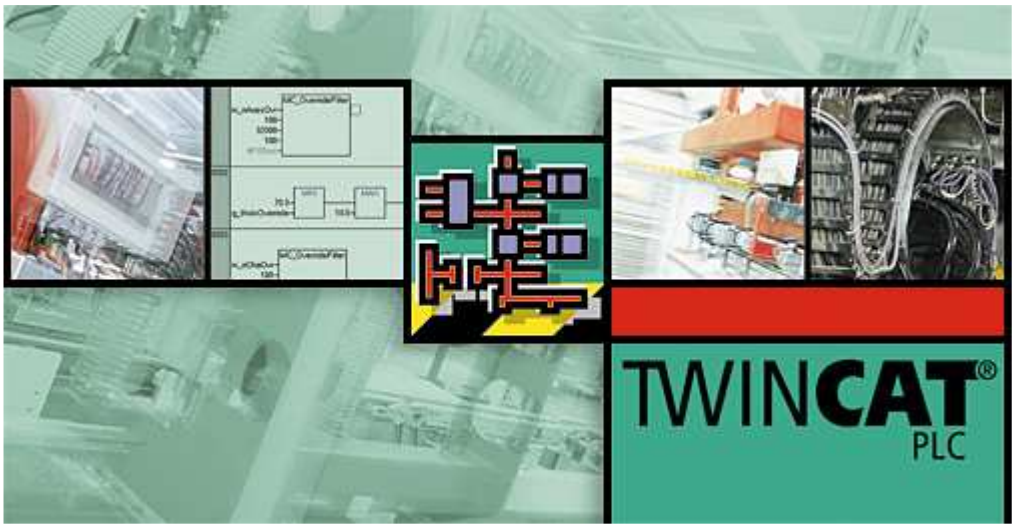
I2C – Library für I2C - Komponenten



I²C-Bus

Horter & Kalb

Inhaltsverzeichnis

1.1. Anforderungsspezifikationen

Die zu erstellende Library muss sämtliche Funktionen zur Verfügung stellen, um die I2C-Eingabekarten einzulesen und die I2C-Ausgabekarten per I2C-Modem steuern zu können. Die Funktionen beschränken sich hauptsächlich für die Komponenten aus dem Produktessortiment von www.horter.de.

Es ist eine Dokumentation zu erstellen, die beschreibt, wie die I2C-Library, ohne Schwierigkeiten erfolgreich in ein TwinCat – System einzubinden ist.

Weiter ist ein einfaches Einsteigerprogramm zu programmiern, das alle Funktionen der I2C-Library nutzt, um einem Anwender den Einstieg zu erleichtern.

Priortät	Funktion	Bemerkung
1.	Anfragen ob Modem vorhanden	
1.	Setzen von einem digitalen 8 Bit Ausgangsport	
1.	Lesen von einem digitalen 8 Bit Eingangsport	
1.	Setzen eines bestimmten analogen Ausganges	
1.	Lesen eines bestimmten analogen Einganges	
1.	Anfragen vom Modemzustand	
2.	Beschreibung für Library-User	Hat 1. Priorität, weil Pflichtenheft dies fordert.
2.	Einsteigerprogramm	Hat 1. Priorität, weil Pflichtenheft dies fordert.
2.	Anfragen von Modemversion	

Wertigkeit der Priorität:
1. > Funktion ist für Library unerlässlich
2. > Funktion ist für Library nicht elementar, aber vervollständigt diese

1.2. Glossar

Bezeichnung	Erklärung	Literatur
TwinCat	Entwicklungsumgebung von Beckhoff	www.beckhoff.com
IO	I = Input (Eingang) O = Output (Ausgang)	
SPS (PLC)	Speicher programmierbare Steuerung	
IC	Integrated Circuit = Elektronischer Baustein	Datenblätter
PIC	Programmable Interface Controller	Datenblätter
SW	Software	
FB	Funktionsblock	
SDL	Serielle Datenleitung	Horter und Kalb
SCL	Serielle Taktleitung	Horter und Kalb
INT	Interrupt (Infosignalleitung)	Horter und Kalb
I2C	Inter-Integrated Circuit	Internet
Library	Bibliothek	
Modem	Datenwandler	Internet
bidirektional	Datenübertragung in beide Richtungen	Internet
Multiplexverfahren	Mehrere Signale bündeln	Internet
Soft - PLC	PC basierte realtime SPS	Internet
Beckhoff Information system	Hilfe System von Beckhoff	www.beckhoff.com
Topologie	Aufbau der Verbindungen	Internet
ARRAY	Indexierbares Datenfeld	Internet

1.3. Grundlagenerarbeitung des I2C-Bus

Auszug von Wikipedia:
I2C (für Inter-Integrated Circuit, gesprochen I-Quadrat-C, I-square-C bzw. fälschlicherweise I-Two-C) ist ein Philips Semiconductors entwickelter serieller Datenbus. Er wird benutzt, um Geräte mit geringer Übertragungsgeschwindigkeit an ein eingebettetes System oder eine Hauptplatine anzuschliessen. Das ursprüngliche System wurde in den frühen 1980er Jahren entwickelt, um verschiedene Chips von Philips in Fernsehgeräten einfach steuern zu können. Einige Hersteller verwenden die Bezeichnung TWI (Two-Wire Interface), da I2C ein eingetragenes Markenzeichen von Philips Semiconductors ist. Technisch sind beide Systeme identisch. Weiteres ist auf http://de.wikipedia.org/wiki/I2C nachzulesen.

Hier möchte ich kurz auf das Funktionsprinzip des I2C-Buses und die wichtigsten verwendeten Bauteile, die Horter und Kalb in den I2C-Karten verwendet, eingehen.

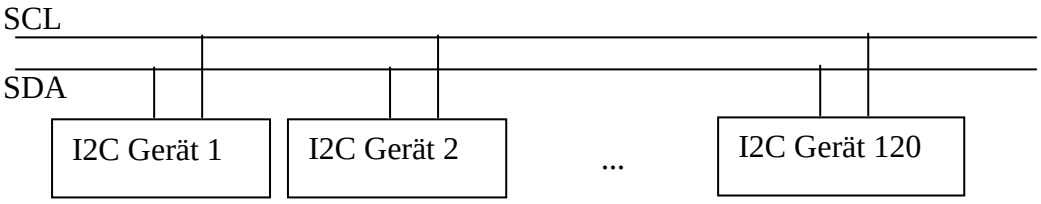


Bild1:

Funktionsprinzip des I2C-Bus (Auszug von Datenblatt):
Der I2C-Bus ist bidirektional und basiert auf einer 2-Drahtkommunikation.
Die zwei Drähte sind SDA (serielle Datenleitung) und SCL (serielle Taktleitung).
Eine Datenübertragung kann nur stattfinden, wenn der Bus nicht besetzt ist.

Bei jedem Taktpuls wird ein Bit übertragen. Die Daten auf der SDA-Leitung müssen während der High Periode des Taktsignals stabil bleiben. Ein Wechsel in dieser High Periode des Taktsignals wird als Steuersignal interpretiert.

Ist der Bus nicht besetzt, so sind die beiden Leitungen High. Ein Pegelwechsel von High nach Low an der Datenleitung, währenddem der Taktpegel High ist, signalisiert den Start. Ein Pegelwechsel von Low nach High an der Datenleitung, währenddem der Taktpegel High ist, signalisiert den Stop.

1.4. IC Beschreibung PCF 8574 (I/O-Karte)

(Auszug von Datenblatt)
Die digitalen Ein- und Ausgabekarten von Horter und Kalb benutzen den IC PCF 8574. Der PCF 8574 ist ein Baustein der für die I/O Erweiterung, via bidirektionale 2-Drahtleitung (I2C), von Microcontrollern verwendet wird.
Mit dem Baustein kann über die 2-Drahtleitung einen 8-Bit I/O-Port gesteuert und abgefragt werden. Die Adressierung der Bausteine wird über 3 Hardware-Pins erledigt, um bis zu acht (PCF 8574 bis 16) I2C-Geräte steuern zu können. Die Adressierung ist noch vom IC-Typ abhängig. Die a-Typen haben den Adressbereich 32-39 und die b-Typen 112-126.
Weitere Details über den PCF8574 können im Anhang entnommen werden.

1.5. IC Beschreibung PCF 8591 (Analog-Karte)

(Auszug von Datenblatt)
Die analogen Ein- und Ausgabekarte von Horter und Kalb benutzen den IC PCF 8591. Dieser IC hat vier analoge Eingänge, einen analogen Ausgang und eine I2C-Bus Schnittstelle. Wiederum drei Adresspins werden benutzt, um die Geräteadresse zu setzen. Es ist demzufolge möglich, acht Geräte an den gleichen Bus zu koppeln, ohne zusätzliche Hardware.
Die Adressierung und Steuerung der Daten, zum und vom Gerät, werden seriell via den 2-Draht I2C-Bus bidirektional übertragen.
Die vier Analogeingänge werden per Multiplexverfahren in den Baustein eingelesen und in einen 8 Bit Digitalwert umgewandelt. Der analoge Ausgang wird aus einem 8 Bit Digitalwert erzeugt. Die Abtastrate ist gegeben durch die Geschwindigkeit des I2C-Buses.
Weitere Details über den PCF8591 können im Anhang entnommen werden.

