bild 26). Dieses ist durch das SPS-Laufzeitsystem gekapselt und somit vor direkten Zugriffen von außen geschützt. Die anwenderspezifische, externe C/C++ Applikation benutzt dagegen stets das Shared Prozessabbild ("Shared Image" in Bild 26). Durch die Aufspaltung in zwei getrennte Prozessabbilder wird zudem eine Entkopplung der Zugriffe zwischen SPS-Programm und externer Applikation erreicht. Eine Synchronisation der beiden parallel laufenden, unabhängigen Prozesse ist nur noch für die kurze Zeitspanne des Umkopierens der Prozessdaten notwendig.

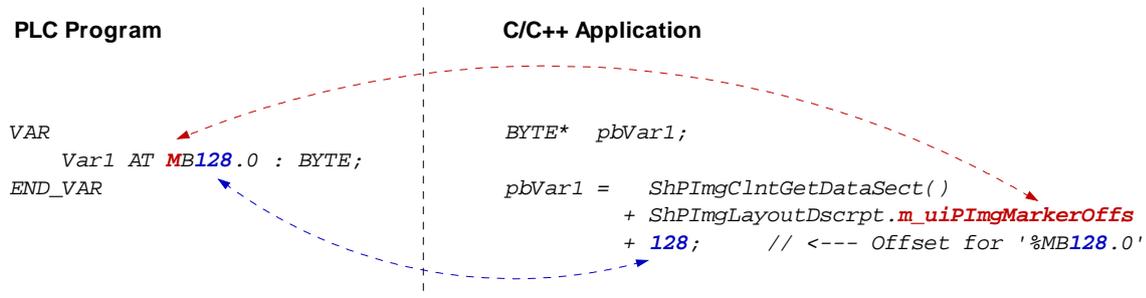
Um den Datenaustausch mit externen Applikationen zu ermöglichen, muss die **Option "Share PLC process image"** in der SPS-Konfiguration aktiviert sein (siehe Abschnitt 7.4.1). Alternativ kann auch der Eintrag `"EnableSharing="` in der Sektion `"[Proclmg]"` innerhalb der Konfigurationsdatei `"/home/plc/plccore-9g20.cfg"` direkt gesetzt werden (siehe Abschnitt 7.4.3). Die entsprechende Konfigurationseinstellung wird beim Starten der SPS-Firmware ausgewertet. Ist die Option `"Share PLC process image"` aktiviert, legt die SPS-Firmware ein zweites Prozessabbild als Shared Memory an ("Shared Image" in Bild 26), das explizit für den Datenaustausch mit externen Applikationen vorgesehen ist. Die SPS-Firmware agiert hier als Server, die externe, anwenderspezifische C/C++ Applikation übernimmt die Rolle des Clients.

Das Kopieren der Daten zwischen beiden Prozessabbildern wird über die **ReadSectorTable** und die **WriteSectorTable** gesteuert. Beide Tabellen werden durch den Client (externe, anwenderspezifische C/C++ Applikation) befüllt und vom Server (SPS-Laufzeitsystem) abgearbeitet. Der Client definiert hier die Bereiche des SPS-Prozessabbildes, aus denen er Daten lesen (*ReadSectorTable*) bzw. in die er Daten schreiben möchte (*WriteSectorTable*). Die Begriffe `"Read"` und `"Write"` beziehen sich somit auf die Datentransferrichtungen aus Sicht des Clients.

Die zu lesenden und zu schreibenden Sektionen können alle Bereiche des gesamten Prozessabbildes umfassen, also sowohl Input-, Output- als auch Merkerbereich. Damit ist es für eine Client-Applikation beispielsweise auch möglich, in den Input-Bereich des SPS-Prozessabbildes zu schreiben und Daten aus dem Output-Bereich zu lesen. Die Reihenfolge der einzelnen Lese- und Schreiboperationen ist in Bild 26 dargestellt. So werden vor der Ausführung eines SPS-Programmzyklus zunächst die physikalischen Eingänge in das lokale Prozessabbild der SPS eingelesen (1), anschließend erfolgt die Übernahme der in der *WriteSectorTable* definierten Bereiche aus dem Shared Prozessabbild in das lokale Prozessabbild (2). Unter Ausnutzung dieser Reihenfolge ist es einer Client-Applikation z.B. auch möglich, den Wert eines physikalischen Eingangs zu überschreiben, was sich sowohl für Simulationszwecke ausnutzen lässt als auch zum Festsetzen von Eingangsdaten auf konstante Werte (`"Forcen"`). Analog dazu werden vor dem Schreiben des Prozessabbildes auf die physikalischen Ausgänge (6) zunächst wiederum die in der *WriteSectorTable* definierten Bereiche aus dem Shared Prozessabbild in das lokale Prozessabbild übernommen (4). Damit ist eine Client-Applikation gleichfalls in der Lage, auch die vom SPS-Programm generierten Ausgangsinformationen zu überschreiben.

Der **Aufbau des Prozessabbildes** wird von der jeweiligen SPS-Firmware fest vorgegeben. Die Client-Applikation erhält die Informationen zum Aufbau des Prozessabbildes über die Funktion ***ShPlmgClntSetup()***. Diese trägt die Startoffsets und Größen von Input-, Output- und Merkerbereich in die beim Aufruf übergebene Struktur vom Typ `tShPlmgLayoutDscrpt` ein. Die Startadresse des Shared Prozessabbildes liefert die Funktion ***ShPlmgClntGetDataSect()***. Bei der Definition einer Variablen im SPS-Programm wird deren absolute Position im Prozessabbild durch Bereich (%I = Input, %Q = Output, %M = Merker) und Offset (z.B. `%MB128.0`) bestimmt. Dabei beginnt der Offset in jedem Bereich wieder mit Null, so dass beispielsweise das Anlegen einer Variablen im Merkerbereich unabhängig von den Größen der davor liegenden Input- und Output-Bereiche erfolgt. Für den Austausch von Daten zwischen SPS-Programm und externer Applikation ist ein korrespondierendes **Variablen-Paar** jeweils im SPS-Programm und in der C/C++ Applikation anzulegen. Dazu ist auf beiden Seiten dieselbe Adresse zu referenzieren. Um bei der Definition der entsprechenden Variablen

im C/C++ Programm ein zum SPS-Programm vergleichbares Adressierungsschema anwenden zu können, spiegelt die Struktur *tShPIImgLayoutDscrpt* den physikalischen Aufbau des Prozessabbildes in der SPS-Firmware mit Input-, Output- und Merkerbereich wider. Damit erfolgt auch im C/C++ Programm die Definition einer Variablen im Shared Prozessabbild unter Angabe des jeweiligen Bereiches und des darauf bezogenen Offsets. Das nachfolgende Beispiel verdeutlicht das Anlegen eines korrespondierenden Variablen-Paares in SPS-Programm und C/C++ Applikation:



Wie bereits weiter oben beschrieben wird der Kopiervorgang zum Austausch des Variableninhaltes zwischen SPS- und C/C++ Programm über **ReadSectorTable** und **WriteSectorTable** gesteuert. Um im dargestellten Beispiel den Wert der Variable von der C/C++ Applikation zum SPS-Programm zu übertragen, ist vom Client (C/C++ Applikation) ein entsprechender Eintrag in der *WriteSectorTable* zu definieren (*WriteSectorTable* da Client die Variable zum Server "schreibt"):

```
// specify offset and size of 'Var1' and define sync type (always or on demand?)
WriteSectTab[0].m_uiPIImgDataSectOffs = ShPIImgLayoutDscrpt.m_uiPIImgMarkerOffs + 128;
WriteSectTab[0].m_uiPIImgDataSectSize = sizeof(BYTE);
WriteSectTab[0].m_SyncType           = kShPIImgSyncOnDemand;

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

Werden in der gleichen Transferrichtung mehrere Variablen-Paare für den Datenaustausch zwischen SPS-Programm und C/C++ Applikation angelegt, dann sollten diese nach Möglichkeit in einem zusammenhängenden Adressbereich definiert werden. Dadurch lassen sie sich in einem gemeinsamen Eintrag der entsprechenden SectorTable zusammenfassen. Als *SectorOffset* ist die Adresse der ersten Variable und als *SectorSize* die Summe der Variablengrößen anzugeben. Durch das Zusammenfassen wird die Effizienz und Performanz der Kopiervorgänge verbessert.

Für jeden Eintrag der *WriteSectorTable* ist außerdem ein entsprechender *SyncType* zu definieren. Dieser legt fest, ob die Sektion generell immer zwischen zwei aufeinander folgenden SPS-Zyklen vom Shared Prozessabbild in das lokale Abbild übernommen wird (**kShPIImgSyncAlways**) oder nur bei Bedarf (**kShPIImgSyncOnDemand**). Bei der Klassifizierung als *SyncOnDemand* werden die Daten nur dann kopiert, wenn die betreffende Sektion zuvor explizit als aktualisiert markiert wurde. Dies erfolgt durch den Aufruf der Funktion **ShPIImgClntWriteSectMarkNewData()** unter Angabe des zugehörigen *WriteSectorTable*-Index (z.B. 0 für *WriteSectTab[0]* usw.).

Für die *ReadSectorTable* ist der *SyncType* fest als **kShPIImgSyncAlways** vorgegeben (der Wert im Memberelement *m\_SyncType* wird ignoriert). Da die SPS-Firmware nicht feststellen kann, welche Variablen das SPS-Programm im vorangegangenen Zyklus verändert hat, werden stets alle in der *ReadSectorTable* definierten Sektionen vom lokalen Abbild in das Shared Prozessabbild übernommen. Die betreffenden Variablen enthalten somit im Shared Prozessabbild immer die aktuellen Werte.

Das Shared Prozessabbild ist eine von der SPS-Firmware und der C/C++ Applikation gemeinsam benutzte Ressource. Um hier Konflikte durch einen zeitgleichen Zugriff beider parallel laufender Prozesse zu verhindern, ist das Shared Prozessabbild intern über eine Semaphore geschützt. Benötigt ein Prozess Zugriff auf das Shared Prozessabbild, betritt er einen kritischen Abschnitt, indem

er zunächst die Semaphore setzt und so den exklusiven Zugriff auf diese Ressource erhält. Möchte der andere Prozess zur selben Zeit ebenfalls auf das Shared Prozessabbild zugreifen, muss er genauso wie der erste Prozess auch einen kritischen Abschnitt betreten, indem er wiederum zunächst versucht, die Semaphore zu setzen. In diesem Fall erkennt das Betriebssystem, dass die Ressource bereits in Benutzung ist. Es blockiert den zweiten Prozess nun so lange, bis der erste Prozess den kritischen Abschnitt wieder verlassen hat und die Semaphore freigibt. Das Betriebssystem stellt dadurch sicher, dass sich immer nur einer der beiden parallel laufenden Prozesse SPS-Laufzeitsystem und C/C++ Applikation im kritischen Abschnitt befinden kann und Zugriff auf das Shared Prozessabbild erhält. Damit sich die beiden Prozesse gegenseitig möglichst wenig in ihrer Abarbeitung behindern, sollten die kritischen Abschnitte so selten wie möglich und dann auch nur so lange wie unbedingt nötig durchlaufen werden. Andernfalls kann es zur Verlängerung der SPS-Zykluszeit sowie zu Laufzeitschwankungen (Jitter) kommen.

Zum Setzen der Semaphore und damit zum Sperren des Shared Prozessabbildes für den exklusiven Zugriff stehen der Client-Applikation die beiden Funktionen **ShPImgClntLockSegment()** zum Betreten sowie **ShPImgClntUnlockSegment()** zum Verlassen des kritischen Abschnitts zur Verfügung. Der Abschnitt zwischen den beiden Funktionen wird auch als geschützter Bereich bezeichnet, in dem die Client-Applikation konkurrenzfreien Zugriff auf das Shared Prozessabbild besitzt. Nur innerhalb eines solchen geschützten Bereiches ist die Konsistenz gelesener bzw. geschriebener Daten gewährleistet. Außerhalb davon kann jederzeit eine Manipulation des Prozessabbildes durch das SPS-Laufzeitsystem erfolgen. Das nachfolgende Beispiel verdeutlicht den exklusiven Zugriff auf das Shared Prozessabbild in der C/C++ Applikation:

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

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

Im angegebenen Beispiel wurde beim Anlegen des *WriteSectorTable*-Eintrages der *SyncType* als *kShPImgSyncOnDemand* definiert. Somit erfolgt eine Übernahme der Variable *Var1* nur dann vom Shared Prozessabbild in das lokale Abbild, wenn die betreffende Sektion zuvor explizit als aktualisiert markiert wurde. Dazu ist der Aufruf der Funktion **ShPImgClntWriteSectMarkNewData()** erforderlich. Da die Funktion *ShPImgClntWriteSectMarkNewData()* die Semaphore nicht verändert, darf sie, wie hier im Beispiel dargestellt, nur in einem geschützten Bereich, also im Code-Abschnitt zwischen *ShPImgClntLockSegment()* und *ShPImgClntUnlockSegment()*, benutzt werden.

Die Synchronisation zwischen lokalem Abbild und Shared Prozessabbild erfolgt durch das SPS-Laufzeitsystem nur zwischen zwei aufeinander folgenden SPS-Zyklen. Einer Client-Applikation (anwenderspezifisches C/C++ Programm) ist dieser Zeitpunkt nicht unmittelbar bekannt, sie kann sich jedoch vom SPS-Laufzeitsystem über die Aktualisierung des Shared Prozessabbild informieren lassen. Dazu muss sie einen Callback-Handler vom Typ *tShPImgAppNewDataSigHandler* definieren, z.B.:

```
static void AppSigHandlerNewData (void)
{
    fNewDataSignaled_1 = TRUE;
}
```

Dieser Callback-Handler ist mit Hilfe der Funktion **ShPImgClntSetNewDataSigHandler()** zu registrieren. Er wird jeweils im Anschluss an eine Synchronisation der beiden Abbilder aufgerufen.

**Der Callback-Handler der Client-Applikation wird im Kontext eines Linux Signal-Handlers gerufen** (das SPS-Laufzeitsystem informiert den Client mit Hilfe der Linux-Funktion *kill()*).

Dementsprechend gelten für den Callback-Handler der Client-Applikation die üblichen **Restriktionen** für Linux Signal-Handler. Insbesondere ist hier nur der Aufruf einiger weniger, explizit als reentranzfest gekennzeichneten Betriebssystem-Funktionen erlaubt. Innerhalb der Client-Applikation muss ebenfalls darauf geachtet werden, dass es nicht zum reentranten Aufruf lokaler Funktionen kommt. Daher sollte, wie im Beispiel gezeigt, innerhalb des Callback-Handlers lediglich ein globales Flag zur Signalisierung gesetzt werden, das dann später in der Hauptschleife der Client-Applikation ausgewertet und verarbeitet wird.

### 8.1.2 API des Shared Prozessabbild Client

Wie im Bild 26 dargestellt, benutzt eine anwenderspezifische C/C++ Applikation ausschließlich das vom *Shared Process Image Client* zur Verfügung gestellte API (Application Programming Interface). Dieses API ist im Headerfile *shpimgclient.h* deklariert und im Sourcefile *shpimgclient.c* implementiert. Es umfasst folgende Typen (teilweise auch in *shpimg.h* definiert) und Funktionen:

#### Struktur *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;
```

Die Struktur *tShPImgLayoutDscrpt* beschreibt den von der SPS-Firmware vorgegebenen Aufbau des Prozessabbildes. Die Client-Applikation erhält die Informationen zum Aufbau des Prozessabbildes über die Funktion *ShPImgClntSetup()*. Diese trägt die Startoffsets und Größen von Input-, Output- und Merkerbereich in die beim Aufruf übergebene Struktur ein.