1.6. Vorabklärung, Analyse und Machbarkeit

Bei Horter und Kalb stehen zwei RS232 Geräte zur Verfügung, die für die Ankopplung an den I2C-Bus sind. Der I2C-RS232-Koppler und das I2C-Modem. Der Unterschied besteht darin, dass der I2C-RS232-Koppler die 5Volt Signale vom I2C-Bus auf V24 Signale für den PC umsetzt und umgekehrt. Es ist keinerlei Logik im I2C-RS232-Koppler vorhanden, er ist ein reiner Pegelwandler.

Das I2C-Modem ist da schon etwas komfortabler, es wandelt ebenfalls die Pegel und besitzt eine Logik, die in einem programmierbaren Baustein (PIC) sitzt. Diese Logik beinhaltet einen einfachen Befehlsatz. Der PIC interpretiert den empfangenen Befehl von dem RS232 Baustein und steuert oder liest nach seiner internen Logik die drei Leitungen SDA, SCL und INT.

Aus den oben genannten Erkenntnissen, entscheide ich mich für das **I2C-Modem**, da damit die Programmierung der Logik, für die Steuerung und Statusabfrage der drei Leitungen, eingespart werden kann.

Weiter habe ich mir Gedanken gemacht, wie ich mit der SoftPLC die Com-Schnittstelle, sprich RS232, vom PC (Notebook) ansprechen könnte. Ich habe also nach der Installation von TwinCat in der SystemInfo (Hilfeumgebung von Beckhoff) nachgeforscht, wie der Com-Port angesteuert werden könnte. Dabei bin ich auf die Serial Communication Library gestossen, mit der die PC Com-Schnittstelle anzusteuern ist.

Das aufzubauende System setzt sich soweit aus folgenden Hardwarekomponenten zusammen:

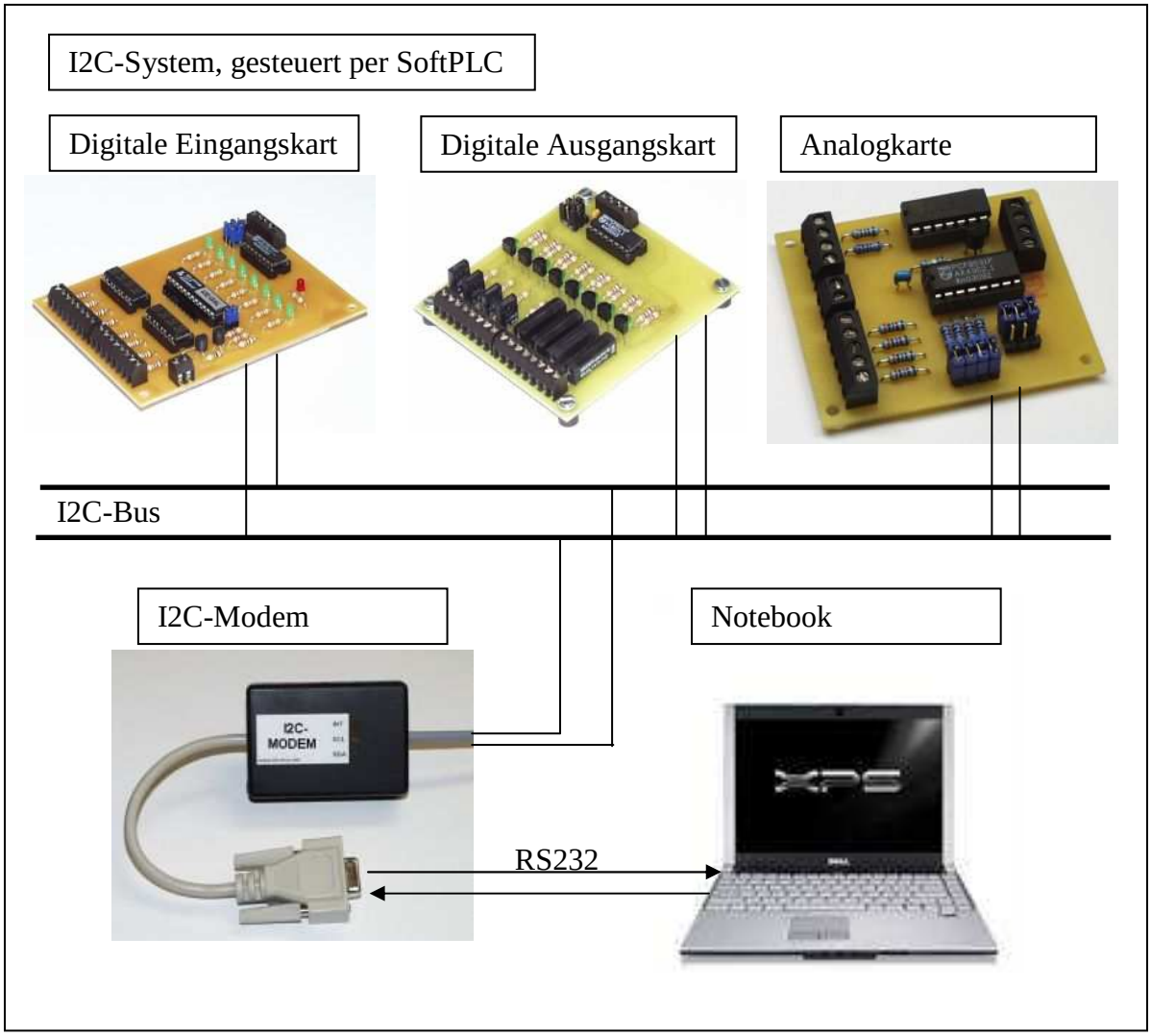


Bild 2:

Hardwarekomponenten:

- digitale Eingangskarte mit 8 Eingängen
- digitale Ausgangskarte mit 8 Ausgängen
- Analogkarte mit 4 analogen Eingängen und 1 analogen Ausgang
- I2C-Modem für die Kommunikation zwischen I2C-Bus und SoftPLC
- Notebook mit SoftPLC

Der Softwareteil des I2C-Systems setzt sich voraussichtlich, mit den Kenntnissen der bisherigen Analyse, aus folgenden Teilen zusammen:

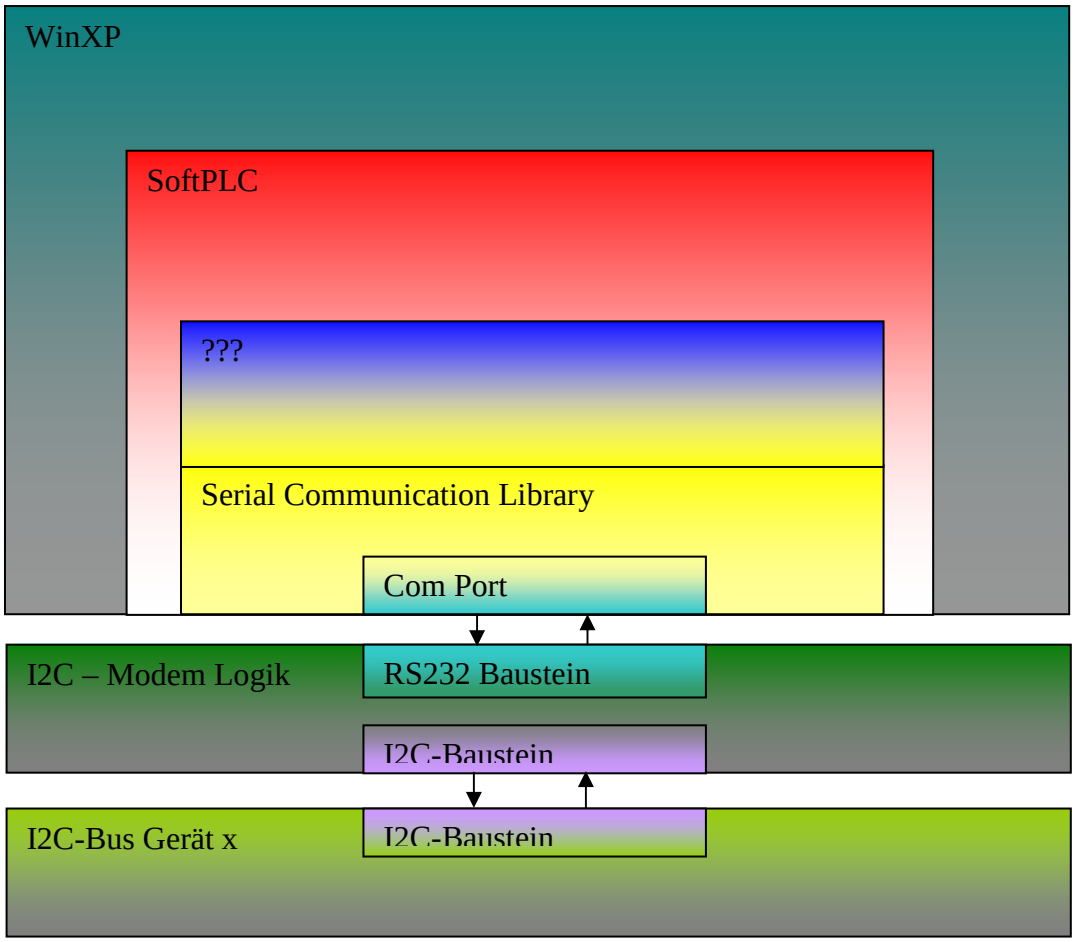


Bild 3:

Softwarekomponenten

- WinXP, das Betriebssystem, das auf dem Notebook oder PC läuft
- SoftPLC von Beckhoff, eine Software SPS die unter WinXP realtimefähig ist
- Serial Communikation Library von Beckhoff, die mit der SoftPLC die Anbindung an den Com-Port ermöglicht
- I2C-Modem Logik setzt die Anfragen, die über die RS232 Schnittstelle kommen, für den I2C-Bus
- I2C-Bus Geräte empfangen und senden die entsprechenden Anfragen und verarbeiten diese nach ihren spezifischen Eigenschaften. Z. Bsp. digitale Eingänge melden, digitale Ausgänge setzen, anlaoge Eingänge melden, analoge Ausgänge steuern usw.

- Ohne den ??? Softwareteil, ist aber das ganze System noch nicht zu gebrauchen. Es benötigt also noch eine SW-Applikation, die dem I2C-Modem die nötigen Anfragen für die I2C-Geräte übergibt. Diesen SW-Teil zu erstellen, ist nun meine Hauptaufgabe.

1.7. Analyse für die Konzepterstellung der I2C-Library

Nach dem Bild 3 ist hier noch der fehlende Softwareteil beschrieben, der mit ??? markiert wurde. Wie oben erwähnt, ist dieser SW-Teil die Hauptaufgabe. Dieser SW-Teil muss nun die geforderte I2C-Library und eine einfache Einsteigerapplikation beinhalten.

Der SW-Aufbau des I2C-Systems ohne Peripherie stellt sich dann so dar:

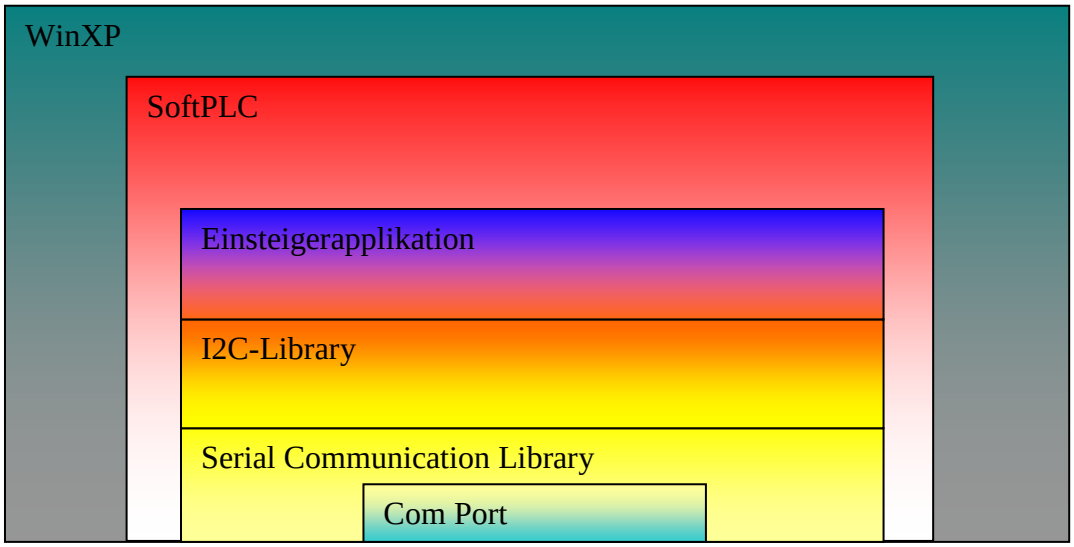


Bild 4:

1.7.1. Hardwaretest mit Excel-Applikation von Horter und Kalb

Nach dem Zusammenbau der elektronischen Bausätze von Horter und Kalb ist auch ein Test dieser Ein- und Ausgabenkarten nötig, um sicherzustellen, dass diese Komponenten auch funktionieren. Dies ist eine Voraussetzung, um dann auch die Entwicklung der I2C-Library angehen zu können. Denn die Softwareentwicklung ist immer ein Hin und Her zwischen programmieren und testen.

Für den Hardewaretest stellt Horter und Kalb eine Excel-Applikation “I2C-Modem-Test.xls” und eine Port.dll zur Verfügung. Die Port.dll ist in das Windows-Systemverzeichnis zu kopieren, damit das Excel-Makro läuft. C:\WINDOWS\system32\Port.dll

Excel-Applikation “I2C-Modem-Test.xls”

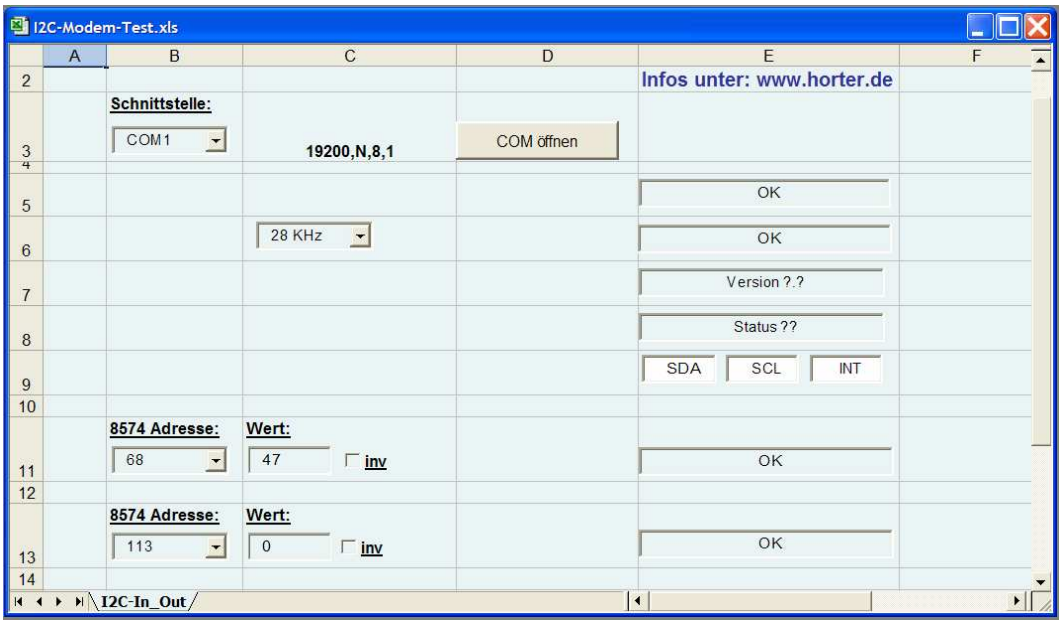


Bild 5:

Über diese Oberfläche kann das I2C-Modem, die Eingangskarte und Ausgangskarte getestet werden. Was sehr hilfreich war, denn es zeigte sich, dass die Ausgangskarte nicht funktionierte. Dies war aber nicht ein defekt der Ausgangskarte, wie sich im Nachhinein herausstellte. Es war meine schlechte Lötarbeit, die einen Kurzschluss verursachte, der die Speisespannung zusammenriss. Nachdem dieser Mangel aber behoben war, konnten alle Peripheriegeräte mit der Excel-Applikation per I2C-Modem gesteuert werden.

Die Voraussetzung von der Hardware her ist also gegeben, um den Softwareteil anzugehen.

1.7.2. Serial Communication Library Test

Eine weitere Foraussetzung die bestehen muss, um dann das Softwarekonzept der I2C-Library zu entwickeln, ist die Kenntnis über die Einbindung der Serial Communication Library und die Anbindung an die Hardware.

Im Beckhoff Information System wird beschrieben, wie die Serial Communication Library prinzipiell kommuniziert und wie die Verknüpfungen mit der Hardware zu erstellen sind. Weiter ist ein Beispielprogramm vorhanden, das ich als erstes versuche in Betrieb zunehmen, um erste Erfahrungen mit der Library machen zu können.

Die Integration der Library und die Verknüpfung zum Com-Port wird unter dem Punkt “Installation und Benutzeranleitung” im Detail beschrieben.

Die ComLib (Serial Communication Library) stellt unter anderen, auch die folgenden Funktionsblöcke zur Verfügung, welche ich verwenden werde.

Funktionsblöcke (FB):

- PcComControl
- ReceiveByte
- SendByte
- ClearComBuffer

Der Funktionsblock PcComControl kommuniziert zwischen der seriellen Schnittstelle (RS232) und der SPS. Dieser FB wird zyklisch aufgerufen in einem schnellen Kommunikationstask (2ms). Er platziert die empfangenen Daten im Receivebuffer (301 Byte) und gleichzeitig überträgt er die bereitstehenden Daten im Transmitbuffer (301Byte) an die RS232.

Der Funktionsblock (FB) ReceiveByte empfangt ein einzelnes Byte von der RS232 in der Ausgangsvariable RxBuffer. Über die Variable ByteReceived = TRUE wird erkannt, dass Bytes im Receivebuffer zum Auslesen bereit stehen. Die Anzahl der auszulesenden Bytes wird mit der Ausgangsvariable ReceivedByte mitgeteilt.

Der Funktionsblock (FB) SendByte sendet die Bytes, die in seiner Eingangsvariable TxBuffer abgelegt wurden. Solange der FB den Ausgang Busy = TRUE meldet, ist die Übermittlung noch nicht abgeschlossen. Ein Byte ist korrekt gesendet worden, wenn das Byte Busy=FALSE und der ERROR = 0 ist.

Die FB ReceiveByte und SendByte habe ich gewählt, weil das I2C-Modem Byte-Befehle versteht. Es kann maximal 16 Byte mit einer Anfrage verarbeiten.

I2C-Modem Byte-Befehle:

- VERSION
- IDENT
- SPEED
- STATUS
- READ
- WRITE

Der Befehl VERSION gibt die Firmware Version vom Modem zurück.

Der Befehl IDENT meldet, ob das Modem angeschlossen ist.

Mit dem Befehl SPEED kann die Übertragungsgeschwindigkeit auf dem I2C-Bus vorgegeben werden.

Der Befehl STATUS meldet den Zustand der Datenleitungen SCL, SDA und INT zurück.

Mit dem Befehl READ kann der Zustand von einem bestimmten I2C-Gerät gelesen werden.

Mit dem Befehl WRITE wird ein bestimmtes I2C-Gerät gesteuert.

1.8. Anwendungsfälle für den Benutzer der I2C-Library festhalten

Use Cases (Anwendungsfälle)

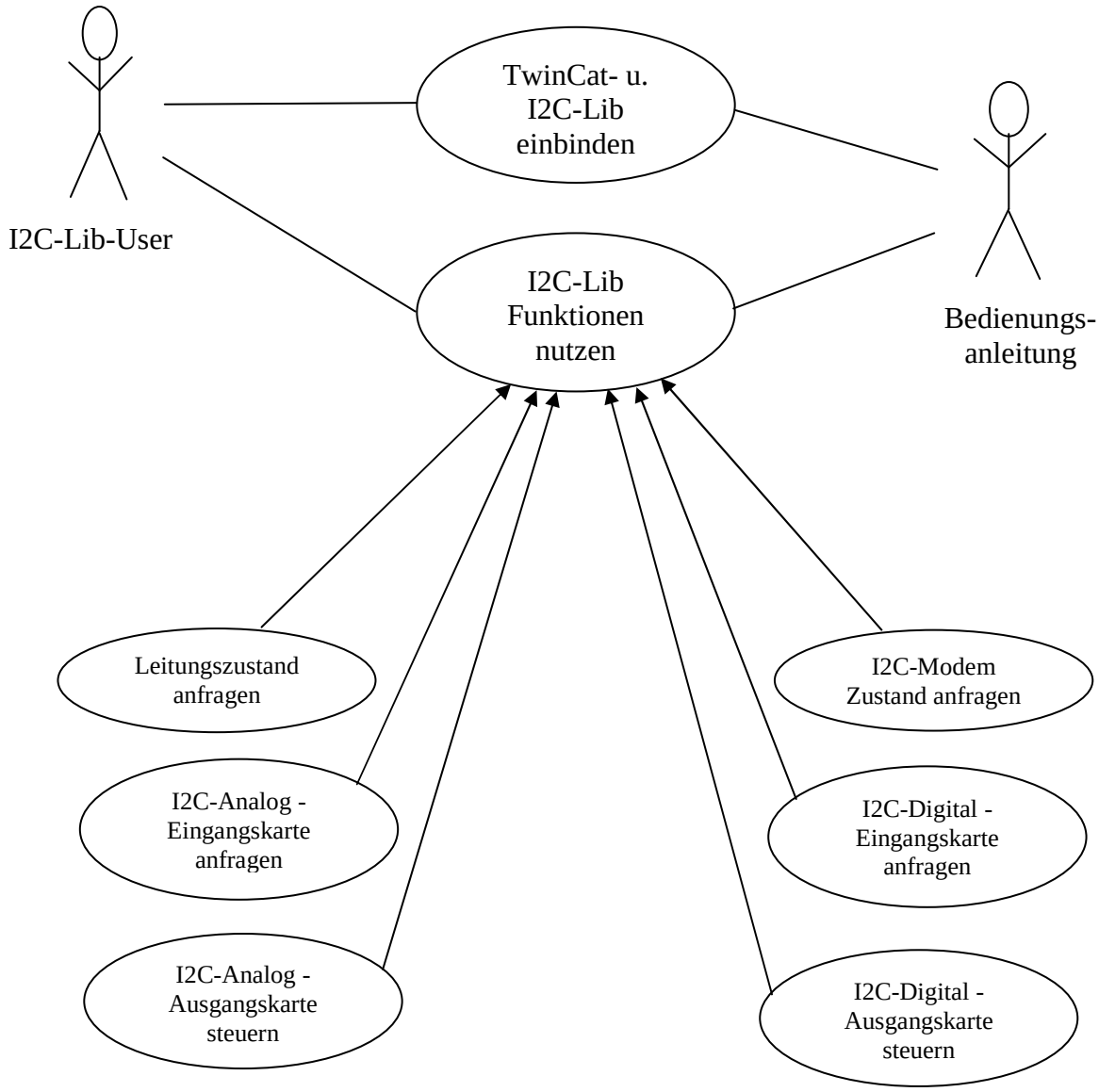


Bild 14:

Akteure:

I2C-Lib-User

Auslöser:

Der I2C-Lib-User möchte mit der TwinCat SoftPLC die I2C-Geräte, von Horter und Kalb, steuern.

Ablauf:

1. Mit Hilfe der Inbetriebnahmeanleitung, muss der I2C-Lib-User die nötigen TwinCat-Library's und die I2C-Library ins TwinCat Projekt einbinden können.

2. Weiter muss er anhand der Inbetriebnahmeanleitung, die Übersicht von den zur Verfügung stehenden Funktionen bekommen und jeweils ein geeignetes Anwendungsbeispiel entnehmen können.

Esentielle Use Cases:

- I2C-Modem Zustand anfragen
- Leitungszustand anfragen
- I2C-Digital - Eingangskarte anfragen
- I2C-Digital - Ausgangskarte steuern
- I2C-Analog - Eingangskarte anfragen
- I2C-Analog - Ausgangskarte steuern

Esentielle Use Cases sind Anwendungsfälle, die unbedingt vorhanden sein müssen, um die gestellte Aufgabe minimal erfüllen zu können.