#### Struktur *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;
```

Die Struktur *tShPImgSectDscrpt* beschreibt den vom Client zu definierenden Aufbau eines *ReadSectorTable*- oder *WriteSectorTable*-Eintrages. Beide Tabellen dienen zur Synchronisation zwischen lokalem Abbild des SPS-Laufzeitsystems und Shared Prozessabbild (siehe Abschnitt 8.1.1). Das Memberelement *m\_uiPImgDataSectOffs* definiert den absoluten Startoffset der Sektion innerhalb des Shared Prozessabbildes. Die jeweiligen Startoffsets von Input-, Output- und Merkerbereich können dazu aus der Struktur *tShPImgLayoutDscrpt* ermittelt werden. Das Memberelement *m\_uiPImgDataSectSize* legt die Größe der Sektion fest, die eine oder mehrere Variablen enthalten kann. Das Memberelement *m\_SyncType* ist nur für Einträge der *WriteSectorTable* relevant und legt fest, ob die Sektion generell immer zwischen zwei aufeinander folgenden SPS-Zyklen vom Shared Prozessabbild in das lokale Abbild übernommen wird (*kShPImgSyncAlways*) oder nur bei Bedarf

**(kShPImgSyncOnDemand)**. Bei der Klassifizierung als *SyncOnDemand* müssen die Daten durch den Aufruf der Funktion *ShPImgClntWriteSectMarkNewData()* als modifiziert gekennzeichnet werden. Sie setzt dazu das Memberelement *m\_fNewData* auf TRUE. Die Client-Applikation sollte dieses Memberelement niemals direkt manipulieren.

### **Funktion ShPImgClntSetup**

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

Die Funktion **ShPImgClntSetup()** initialisiert den *Shared Process Image Client* und verbindet sich mit dem vom SPS-Laufzeitsystem angelegten Speichersegment für das Shared Prozessabbild. Anschließend trägt sie die Startoffsets und Größen von Input-, Output- und Merkerbereich in die beim Aufruf übergebene Struktur vom Typ *tShPImgLayoutDscrpt* ein. Damit erhält die Client-Applikation Kenntnis über den Aufbau des von der SPS-Firmware verwalteten Prozessabbildes.

Ist zum Aufrufzeitpunkt der Funktion kein SPS-Laufzeitsystem aktiv oder hat dieses kein Shared Prozessabbild angelegt (Option "*Share PLC process image*" in der SPS-Konfiguration deaktiviert, siehe Abschnitt 8.1.1), kehrt die Funktion mit dem Returnwert FALSE zurück. Bei erfolgreichem Abschluss der Initialisierung ist der Rückgabewert TRUE.

### **Funktion ShPImgClntRelease**

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

Die Funktion **ShPImgClntRelease()** beendet den *Shared Process Image Client* und trennt die Verbindung zu dem vom SPS-Laufzeitsystem angelegten Speichersegment für das Shared Prozessabbild.

Nach erfolgreicher Ausführung liefert die Funktion den Rückgabewert TRUE, im Fehlerfall ist der Returnwert auf FALSE gesetzt.

### **Funktion ShPImgClntSetNewDataSigHandler**

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

Die Funktion **ShPImgClntSetNewDataSigHandler()** registriert einen applikationsspezifischen Callback-Handler. Dieser Callback-Handler wird im Anschluss an eine Synchronisation der beiden Abbilder aufgerufen. Durch Übergabe von NULL wird ein registrierter Callback-Handler wieder abgemeldet.

Der **Callback-Handler wird im Kontext eines Linux Signal-Handlers gerufen**. Dementsprechend gelten die üblichen **Restriktionen** für Linux Signal-Handler (siehe Abschnitt 8.1.1).

Nach erfolgreicher Ausführung liefert die Funktion den Rückgabewert TRUE, im Fehlerfall ist der Returnwert auf FALSE gesetzt.

### **Funktion *ShPImgClntGetHeader***

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

Die Funktion ***ShPImgClntGetHeader()*** liefert einen Pointer auf die zur Verwaltung des Shared Prozessabbild intern verwendete Struktur vom Typ *tShPImgHeader*. Diese Struktur wird von der Client-Applikation normalerweise nicht benötigt, da alle darin enthaltenen Daten über Funktionen des vom *Shared Process Image Client* zur Verfügung gestellten API gelesen und geschrieben werden können.

### **Funktion *ShPImgClntGetDataSect***

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

Die Funktion ***ShPImgClntGetDataSect()*** liefert einen Pointer auf den Anfang des Shared Prozessabbildes. Dieser Pointer stellt die Basisadresse für alle Zugriffe auf das Shared Prozessabbild, einschließlich der Definition von Sektionen der *ReadSectorTable* und *WriteSectorTable* dar (siehe Abschnitt 8.1.1).

### **Funktionen *ShPImgClntLockSegment* und *ShPImgClntUnlockSegment***

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

Für den exklusiven Zugriff auf das Shared Prozessabbild stehen der Client-Applikation die beiden Funktionen ***ShPImgClntLockSegment()*** zum Betreten sowie als Komplement ***ShPImgClntUnlockSegment()*** zum Verlassen des kritischen Abschnitts zur Verfügung. Der Abschnitt zwischen den beiden Funktionen ist ein geschützter Bereich, in dem die Client-Applikation konkurrenzfreien Zugriff auf das Shared Prozessabbild besitzt (siehe Abschnitt 8.1.1). Nur innerhalb eines solchen geschützten Bereiches ist die Konsistenz gelesener bzw. geschriebener Daten gewährleistet. Außerhalb davon kann jederzeit eine Manipulation des Prozessabbildes durch das SPS-Laufzeitsystem erfolgen. Damit die Client-Applikation das SPS-Laufzeitsystem in seiner Abarbeitung so wenig wie möglich behindert, sollten die kritischen Abschnitte so selten wie möglich und dann auch nur so lange wie unbedingt nötig gesetzt werden. Andernfalls kann es zur Verlängerung der SPS-Zykluszeit sowie zu Laufzeitschwankungen (Jitter) kommen.

Nach erfolgreicher Ausführung liefert die Funktion den Rückgabewert TRUE, im Fehlerfall ist der Returnwert auf FALSE gesetzt.

### **Funktion *ShPImgClntSetupReadSectTable***

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

Die Funktion ***ShPImgClntSetupReadSectTable()*** initialisiert die *ReadSectorTable* mit den vom Client definierten Werten. Der Client legt hier die Bereiche des SPS-Prozessabbildes fest, aus denen er Daten lesen möchte (siehe Abschnitt 8.1.1). Als Parameter *paShPImgReadSectTab\_p* ist die Anfangsadresse eines Feldes aus Elementen der Struktur *tShPImgSectDscrpt* zu übergeben. Der Parameter *uiNumOfReadDscrptUsed\_p* gibt an, wie viele Elemente das Feld besitzt.

Für die *ReadSectorTable* ist der *SyncType* fest als *kShPImgSyncAlways* vorgegeben.

Die maximale Anzahl möglicher Elemente für die *ReadSectorTable* ist durch die Konstante

*SHPIMG\_READ\_SECT\_TAB\_ENTRIES* definiert und kann nur verändert werden, wenn gleichzeitig auch die Shared Library "*pc9g20drv.so*" neu generiert wird (setzt SO-1106 - "Driver Development Kit für das ECUcore-9G20" voraus, siehe Abschnitt 8.2).

Nach erfolgreicher Ausführung liefert die Funktion den Rückgabewert TRUE, im Fehlerfall ist der Returnwert auf FALSE gesetzt.

### **Funktion *ShPImgClntSetupWriteSectTable***

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

Die Funktion ***ShPImgClntSetupWriteSectTable()*** initialisiert die *WriteSectorTable* mit den vom Client definierten Werten. Der Client legt hier die Bereiche des SPS-Prozessabbildes fest, in die er Daten schreiben möchte (siehe Abschnitt 8.1.1). Als Parameter *paShPImgWriteSectTab\_p* ist die Anfangsadresse eines Feldes aus Elementen der Struktur *tShPImgSectDscrpt* zu übergeben. Der Parameter *uiNumOfWriteDscrptUsed\_p* gibt an, wie viele Elemente das Feld besitzt.

In der *WriteSectorTable* ist für jeden Eintrag der *SyncType* zu definieren. Dieser legt fest, ob die Sektion immer zwischen zwei SPS-Zyklen vom Shared Prozessabbild in das lokale Abbild übernommen wird (***kShPImgSyncAlways***) oder nur bei Bedarf (***kShPImgSyncOnDemand***), wenn die betreffende Sektion durch den Aufruf von *ShPImgClntWriteSectMarkNewData()* explizit als aktualisiert markiert wurde.

Die maximale Anzahl möglicher Elemente für die *WriteSectorTable* ist durch die Konstante *SHPIMG\_WRITE\_SECT\_TAB\_ENTRIES* definiert und kann nur verändert werden, wenn gleichzeitig auch die Shared Library "*pc9g20drv.so*" neu generiert wird (setzt SO-1106 - "Driver Development Kit für das ECUcore-9G20" voraus, siehe Abschnitt 8.2).

Nach erfolgreicher Ausführung liefert die Funktion den Rückgabewert TRUE, im Fehlerfall ist der Returnwert auf FALSE gesetzt.

### **Funktion *ShPImgClntWriteSectMarkNewData***

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

Die Funktion ***ShPImgClntWriteSectMarkNewData()*** markiert den Inhalt einer durch die *WriteSectorTable* verwalteten Sektion als modifiziert. Sie wird benutzt, um für Sektionen mit *SyncType* ***kShPImgSyncOnDemand*** das Kopieren der Daten vom Shared Prozessabbild in das lokale Abbild der SPS zu veranlassen.

Da die Funktion *ShPImgClntWriteSectMarkNewData()* direkt auf den Header des Shared Prozessabbildes zugreift, ohne zuvor die Semaphore zu setzen, darf sie nur innerhalb eines geschützten Bereiches, also im Code-Abschnitt zwischen *ShPImgClntLockSegment()* und *ShPImgClntUnlockSegment()*, benutzt werden.

Nach erfolgreicher Ausführung liefert die Funktion den Rückgabewert TRUE, im Fehlerfall ist der Returnwert auf FALSE gesetzt.

## **8.1.3 Erstellen einer anwenderspezifischen Client Applikation**

Voraussetzung für die Implementierung anwenderspezifischer C/C++ Applikationen ist das **Softwarepaket SO-1105 ("VMware-Image des Linux-Entwicklungssystems")**. Diese beinhaltet ein

komplett eingerichtetes Linux-Entwicklungssystem in Form eines VMware-Images und ermöglicht so einen leichten Einstieg in die Softwareentwicklung unter C/C++ für das PLCcore-9G20. Das VMware-Image ist damit die ideale Voraussetzung, um Linux-basierte Anwenderprogramme auf demselben Host-PC zu entwickeln, auf dem auch schon das IEC 61131-Programmiersystem *OpenPCS* installiert ist. Neben der GNU-Crosscompiler Toolchain für ARM9-Prozessoren sind im VMware-Image des Linux-Entwicklungssystems bereits verschiedene, für eine effektive Softwareentwicklung essentielle Server-Dienste vorkonfiguriert und sofort nutzbar. Details zum VMware-Image des Linux-Entwicklungssystems sowie dessen Nutzung beschreibt das "*System Manual ECUcore-9G20*" (Manual-Nr.: L-1253).

Wie im Bild 26 dargestellt, benutzt eine anwenderspezifische C/C++ Applikation das vom *Shared Process Image Client* zur Verfügung gestellte API (Files *shpimgclient.c* und *shpimgclient.h*). Der *Shared Process Image Client* wiederum setzt auf den vom *Shared Memory Client* bereitgestellten Diensten auf (Files *shmclient.c* und *shmclient.h*). Beide Client-Implementierungen sind zur Generierung einer anwenderspezifischen C/C++ Applikation notwendig. Die entsprechenden Files sind im Archiv des *Shared Process Image Demos* (***shpimgdemo.tar.gz***) enthalten. Zur Erstellung eigener anwenderspezifischer Client Applikationen wird dringend empfohlen, dieses Demoprojekt als Ausgangspunkt für eigene Anpassungen und Erweiterungen zu verwenden. Des Weiteren beinhaltet das Demoprojekt ein Makefile mit allen relevanten Konfigurationseinstellungen, die zum Erstellen einer auf dem PLCcore-9G20 lauffähigen Linux-Applikation notwendig sind. Tabelle 21 listet die im Archiv "*shpimgdemo.tar.gz*" enthaltenen Files auf und klassifiziert diese als allgemeingültigen Bestandteil einer C/C++ Applikation bzw. als spezifische Komponente für das Demoprojekt "*shpimgdemo*".

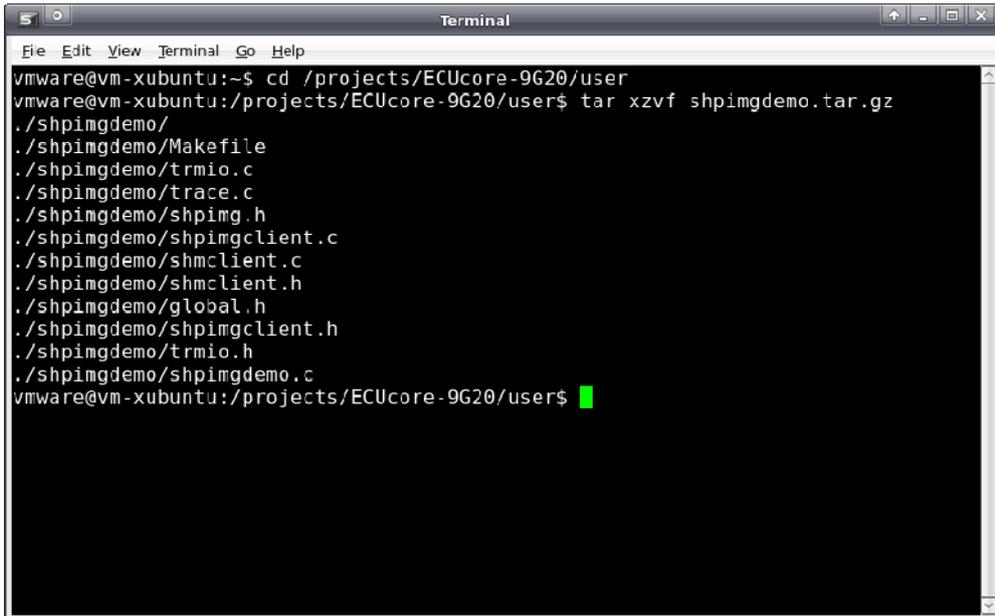
Tabelle 21: Inhalt des Archiv-Files "*shpimgdemo.tar.gz*"

File	Erforderlich für alle C/C++ Applikationen	Spezifisch für Demo " <i>shpimgdemo</i> "
shpimgclient.c	x	
shpimgclient.h	x	
shmclient.c	x	
shmclient.h	x	
shpimg.h	x	
global.h	x	
Makefile	als Vorlage, ist anzupassen	
shpimgdemo.c		x
trmio.c		x
trmio.h		x
trace.c		x

Das Archiv-File "***shpimgdemo.tar.gz***" mit dem *Shared Process Image Demo* ist innerhalb des Linux-Entwicklungssystems in ein beliebiges Unterverzeichnis im Pfad */projects/ECUcore-9G20/user* zu entpacken. Dazu ist das Kommando "*tar*" wie folgt aufzurufen:

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

Das Kommando "*tar*" legt beim Entpacken selbständig das Unterverzeichnis "*shpimgdemo*" an. Wird es beispielsweise im Verzeichnis */projects/ECUcore-9G20/user* aufgerufen, so entpackt es die im Archiv enthaltenen Files in den Pfad */projects/ECUcore-9G20/user/shpimgdemo*". Bild 27 veranschaulicht das Entpacken von "*shpimgdemo.tar.gz*" innerhalb des Linux-Entwicklungssystems.



```

Terminal
File Edit View Terminal Go Help
vmware@vm-xubuntu:~$ cd /projects/ECUcore-9G20/user
vmware@vm-xubuntu:/projects/ECUcore-9G20/user$ tar xzvf shpingdemo.tar.gz
./shpingdemo/
./shpingdemo/Makefile
./shpingdemo/trmio.c
./shpingdemo/trace.c
./shpingdemo/shping.h
./shpingdemo/shpingclient.c
./shpingdemo/shmclient.c
./shpingdemo/shmclient.h
./shpingdemo/global.h
./shpingdemo/shpingclient.h
./shpingdemo/trmio.h
./shpingdemo/shpingdemo.c
vmware@vm-xubuntu:/projects/ECUcore-9G20/user$

```

Bild 27: Entpacken des Archiv-Files *shpingdemo.tar.gz* im Linux-Entwicklungssystem

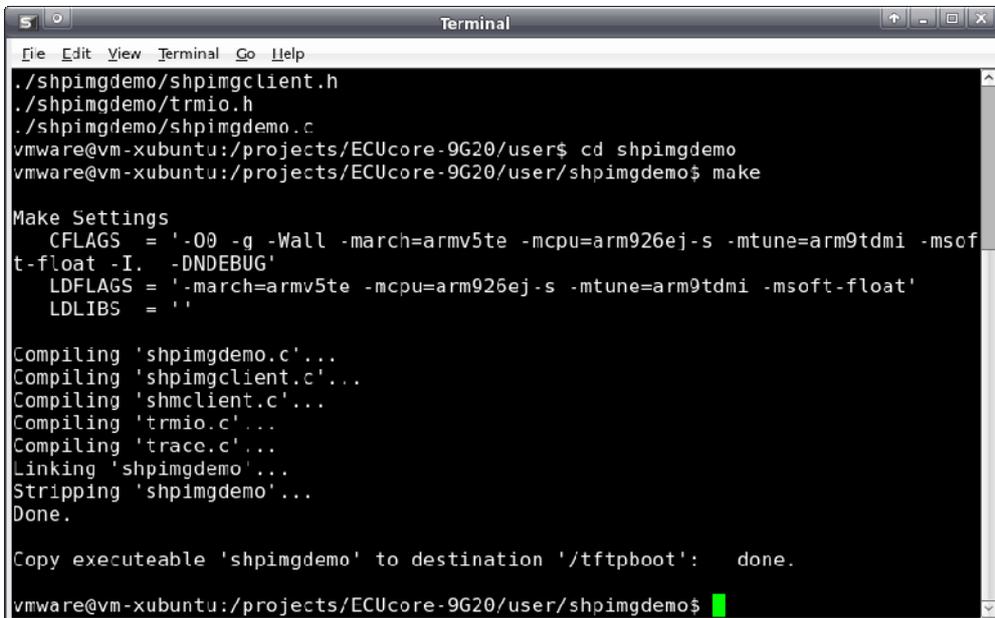
Nach dem Entpacken und Wechseln in das Unterverzeichnis "*shpingdemo*" kann das Demoprojekt durch Aufruf des Kommandos "*make*" erstellt werden:

```

cd shpingdemo
make

```

Bild 28 zeigt die Generierung des Demoprojektes "*shpingdemo*" im Linux-Entwicklungssystem.



```

Terminal
File Edit View Terminal Go Help
./shpingdemo/shpingclient.h
./shpingdemo/trmio.h
./shpingdemo/shpingdemo.c
vmware@vm-xubuntu:/projects/ECUcore-9G20/user$ cd shpingdemo
vmware@vm-xubuntu:/projects/ECUcore-9G20/user/shpingdemo$ make

Make Settings
  CFLAGS = '-O0 -g -Wall -march=armv5te -mcpu=arm926ej-s -mtune=arm9tdmi -msoft-
t-float -I. -DNDEBUG'
  LDFLAGS = '-march=armv5te -mcpu=arm926ej-s -mtune=arm9tdmi -msoft-float'
  LDLIBS = ''

Compiling 'shpingdemo.c'...
Compiling 'shpingclient.c'...
Compiling 'shmclient.c'...
Compiling 'trmio.c'...
Compiling 'trace.c'...
Linking 'shpingdemo'...
Stripping 'shpingdemo'...
Done.

Copy executable 'shpingdemo' to destination '/tftpboot': done.

vmware@vm-xubuntu:/projects/ECUcore-9G20/user/shpingdemo$

```

Bild 28: Generierung des Demoprojektes "*shpingdemo*" im Linux-Entwicklungssystem

Die Anwendung und Bedienung des Demoprojektes "*shpingdemo*" auf dem PLCcore-9G20 beschreibt Abschnitt 8.1.4.

### 8.1.4 Beispiel zur Nutzung des Shared Prozessabbild

Als Beispiel zum Datenaustausch zwischen einem SPS-Programm und einer anwenderspezifischen C/C++ Applikation dient das im Abschnitt 8.1.3 beschriebene Demoprojekt "shpimgdemo" in Verbindung mit dem SPS-Programmbeispiel "RunLight".

#### Technischer Hintergrund

Für eine C/C++ Applikation sind alle Variablen über das Shared Prozessabbild nutzbar, die das SPS-Programm als direkt adressierte Variablen im Prozessabbild anlegt. Im SPS-Programmbeispiel "RunLight" sind dies folgende Variablen:

```
(* variables for local control via on-board I/O's *)
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 *)
uiRemoteSliderLen AT %MW512.0 : UINT;      (* out: length of sliderbar      *)
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                       *)
```

Um von einer C/C++ Applikation über das Shared Prozessabbild auf die Variablen des SPS-Programms zugreifen zu können, müssen zum Einen entsprechende Sektionen für die *ReadSectorTable* und die *WriteSectorTable* angelegt werden und zum anderen ist die Definition von Pointern für den Zugriff auf die Variablen notwendig. Der nachfolgende Programmausschnitt verdeutlicht dies am Beispiel von "shpimgdemo.c". Die Funktion *ShPIImgClntSetup()* trägt die Startoffsets von Input-, Output- und Merkerbereich in die übergebene Struktur *ShPIImgLayoutDscrpt* ein. Zusammen mit der von *ShPIImgClntGetDataSect()* gelieferten Anfangsadresse lassen sich so die absoluten Anfangsadressen der einzelnen Bereiche innerhalb des Shared Prozessabbildes bestimmen. Um die Adresse einer bestimmten Variablen zu ermitteln, ist noch ihr entsprechender Offset innerhalb der jeweiligen Sektion zu addieren. So ergibt sich beispielsweise die absolute Adresse für den Zugriff auf die Variable "*bRemoteDirCtrl AT %MB515.0 : BYTE;*" als Summe aus der Anfangsadresse des Shared Prozessabbildes (*pabShPIImgDataSect*), dem Startoffset des Merkerbereiches (*ShPIImgLayoutDscrpt.m\_uiPIImgMarkerOffs* für "%M...") sowie der im SPS-Programm definierten direkten Adresse innerhalb des Merkerbereiches (515 für "%MB515.0"):

```
pbPIImgVar_61131_bDirCtrl = (BYTE*) (pabShPIImgDataSect
    + ShPIImgLayoutDscrpt.m_uiPIImgMarkerOffs + 515);
```

Der nachfolgende Code-Ausschnitt zeigt die vollständige Definition aller im Demoprojekt verwendeten Variablen für den Datenaustausch mit dem SPS-Programm:

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

pabShPIImgDataSect = ShPIImgClntGetDataSect();
```

```

// ---- 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");
}