2. Realisierung

2.1. Funktionsauflistung der I2C-Library

Programm / Funktionsblock / Struktur	***Eingangs-variablen***	***Ausgangs-variablen***	***Beschreibung***
FAST_PC_ComContol			Treiberbaustein aus der TwinCAT Library "ComLib.lib". Kommuniziert mit R232-Hardware.
	g_COMin		I/O Variabeln für den PC-COM Port
		g_COMout	
	g_RxBuffer		Sende Databuffer
		g_TxBuffer	Emfangs Databuffer
I2C_COM_Port			Koordiniert die Datenkommunikation
		o_bErrorTx	Meldet Sendefehler
		o_bErrorRx	Meldet Empfangsfehler
FB_GetIdent			Kontrolliert Modem Befehlsnummer: **16**
	i_boSend		Anfrage starten = TRUE
		o_boModemOn	Modem vorhanden = TRUE
FB_GetStatus			Status des I2C-Buses Befehlsnummer: **48**
	i_boSend		Anfrage starten = TRUE
		o_boSDA o_boSCL o_boINT	Meldet den Zustand der Signalleitungen, True oder False
FB_SetSpeed			I2C-Bus, Speed setzen Befehlsnummer: **32**
	i_boSend		Anfrage starten = TRUE
	i_bSpeed		Addieren zur Befehlsnummer: 0 => 43 KHz 1 => 28 KHz 2 => 17 KHz 3 => 9 KHz 4 => 5 KHz 5 => 2,5 KHz 6 => 1,3 KHz
		o_boOK	Anfrage erfolgreich

FB_GetVersion			Version von Modem

			Befehlsnummer: **80**
	i_boSend		Anfrage starten = TRUE
		o_strVersion	Version im String-Format
FB_WriteData			Ausgangskarten setzen Befehlsnummer: **64**
	i_boSend		Anfrage starten = TRUE
	i_bAnzByte		Anzahl der zu sendenden BYTE
	i_bSlaveAddress		Adresse der anzu-sprechenden Karte
	i_abDaten		Inhalt der zu schreibenden Daten (Datenarray 1-16)
		o_strErrorState	Fehlerinfo in String-Form
		o_boOK	Erfolgsbestätigung
FB_ReadData			Eingangskarten lesen Befehlsnummer: **128**
	i_boSend		Anfrage starten = TRUE
	i_bAnzByte		Anzahl der zu lesenden BYTE
	i_bSlaveAddress		Eingangskarten Adresse
		o_strErrorState	Fehlerinfo in String-Form
		o_boOK	Erfolgsbestätigung
		o_abDaten	Inhalt der empfangenen Daten (Datenarray 1-16)
g_aCommunication			In diesen Strukturarray 1-36, werden die Information der vorhandenen Ein- und Ausgabekarten abgelegt.
	i_bModemCmd		Definiert, welche Anfrage erfolgen muss.
	i_bAnzByte		Anzahl zu schreibende und sendende Bytes
	i_bSlaveAddress		Definiert, an welche Karte die Anfrage soll.
	i_abDaten		Trägt Daten zum Senden
		o_strErrorState	Fehlerinfo in String-Form
		o_boOK	Erfolgsbestätigung
		o_abDaten	Trägt empfangene Daten

		o_strVersion	Trägt Modemversion
		o_boSDA o_boSCL o_boINT	Trägt I2C-Bus Signalleitungs-information

fErrorTypToByte			Com-Fehler in Nummern wandeln: 0 = kein Fehler 1 = Eingangsparam-eter wechselte während des Empfangens 2 = String > Transmitbuffer 10 = Ende von String 11 = String kann keine Null characters empfangen 20 = ungültiger Datenzeiger 21 = ungültiger Datenzeiger für Empfangsdaten 22 = ungültige Länge für Empfangsdaten 23 = Ende von Datenblock
	i_TError		Fehlername
		fErrorTypToByte	Fehlernummer
I2C_COM_Port			Sende- und Empfangs-Funktionsblock
		o_bErrorTx	TxCom-Fehler in Nummern
		o_bErrorRx	RxCom-Fehler in Nummern
READ_WRITE _IO_CTRL			Programm koordiniert den Datenaustausch zwischen der SoftPLC von Beckhoff TwinCat und den I2C-Komponenten
	aT_Communication		Information der vorhandenen Ein- und Ausgabekarten
		aT_Communication	Rückgabewerte der vorhandenen Ein- und Ausgabekarten

2.2. Software Architektur

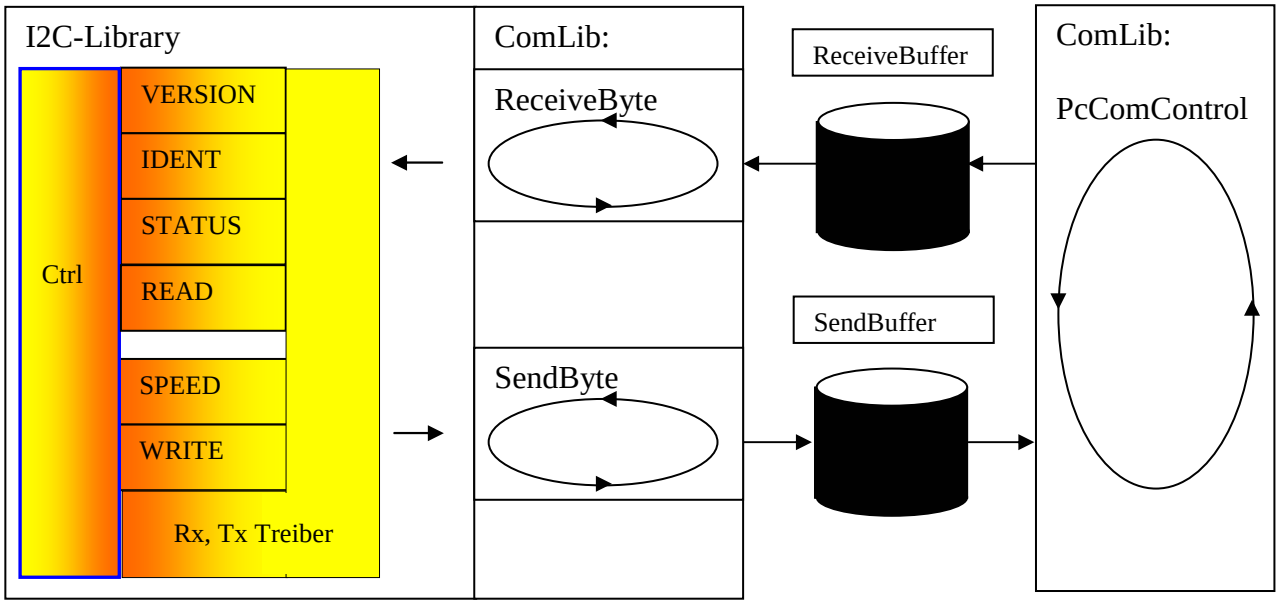


Bild 15:

Der Block Ctrl entspricht dem Programm READ_WRITE_IO_CTRL und der Block “Rx, Tx Treiber” entspricht dem Programm I2C_COM_Port.

Die Software Architektur ist so aufgebaut, dass ein Controller READ_WRITE_IO_CTRL anhand einer Liste “Strukturarrays g_aCommunication” die einzelnen Funktionen VERSION, IDENT, STATUS, READ, SPEED und WRITE mit den Informationen aus dem “Strukturarrays g_aCommunication” aufruft und das Programm I2C_COM_Port für die Datenübertragung treibt.

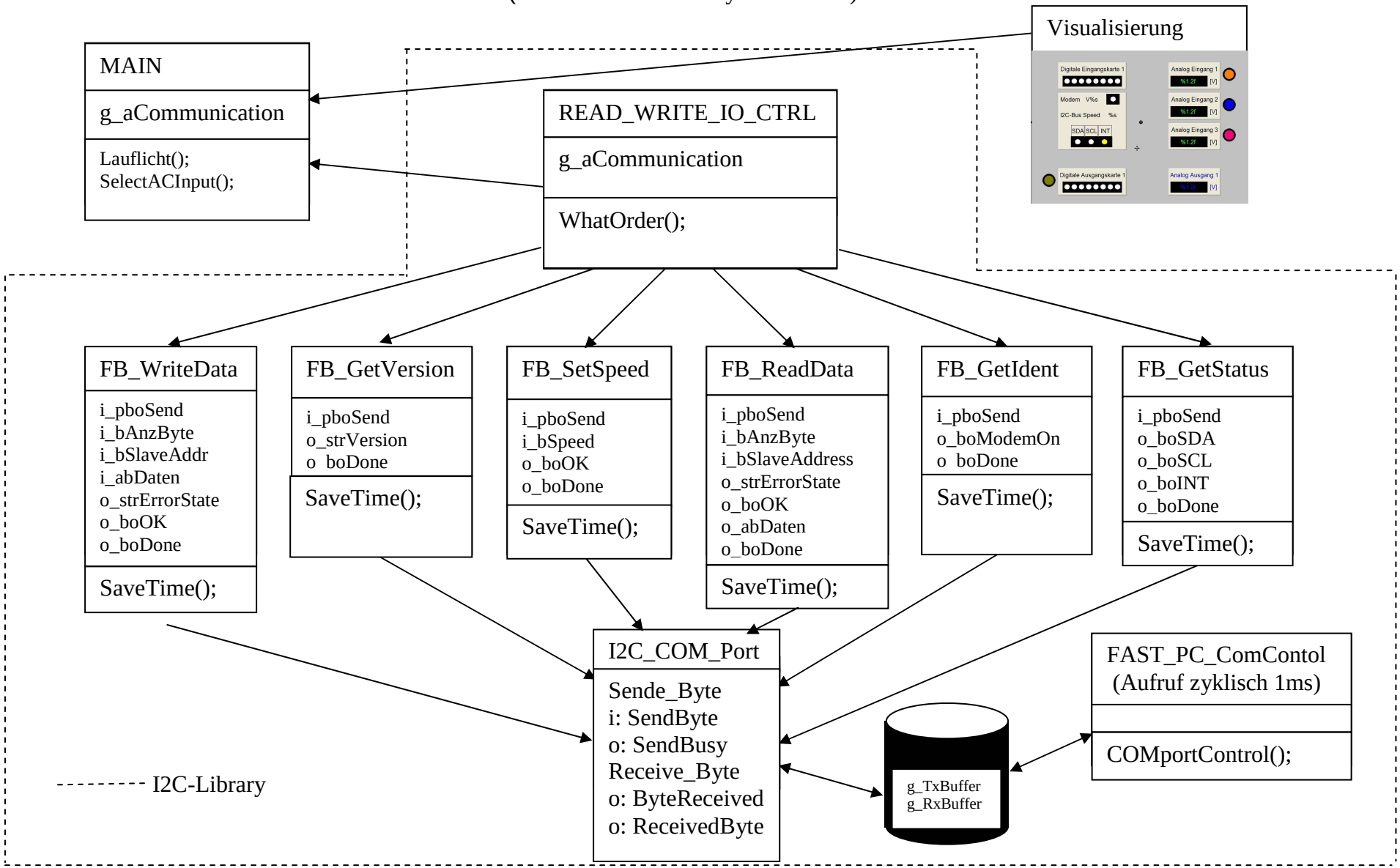
Die einzelnen Funktionen VERSION, IDENT, STATUS, READ, SPEED und WRITE sind eigenständig funktionstüchtig. Ein I2C-Library-User kann diese nach seinem eigenen Konzept aufrufen und verwenden.