```

## Ausführung auf dem PLCcore-9G20

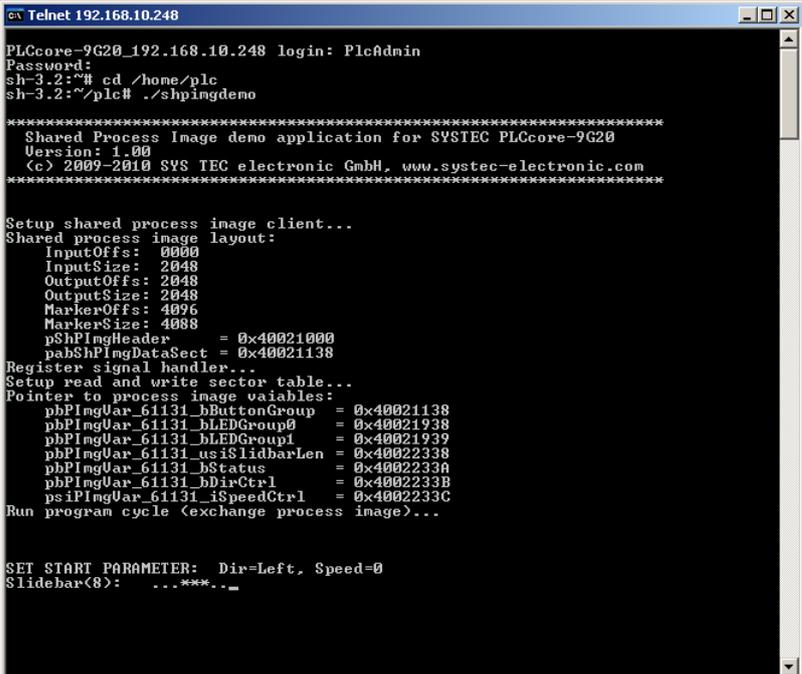
Um das *Shared Process Image Demo* auch ohne vorherige Einarbeitung in die Linux-basierte C/C++ Programmierung für das PLCcore-9G20 ausprobieren zu können, wurde eine bereits fertig übersetzte und lauffähige Version des Programms zusammen mit der SPS-Firmware auf dem Modul installiert (`/home/plc/shpimgdemo`). Die folgende Beschreibung bezieht sich auf diese Programmversion. Alternativ kann das Demoprojekt auch aus den entsprechenden Source-Files neu erstellt (siehe Abschnitt 8.1.3) und anschließend gestartet werden.

Zur Ausführung des *Shared Process Image Demos* auf dem PLCcore-9G20 sind folgende Schritte notwendig:

1. **Aktivieren der Option "Share PLC process image"** in der SPS-Konfiguration (siehe Abschnitte 8.1.1 bzw. 7.4.1 und 7.4.3).
2. Öffnen des SPS-Programmbeispiels *"RunLight"* im IEC 61131-Programmiersystem *OpenPCS* und Übersetzen des Projektes für eine Zielhardware vom Typ *"SYSTEC - PLCcore-9G20"*
3. Auswahl der Netzwerkverbindung zum PLCcore-9G20 und Download des Programms
4. Starten des SPS-Programms auf dem PLCcore-9G20
5. Anmeldung an der Kommando-Shell des PLCcore-9G20 wie in Abschnitt 7.8.1 beschrieben
6. Wechseln in das Verzeichnis `/home/plc` und Aufruf des Demoprogramms `"shpimgdemo"`:

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

Auf dem PLCcore-9G20 werden die digitalen Ausgänge als Lauflicht angesteuert. Die Geschwindigkeit ist über den Analogeingang A10 (Poti am ADC des Developmentboards) veränderbar. Mit Hilfe der Taster S400 (DIO) und S401 (DI1) kann die Laufrichtung umgeschaltet werden. Nach dem Start des Demoprogramms *"shpimgdemo"* auf dem PLCcore-9G20 werden im Terminal zyklisch die aktuellen Statusinformationen zum Lauflicht angezeigt (siehe Bild 29).



```

Telnet 192.168.10.248
PLCcore-9G20_192.168.10.248 login: PlcAdmin
Password:
sh-3.2:~# cd /home/plc
sh-3.2:~/plc# ./shpimgdemo
*****
Shared Process Image demo application for SYSTEC PLCcore-9G20
Version: 1.00
(c) 2009-2010 SYS TEC electronic GmbH, www.systec-electronic.com
*****

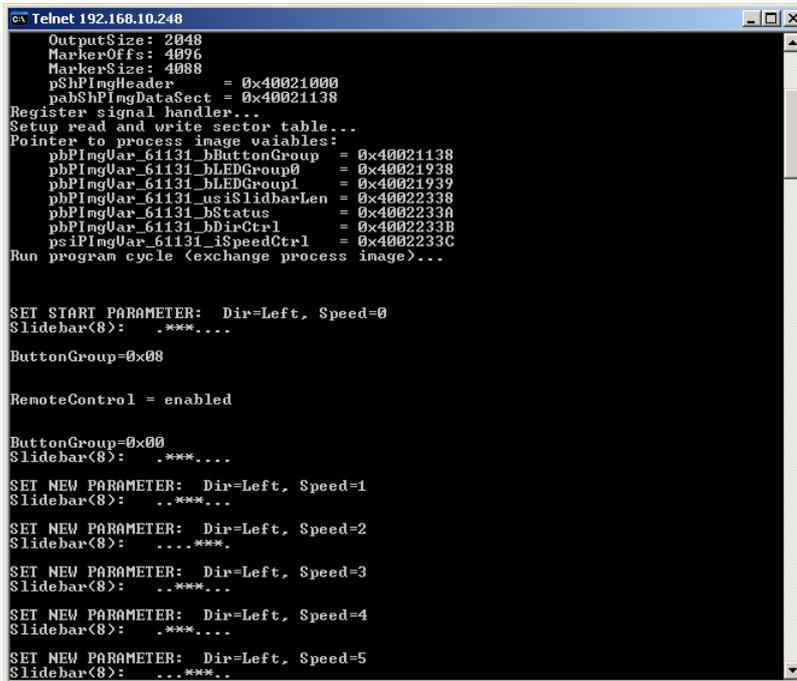
Setup shared process image client...
Shared process image layout:
  InputOfs: 0000
  InputSize: 2048
  OutputOfs: 2048
  OutputSize: 2048
  MarkerOfs: 4096
  MarkerSize: 4088
  pShPingHeader = 0x40021000
  pabShPingDataSect = 0x40021138
Register signal handler...
Setup read and write sector table...
Pointer to process image variables:
  pbPingVar_61131_bButtonGroup = 0x40021138
  pbPingVar_61131_bLEDGroup0 = 0x40021938
  pbPingVar_61131_bLEDGroup1 = 0x40021939
  pbPingVar_61131_usiSliderLen = 0x40022338
  pbPingVar_61131_bStatus = 0x4002233A
  pbPingVar_61131_bDirCtrl = 0x4002233B
  psiPingVar_61131_iSpeedCtrl = 0x4002233C
Run program cycle (exchange process image)...

SET START PARAMETER: Dir=Left, Speed=0
Slider(8):  ..***..

```

Bild 29: Terminalausgaben des Demoprogramms *"shpimgdemo"* nach dem Start

7. Nach Drücken des Tasters S403 (DI3) wird die Richtungs- und Geschwindigkeitskontrolle des Lauflichtes an das Demoprogramm "shpimgdemo" abgegeben. Im Terminalfenster kann nun mit den Cursor-Tasten links und rechts (← und →) die Laufrichtung sowie mit den Cursor-Tasten auf und ab (↑ und ↓) die Geschwindigkeit des Lauflichtes durch die C-Applikation festgelegt werden.



```

Telnet 192.168.10.248
OutputSize: 2048
MarkerOfs: 4096
MarkerSize: 4088
pShPingHeader = 0x40021000
pShPingDataSect = 0x40021138
Register signal handler...
Setup read and write sector table...
Pointer to process image variables:
pbPingVar_61131_bButtonGroup = 0x40021138
pbPingVar_61131_bLEDGroup0 = 0x40021938
pbPingVar_61131_bLEDGroup1 = 0x40021939
pbPingVar_61131_usiSliderLen = 0x40022338
pbPingVar_61131_bStatus = 0x4002233A
pbPingVar_61131_bDirCtrl = 0x4002233B
psIPingVar_61131_iSpeedCtrl = 0x4002233C
Run program cycle (exchange process image)...

SET START PARAMETER: Dir=Left, Speed=0
Slider(8): .***.

ButtonGroup=0x08

RemoteControl = enabled

ButtonGroup=0x08
Slider(8): .***.

SET NEW PARAMETER: Dir=Left, Speed=1
Slider(8): .***.

SET NEW PARAMETER: Dir=Left, Speed=2
Slider(8): .***.

SET NEW PARAMETER: Dir=Left, Speed=3
Slider(8): .***.

SET NEW PARAMETER: Dir=Left, Speed=4
Slider(8): .***.

SET NEW PARAMETER: Dir=Left, Speed=5
Slider(8): .***.

```

Bild 30: Terminalausgaben des Demoprogramms "shpimgdemo" nach Benutzereingaben

Bild 30 zeigt die Terminalausgaben des Demoprogramms "shpimgdemo" als Reaktion auf die Betätigung der Cursor-Tasten.

Das Demoprogramm "shpimgdemo" lässt sich durch Drücken von "Ctrl+C" im Terminalfenster beenden.