Der Controller READ_WRITE_IO_CTRL erlaubt eine Automatisierung der Anfragen von VERSION bis WRITE. Er arbeitet eine Anfrage nach der anderen ab, so wie es die Reihenfolge des Strukturarrays “g_aCommunication” vorgibt.

2.3. Software Design

2.3.1. Klassendiagramm

Bild 16: (Aufruf der Klassen zyklisch 10ms)



Erläuterung zu Klassendiagramm, Bild 16:

Das Programm FAST_PC_ComContol wird in einem 1ms Task von der Priorität 0 aufgerufen. Es treibt die RS232 Hardware mit der Instanz COMportControl, bestehend aus dem Funktionsbaustein (FB) PcComControl von der Beckhoff TwinCat Library COMlib.lib. Die zu sendenden Daten werden im g_TXBuffer abgelegt und durch COMportControl über die Hardware abgesetzt. Die empfangenen Daten legt die Instanz COMportControl im g_RXBuffer bereit.

Die weiteren Programme, FB's oder Funktionen werden in einem 10ms Task von der Priorität 1 aufgerufen.
Die Variable "o_boDone" beschreibt, ob eine Modemanfrage beendet ist oder ob diese noch in Bearbeitung ist.
Mit der Variable "i_pboSend" wird jeweils die Übertragung für eine Modemanfrage gestartet.
Die Funktion SaveTime(); misst die Kommunikationszeit für die jeweilige Modemanfrage.

Das Programm I2C_COM_Port, aufgerufen von dem Programm READ_WRITE_IO_CTRL, ruft die Funktionsblöcke Sende_Byte und Receive_Byte auf.
Der FB Sende_Byte legt ein zu sendendes Byte, übertragen in der "SendByte" Variable, in den g_TXBuffer und meldet nach erfolgter Übertragung, über die Variable "SendBusy", die Sendefreigabe für ein weiteres Byte.
Der FB Receive_Byte überwacht den g_RXBuffer auf neu eingetroffene Daten. Bei erfolgtem Empfang wird dies über die Variable "ByteReceived" gemeldet und in der Variable ReceivedByte wird mitgeteilt, welche Anzahl empfangener Bytes erfolgte. Zusätzlich wird in der Struktur von dem Empfangsbuffer g_RXBuffer die Position des nächsten zu empfangenden gültigen Bytes, in der Variable "RdIdx", eingetragen. Dadurch kann der Anfang der übertragenen Bytes ermittelt werden.
BSP:
Position des ersten Bytes = P1B
Position des nächsten gültigen Bytes = RdIdx
Anzahl empfangener Bytes = ReceivedByte
P1B = RdIdx – ReceivedByte

Die Instanz fbWriteData, bestehend aus dem FB_WriteData, aufgerufen von dem Programm READ_WRITE_IO_CTRL, erlaubt das Senden von Bytedaten an einen I2C-Baustein, der an das I2C-Modem gekoppelt ist. Über die Variable "i_bAnzByte" wird der Funktion, die Menge der zu übertragenden Bytes übergeben. Die Variable "i_bSlaveAddr" bestimmt welcher I2C-Baustein oder welche Ausgangskarte die Bytefolge von dem Byte-Array "i_abDaten" bekommen soll. Nach erfolgter Übertragung meldet die Funktion über die Variable "o_boOK" den Erfolg oder Misserfolg. Die Variable "o_strErrorState" beschreibt per Text einen allfälligen Fehler.

Die Instanz fbReadData, bestehend aus dem FB_ReadData, aufgerufen von dem Programm READ_WRITE_IO_CTRL, erlaubt das Empfangen von Bytedaten aus einem I2C-Baustein, der an das I2C-Modem gekoppelt ist. Über die Variable "i_bAnzByte" wird der Funktion, die Menge der zu empfangenden Bytes übergeben. Die Variable "i_bSlaveAddr" bestimmt aus welchem I2C-Baustein oder aus welcher Eingangskarte die Bytefolge in dem Byte-Array "o_abDaten" abgelegt werden soll. Nach erfolgter Übertragung meldet die Funktion über die Variable "o_boOK" den Erfolg oder Misserfolg. Die Variable "o_strErrorState" beschreibt per Text einen allfälligen Fehler.

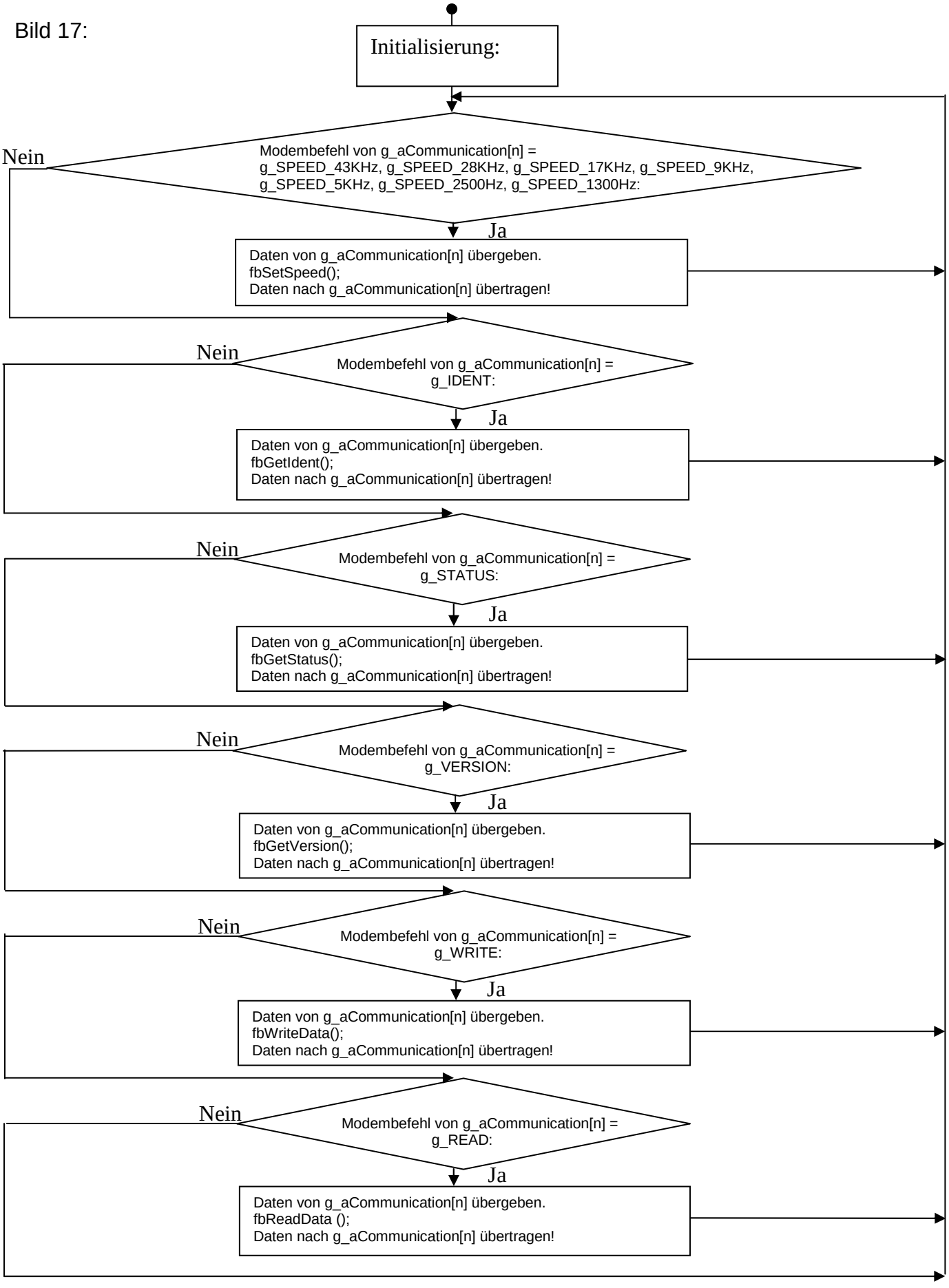
Die Instanz fbGetVersion, fbGetIdent, fbSetSpeed und fbGetStatus melden I2C-Modem Informationen und diese werden ebenfalls von dem Programm READ_WRITE_IO_CTRL aufgerufen. Details zu den Input und Output Variablen können der Tabelle 4.1 entnommen werden.