## 8.2 Driver Development Kit (DDK) für das PLCcore-9G20

Das Driver Development Kit (DDK) für das ECUcore-9G20 (bzw. PLCcore-9G20) wird als zusätzliches Softwarepaket mit der Artikelnummer SO-1106 vertrieben. Es ist nicht im Lieferumfang des PLCcore-9G20 bzw. dem Development Kit PLCcore-9G20 enthalten. Details zum DDK beschreibt das "Software Manual Driver Development Kit für ECUcore-9G20" (Manual-Nr.: L-1257).

Das Driver Development Kit für das ECUcore-9G20 (bzw. PLCcore-9G20) ermöglicht dem Anwender die eigenständige Anpassung der I/O-Ebene an ein selbst entwickeltes Baseboard. Das auf dem PLCcore-9G20 eingesetzte Embedded Linux unterstützt das dynamische Laden von Treibern zur Laufzeit und ermöglicht so die Trennung von SPS-Laufzeitsystem und I/O-Treiber. Damit ist der Anwender in der Lage, den I/O-Treiber komplett an die eigenen Bedürfnisse anzupassen, ohne dazu das SPS-Laufzeitsystem selbst verändern zu müssen.

Mit Hilfe des DDK können folgende Ressourcen in die I/O-Ebene einbezogen werden:

- Peripherie (in der Regel GPIO) des AT91SAM9G20
- on-board FPGA (Lattice ECP2-6)
- Adress-/Datenbus (memory-mapped Peripherie)
- SPI-Bus und I<sup>2</sup>C-Bus

- alle anderen vom Betriebssystem bereitgestellten Ressourcen wie z.B. Filesystem und TCP/IP

Bild 31 vermittelt einen Überblick über die DDK-Struktur und die enthaltenen Komponenten. Im DDK enthalten sind die Sourcen der FPGA-Software (VHDL), des Linux Kernel-Treibers (*pc9g20drv.ko*) und der Linux User-Bibliothek (*pc9g20drv.so*). Ebenfalls Bestandteil des DDK ist ein PLD/FPGA Programming Tool (*pldtool* + *plddrv.ko*), das ein Softwareupdate des FPGA unter Linux ohne zusätzliche JTAG-Hardware ermöglicht.

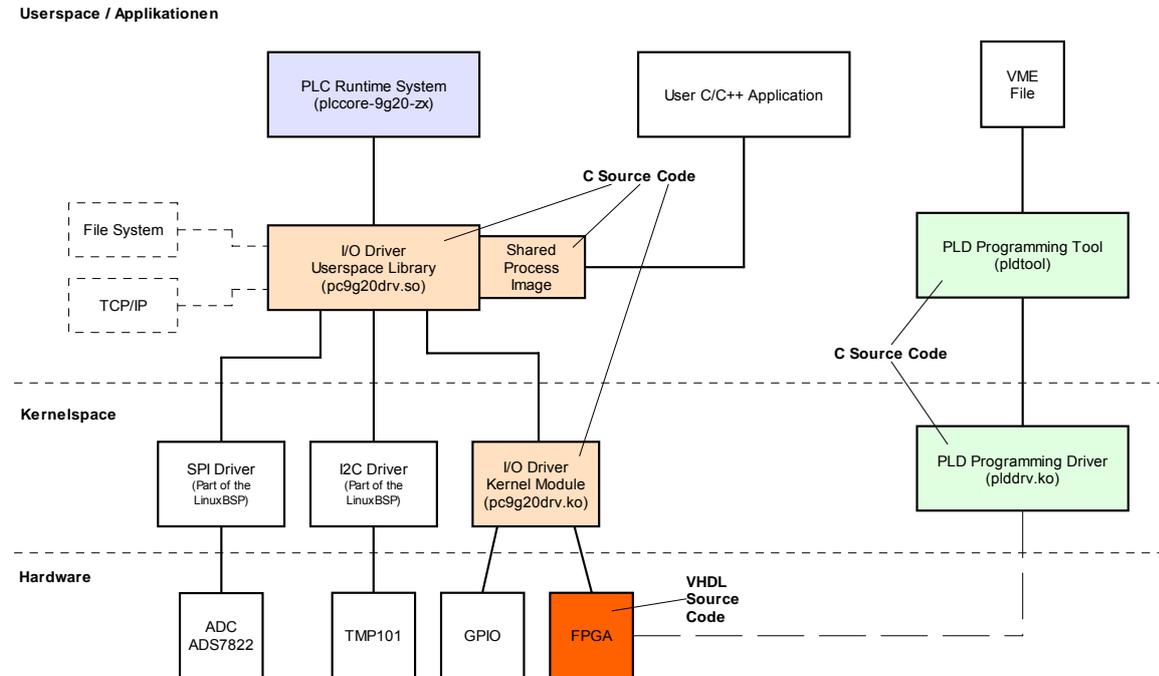


Bild 31: Übersicht zum Driver Development Kit für das PLCcore-9G20

### Lieferumfang / Komponenten des DDK:

Das DDK beinhaltet folgende Komponenten:

1. VHDL-Projekt für FPGA, beinhaltet alle notwendigen Files um FPGA-Software neu zu erstellen (VHLS-Sourcefiles, Pin-Zuordnung, Timingeinstellungen, Projektfile usw.)
2. Sourcecode für Linux Kernel-Treiber (*pc9g20drv.ko*, siehe Bild 31), beinhaltet alle notwendigen Files um Kernel-Treiber neu zu erstellen (C- und H-Files, Makefile usw.)
3. Sourcecode für Linux User-Bibliothek (*pc9g20drv.so*, siehe Bild 31), beinhaltet alle notwendigen Files um User-Bibliothek (incl. der Implementierung des Shared Process Image) neu zu erstellen (C- und H-Files, Makefile usw.)
4. PLD/FPGA Programming Tool (*pldtool* + *plddrv.ko*), ermöglicht das Softwareupdate des FPGA unter Linux ohne zusätzliche JTAG-Hardware
5. I/O-Treiber Demoapplikation (*iodrvdemo*) im Sourcecode, ermöglicht den schnellen und einfachen Test des I/O-Treibers
6. Dokumentation

Das Driver Development Kit setzt das Softwarepaket **SO-1105** ("VMware-Image des Linux-Entwicklungssystems") voraus, das sowohl die Quellen des verwendeten LinuxBSP als auch die erforderliche GNU-Crosscompiler Toolchain für ARM9-Prozessoren enthält.

### 8.3 Testen der Hardwareanschaltung

Das PLCcore-9G204 ist primär als Zulieferteil für den Einbau in industriellen Steuerungen bestimmt. Dazu wird das PLCcore-9G20 typischerweise auf einer anwenderspezifischen Basisplatte betrieben. Um hier eine einfache Kontrolle der korrekten I/O-Anschaltung zu ermöglichen, wird das Testprogramm "*iodrvdemo*" zusammen mit der SPS-Firmware auf dem Modul installiert. Dieses Testprogramm setzt direkt auf dem I/O-Treiber auf und ermöglicht so den unmittelbaren Peripheriezugriff.

Damit das Testprogramm "*iodrvdemo*" exklusiven Zugriff auf alle I/O-Ressourcen erhält, muss ein evtl. laufendes SPS-Laufzeitsystem zunächst beendet werden. Dies ist am einfachsten durch den Aufruf des Skriptes "*stopplc*" möglich:

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

Anschließend kann der I/O-Treiber erneut geladen und das Testprogramm "*iodrvdemo*" gestartet werden:

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

Bild 32 veranschaulicht das Testen der Hardwareanschaltung mit "*iodrvdemo*".

```

c:\ Telnet 192.168.10.248

PLCcore-9G20_192.168.10.248 login: PlcAdmin
Password:
sh-3.2:~# cd /home/plc
sh-3.2:~/plc# ./stopplc
Killing PLC runtime system... done.
Unload I/O driver... done.
Unload CAN driver... done.
sh-3.2:~/plc# insmod pc9g20drv.ko
sh-3.2:~/plc# ./iodrvdemo

*****
Test application for SYSTEC PLCcore-9G20 board driver
Version: 1.00
(c) 2009-2010 SYS TEC electronic GmbH, www.systec-electronic.com
*****

I/O Driver version: KernelModule=1.00, UserLib=1.00

Hardware: CPU Board: 4258.00 (&#01H)
          CPU FPGA: 1.04 (&#06H)
          I/O Board: 4261.02 (&#02H)

IO config: Digital In: 19
           Digital Out: 8
           Analog In: 3
           Analog Out: 0
           Counter: 4
           PWM/PTO: 4
           TempSensor: 1
Driver: Config: 0000H

FPGA interrupt selftest: successful

Please Select:
0 - Exit this application
1 - Run Basic I/O test (digital I/O and user switches)
2 - Run Counter test
3 - Run PWM test (pre-configured demo)
4 - Run PWM test (manual parameter input)
5 - Run PTO test (pre-configured demo)
6 - Run PTO test (manual parameter input)
7 - Run ADC test
8 - Run EEPROM test
T - Run Temperature Sensor test
P - Run Process Image test
Select: 1

=== Basic I/O Test ===

Start basic I/O main loop... (press ESC to abort)

DI=0xFA-0x00-0x00 D0=0x00-0x00-0x01 bHexSwitch=0x20 bDipSwitch=0x00 R/S/M-Switch = RUN
DI=0xFA-0x00-0x00 D0=0x00-0x00-0x02 bHexSwitch=0x20 bDipSwitch=0x00 R/S/M-Switch = RUN
DI=0xFA-0x00-0x00 D0=0x00-0x00-0x04 bHexSwitch=0x20 bDipSwitch=0x00 R/S/M-Switch = RUN
DI=0xFA-0x00-0x00 D0=0x00-0x00-0x08 bHexSwitch=0x20 bDipSwitch=0x00 R/S/M-Switch = RUN
=

```

Bild 32: Testen der Hardwareanschaltung mit "iodrvdemo"

## Anhang A: Firmware-Funktionsumfang des PLCcore-9G20

Tabelle 22 listet die auf dem PLCcore-9G20 verfügbaren Firmware-Funktionen und -Funktionsbausteine auf.

### Zeichenerklärung:

FB	Funktionsbaustein
FUN	Funktion
Online Help	<i>OpenPCS</i> Online-Hilfe
L-1054	Manual " <i>SYS TEC spezifische Erweiterungen für OpenPCS / IEC 61131-3</i> ", Manual-Nr.: L-1054)
PARAM:={0,1,2}	für den angegebenen Parameter sind die Werte 0,1 und 2 zulässig

Tabelle 22: Firmware-Funktionen und -Funktionsbausteine des PLCcore-9G20

Name	Type	Reference	Remark
<b>PLC standard Functions and Function Blocks</b>			
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	
<b>Functions and Function Blocks for string manipulation</b>			
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	

<b>Functions and Function Blocks for OpenPCS specific task controlling</b>			
ETRC	FB	L-1054	
PTRC	FB	L-1054	
GETVARDATA	FB	Online Help	
GETVARFLATADDRESS	FB	Online Help	
GETTASKINFO	FB	Online Help	
<b>Functions and Function Blocks for handling of non-volatile data</b>			
NVDATA_BIT	FB	L-1054	DEVICE:={0}, see <sup>(1)</sup>
NVDATA_INT	FB	L-1054	DEVICE:={0}, see <sup>(1)</sup>
NVDATA_STR	FB	L-1054	DEVICE:={0}, see <sup>(1)</sup>
<b>Functions and Function Blocks for handling of time</b>			
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	
<b>Functions and Function Blocks for counter inputs and pulse outputs</b>			
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}
<b>Functions and Function Blocks for Serial interfaces</b>			
SIO_INIT	FB	L-1054	PORT:={0,1,2,3}, see <sup>(2)</sup>
SIO_STATE	FB	L-1054	PORT:={0,1,2,3} see <sup>(2)</sup>
SIO_READ_CHR	FB	L-1054	PORT:={0,1,2,3} see <sup>(2)</sup>
SIO_WRITE_CHR	FB	L-1054	PORT:={0,1,2,3} see <sup>(2)</sup>
SIO_READ_STR	FB	L-1054	PORT:={0,1,2,3} see <sup>(2)</sup>
SIO_WRITE_STR	FB	L-1054	PORT:={0,1,2,3} see <sup>(2)</sup>
<b>Functions and Function Blocks for CAN interfaces / CANopen</b>			
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}

<b>Functions and Function Blocks for Ethernet interfaces / UDP</b>			
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) Auf dem PLCcore-9G20 erfolgt die Ablage der nicht-flüchtigen Daten in der Datei *"/home/plc/PlcPData.bin"*. Diese Datei hat eine fixe Größe von 32 kByte. Beim Aufruf der Funktionsbausteine vom Typ *NVDATA\_Xxx* in einem Schreib-Modus werden die modifizierten Daten unmittelbar in die Datei *"/home/plc/PlcPData.bin"* geschrieben (*"flush"*). Dadurch gehen bei einer Spannungsunterbrechung keine ungesicherten Daten verloren.
  
- (2) Die Schnittstelle COM0 (PORT:=0) dient primär als Serviceschnittstelle zur Administration des PLCcore-9G20. Daher sollten über diese Schnittstelle nur Zeichen ausgegeben werden. Empfangene Zeichen versucht das Modul stets als Linux-Kommandos zu interpretieren und auszuführen (siehe Abschnitt 6.5.1).

# Anhang B: Referenzdesigns zum PLCcore-9G20

## I/O examples for PLCcore-9G20

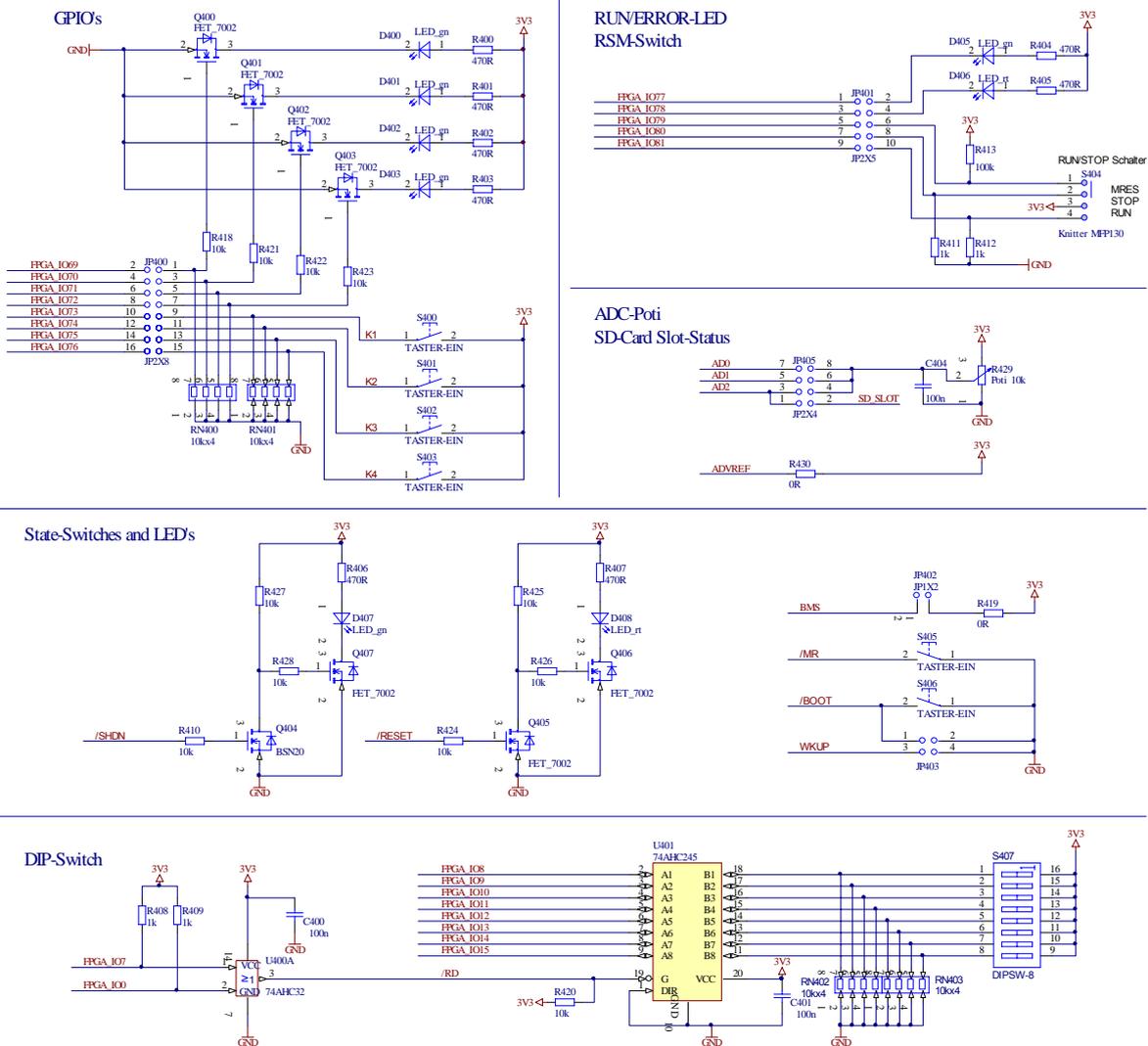
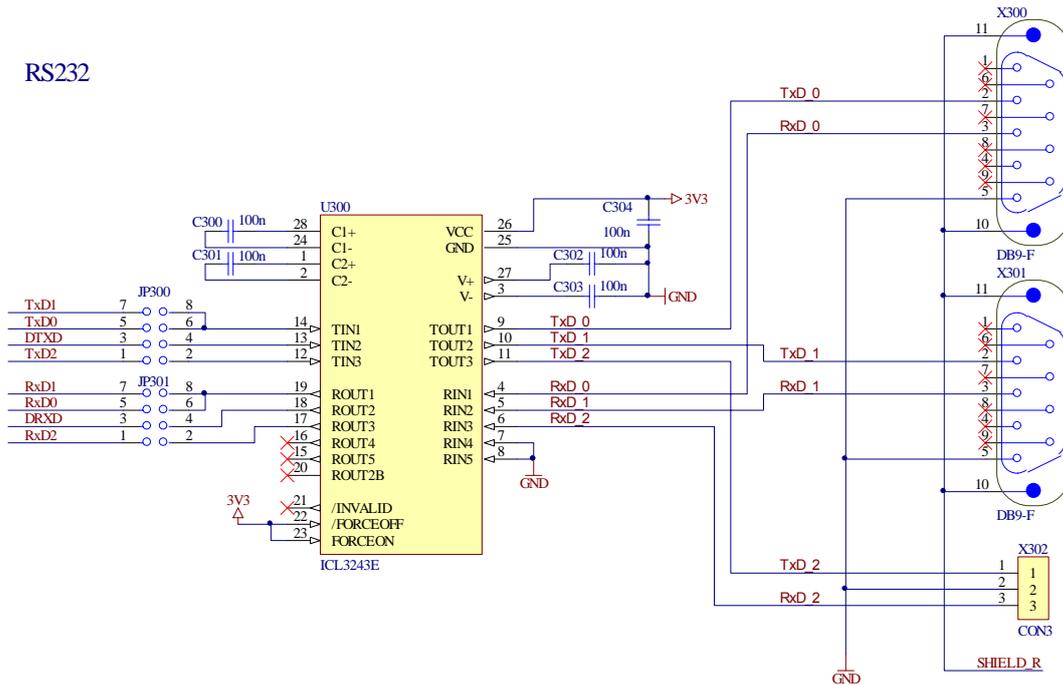


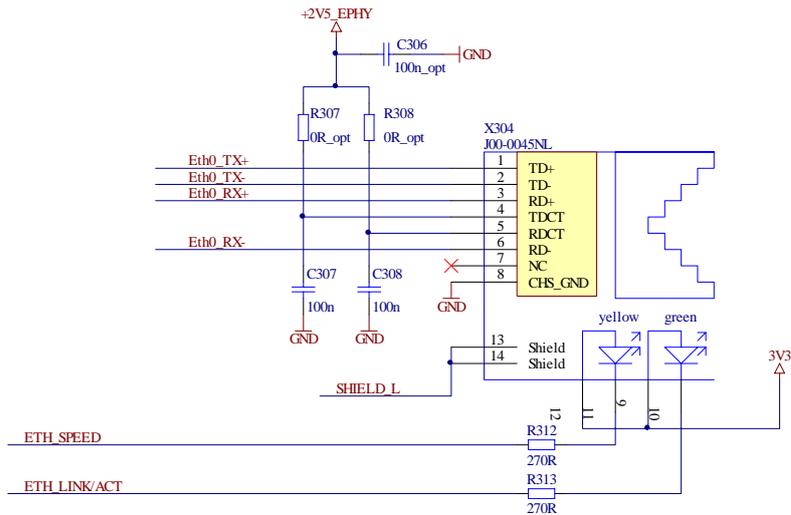
Bild 33: Referenzdesign zur I/O-Anschaltung

# interface examples for PLCcore-9G20

## RS232



## Ethernet



## CAN

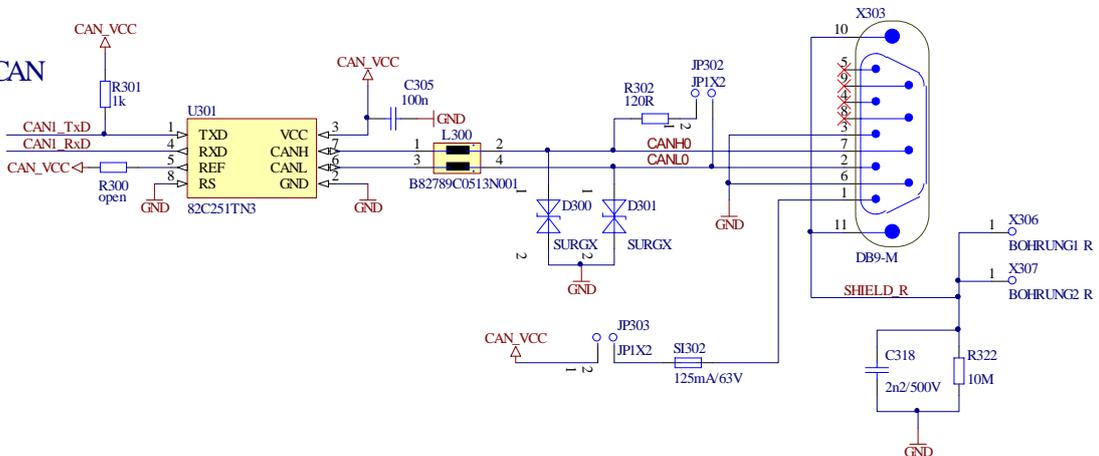


Bild 34: Referenzdesign zur Interface-Beschaltung

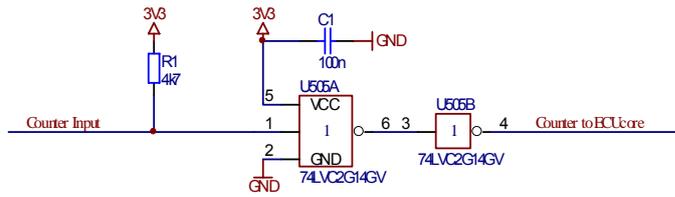


Bild 35: Referenzdesign zur Anschaltung der Zählereingänge

## Anhang C: GNU GENERAL PUBLIC LICENSE

Das auf dem PLCcore-9G20 eingesetzte Embedded Linux ist unter der GNU General Public License, Version 2 lizenziert. Der vollständige Text dieser Lizenz ist nachfolgend aufgeführt. Eine deutsche Übersetzung ist unter <http://www.gnu.de/documents/gpl-2.0.de.html> zu finden. Es handelt sich jedoch dabei nicht um eine offizielle oder im rechtlichen Sinne anerkannte Übersetzung.

Das verwendete SPS-System sowie die vom Anwender entwickelten SPS- und C/C++ Programme unterliegen **nicht** der GNU General Public License!

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

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

#### **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

/	
/home	45
/home/etc/autostart	17, 37
/home/plc/plccore-9g20.cfg	34
/home/plc/PlcPData.bin	70
/tmp	45, 47
<b>A</b>	
Abmessungen	8
adduser	43
Administration	
Systemvoraussetzungen	28
Anschlussbelegung	14
autostart	17, 37
AWL	9
<b>B</b>	
Bedienelemente	
Error-LED	24
Run/Stop-Schalter	23
Run-LED	23
Bitrate	36
Boot-Bedingungen	29
Bootkonfiguration	37
<b>C</b>	
CAN0	11, 21, 26
CANopen	9, 25
CANopen Master	9
CANopen-Chip	9
CE-Konformität	5
CFG-Datei	36
COM	21
COM0	11, 21
COM1	11, 21
COM2	11, 21
COM3	21
COM4	21
<b>D</b>	
date	44
Dateisystem	45
deluser	43
Development Kit	10
Developmentboard	
Anschlüsse	11
Bedienelemente	12
df46	
DIP-Schalter 1, Lage und Bedeutung	30
Driver Development Kit	13, 64
<b>E</b>	
Embedded Linux	9
EMV-Gesetz	5
Entwicklungskit	10
Error-LED	24
ETH0	11, 21
SPS-Programmbeispiel	21
<b>F</b>	
Firmware-Variante auswählen	38
FTP	
Anmeldung am PLCcore-9G20	41
FTP-Client	28
FUB	9
<b>G</b>	
GNU	9
GPL	74
<b>H</b>	
Hardwareanschaltung testen	66
hwclock	44
<b>I</b>	
iodrvdemo	66
<b>J</b>	
JFFS2	45
<b>K</b>	
Knotenadresse	36
Kommunikations-FB	18
Kommunikationsschnittstellen	
CAN	21
COM	21
ETH	21
Konfiguration	
Kommandos	32
SPS	33
Konfigurationsmodus	29
KOP	9
<b>L</b>	
Linux	9
Löschen SPS-Programm	24
<b>M</b>	
Manuals	
Übersicht	6
Master-Modus	36
<b>N</b>	
Nutzerkonten	
Anlegen und Löschen	43
Passwort ändern	43
vordefinierte	39
<b>O</b>	
OpenPCS	9

**P**

passwd.....	44
Pinout.....	14
plccore-9g20.cfg .....	34, 36, 48
PlcPData.bin .....	70
Programmierung.....	18
Prozessabbild	
Aufbau und Adressierung .....	19
Pulsausgänge.....	22

**R**

ReadSectorTable.....	52
Referenzdesign .....	71
root.sum.jffs2 .....	49
RTC setzen.....	44
Run/Stop-Schalter .....	23
Codierung .....	16
Löschen SPS-Programm .....	24
Run-LED .....	23

**S**

Shared Prozessabbild	
Aktivierung .....	52
API-Beschreibung .....	55
Beispiel.....	61
Signalisierung .....	55
Übersicht.....	51
Variablen-Paar .....	53
<i>ShPImgCIntGetDataSect</i> .....	57
<i>ShPImgCIntGetHeader</i> .....	57
<i>ShPImgCIntLockSegment</i> .....	57
<i>ShPImgCIntRelease</i> .....	56
<i>ShPImgCIntSetNewDataSigHandler</i> .....	56
<i>ShPImgCIntSetup</i> .....	56
<i>ShPImgCIntSetupReadSectTable</i> .....	57
<i>ShPImgCIntSetupWriteSectTable</i> .....	58
<i>ShPImgCIntUnlockSegment</i> .....	57
<i>ShPImgCIntWriteSectMarkNewData</i> .....	58
<i>shpimgdemo</i> .....	59
<i>shpimgdemo.tar.gz</i> .....	59
SO-1105 .....	59
SO-1106 .....	64

Softwareupdate

Linux-Image.....	49
SPS-Firmware .....	46
SPS-Programmbeispiel	
ETH0 .....	21
ST.....	9
stopplc.....	66
Systemstart.....	17
Systemzeit setzen.....	44

**T**

Telnet	
Anmeldung am PLCcore-9G20 .....	40
Telnet-Client.....	28
Terminaleinstellungen.....	31
Terminalprogramm.....	28
TFTPD32 .....	49
<i>tShPImgLayoutDscrpt</i> .....	55
<i>tShPImgSectDscrpt</i> .....	55

**U**

U-Boot Kommandoprompt	
Aktivierung.....	29
Terminaleinstellungen .....	31
U-Boot Kommandos	
BoardID Konfiguration.....	38
Ethernet Konfiguration .....	32
Update Linux-Image.....	49
<i>UdpRemoteCtrl</i> .....	21
USB-RS232 Adapter Kabel .....	13

**V**

vordefinierte Nutzerkonten.....	39
---------------------------------	----

**W**

WEB-Frontend .....	33
WinSCP .....	41
WriteSectorTable .....	52

**Z**

Zählereingänge.....	22
Zubehör.....	13



**Dokument:** System Manual PLCcore-9G20  
**Dokumentnummer:** L-1254d\_1, 1. Auflage Juli 2010

---

**Wie würden Sie dieses Handbuch verbessern?**

---

---

---

---

**Haben Sie in diesem Handbuch Fehler entdeckt?**

Seite

---

---

---

---

**Eingesandt von:**

Kundennummer: \_\_\_\_\_

Name: \_\_\_\_\_

Firma: \_\_\_\_\_

Adresse: \_\_\_\_\_

---

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