Das Programm READ_WRITE_IO_CTRL, aufgerufen vom MAIN – Programm, koordiniert die Modemanfragen nach der Abarbeitungsliste g_aCommunication. Aus dem Strukturarray g_aCommunication entnimmt READ_WRITE_IO_CTRL den anzufragenden Modembefehl. Denn der Array representiert die Hardwaretopologie. Pro Arrayplatz wird ein Modembefehl eingetragen und einer nach dem anderen wird abgearbeitet und dies geschieht immer wieder von neuem. Dadurch entsteht die Kommunikationsform Polling, ein ständiges Anfragen und Übertragen der Werte.

Im MAIN – Programm wird die Hardwaretopologie im Strukturarray g_aCommunication eingetragen. Die zu schreibenden Daten sind zum zugehörigen Modembefehl mit in den ensprechenden Arrayplatz einzutragen. Ebenfalls sind die empfangenen Daten, wieder aus dem entsprechenden Arrayplatz, auszulesen und weiter zu leiten.

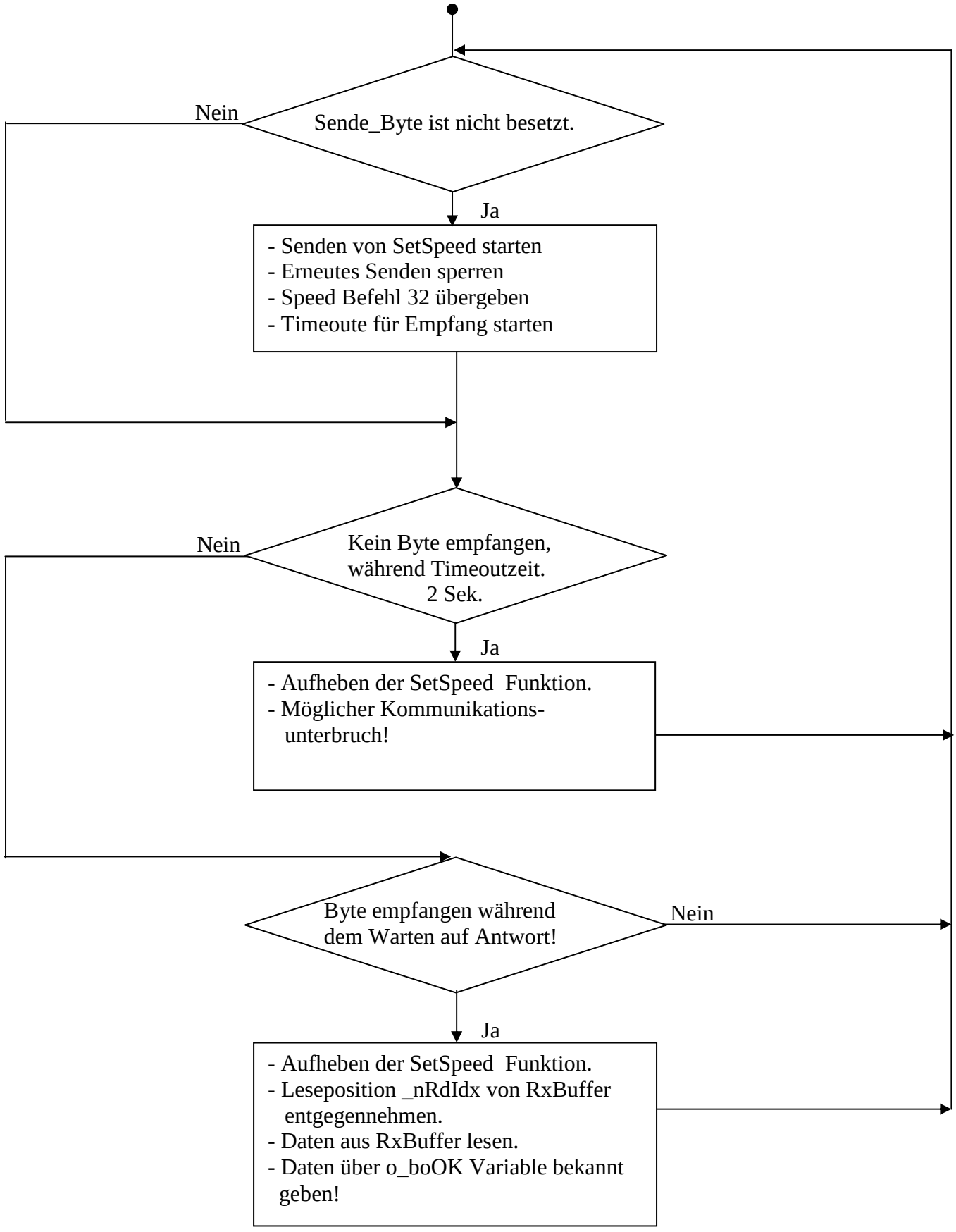
4.3.2 Ablaufdiagramm READ_WRITE_IO_CTRL

Bild 17:



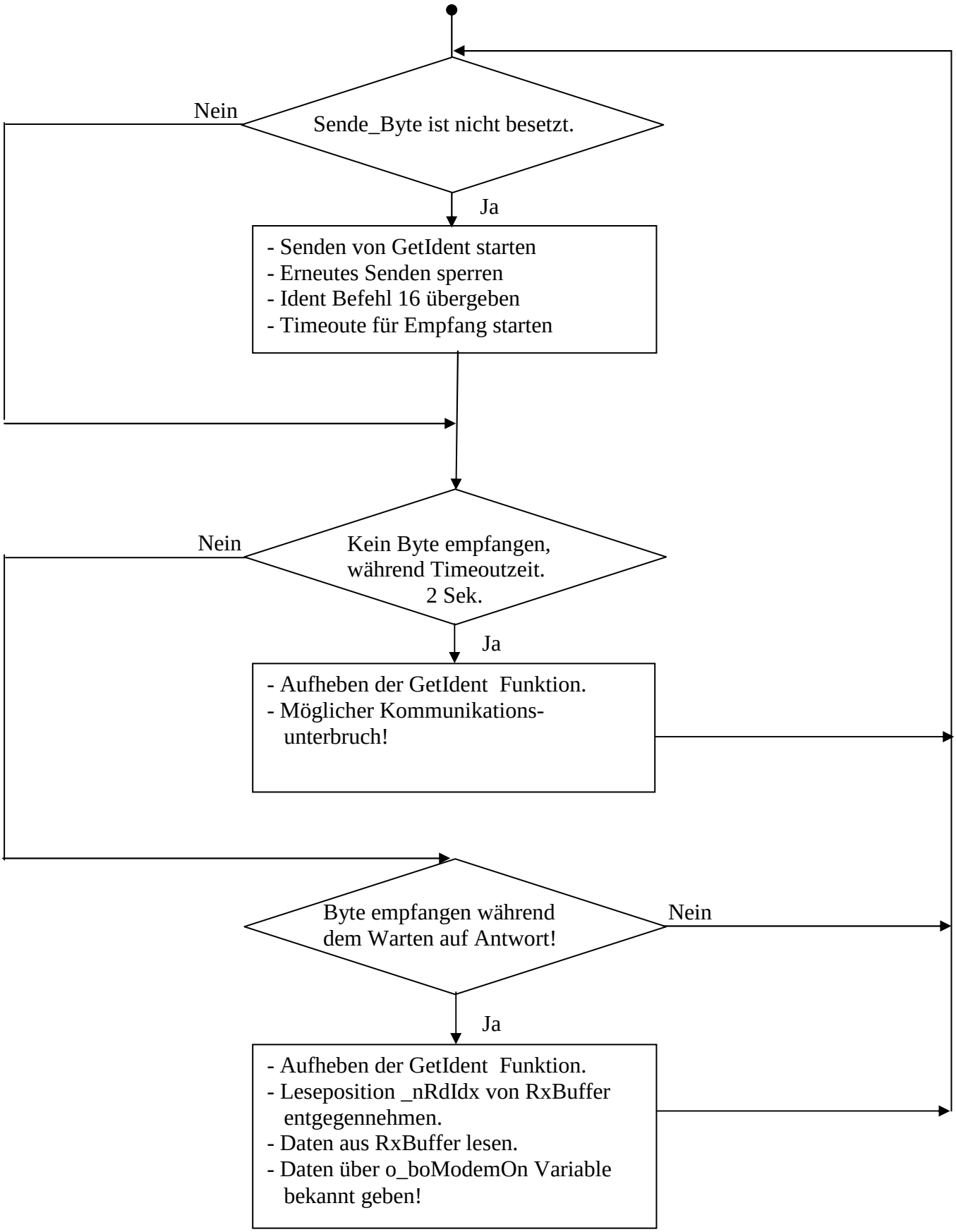
4.3.3 Ablaufdiagramm FB_SetSpeed

Bild 18:



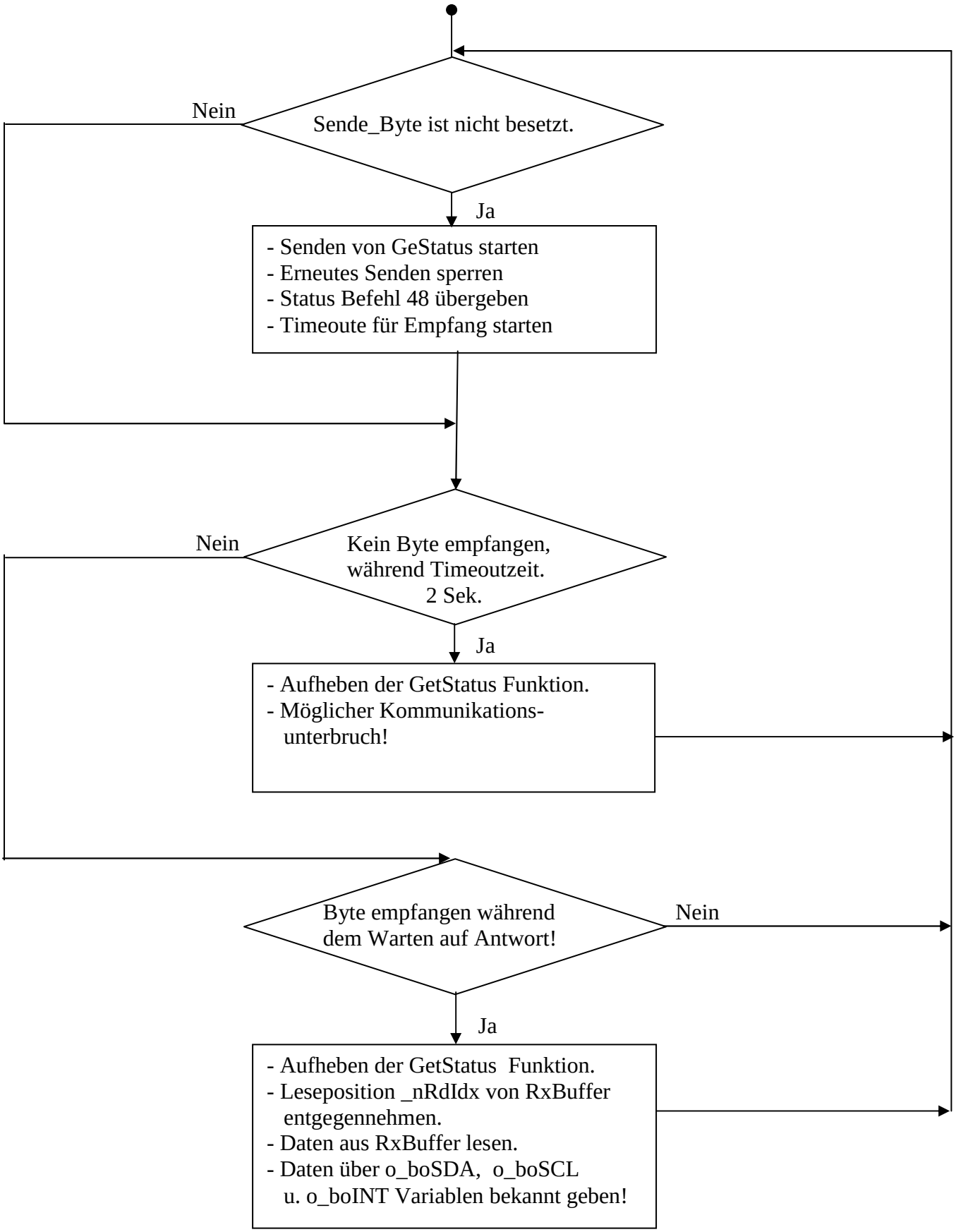
4.3.4 Ablaufdiagramm FB_GetIdent

Bild 19:



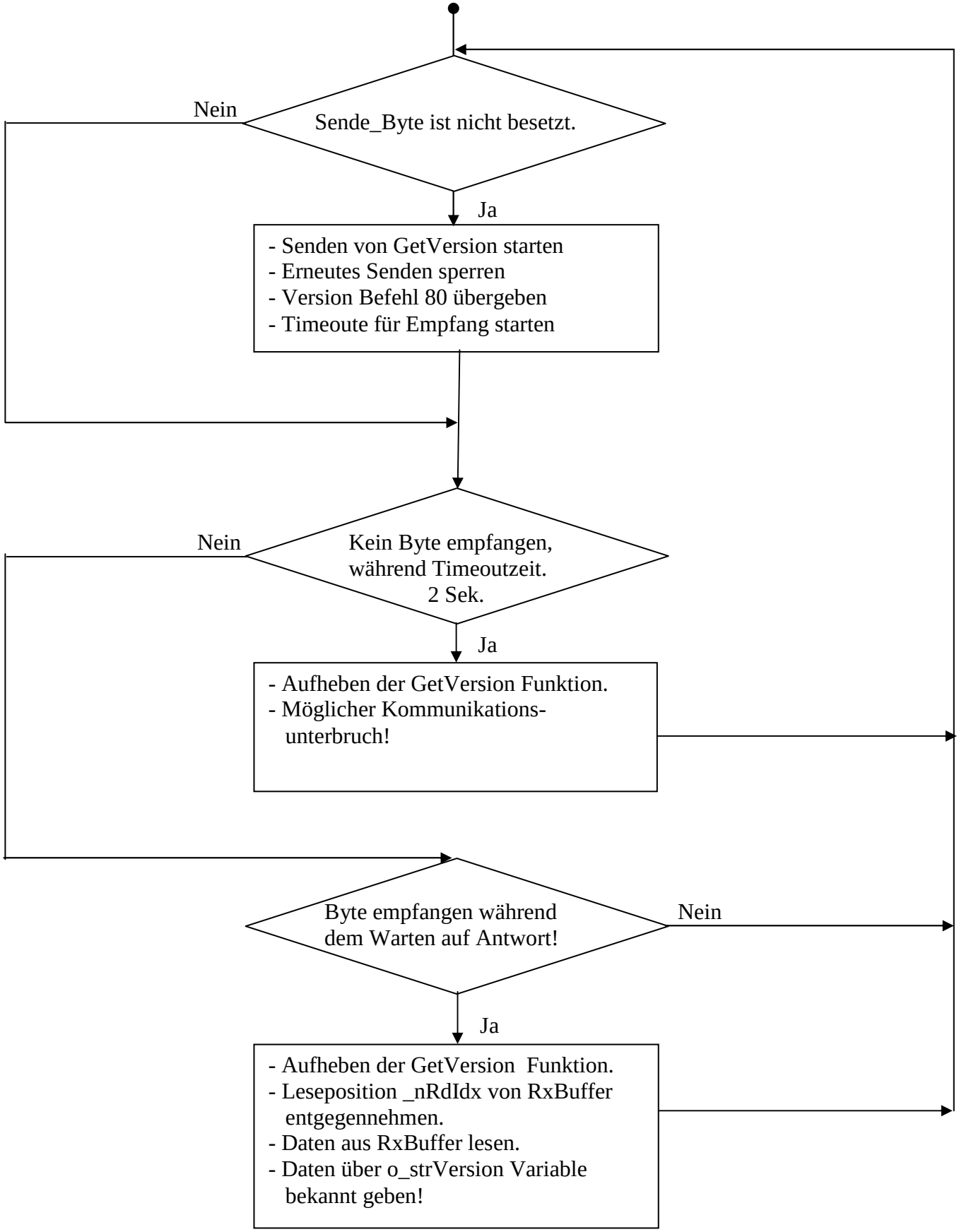
4.3.5 Ablaufdiagramm FB_GetStatus

Bild 20:



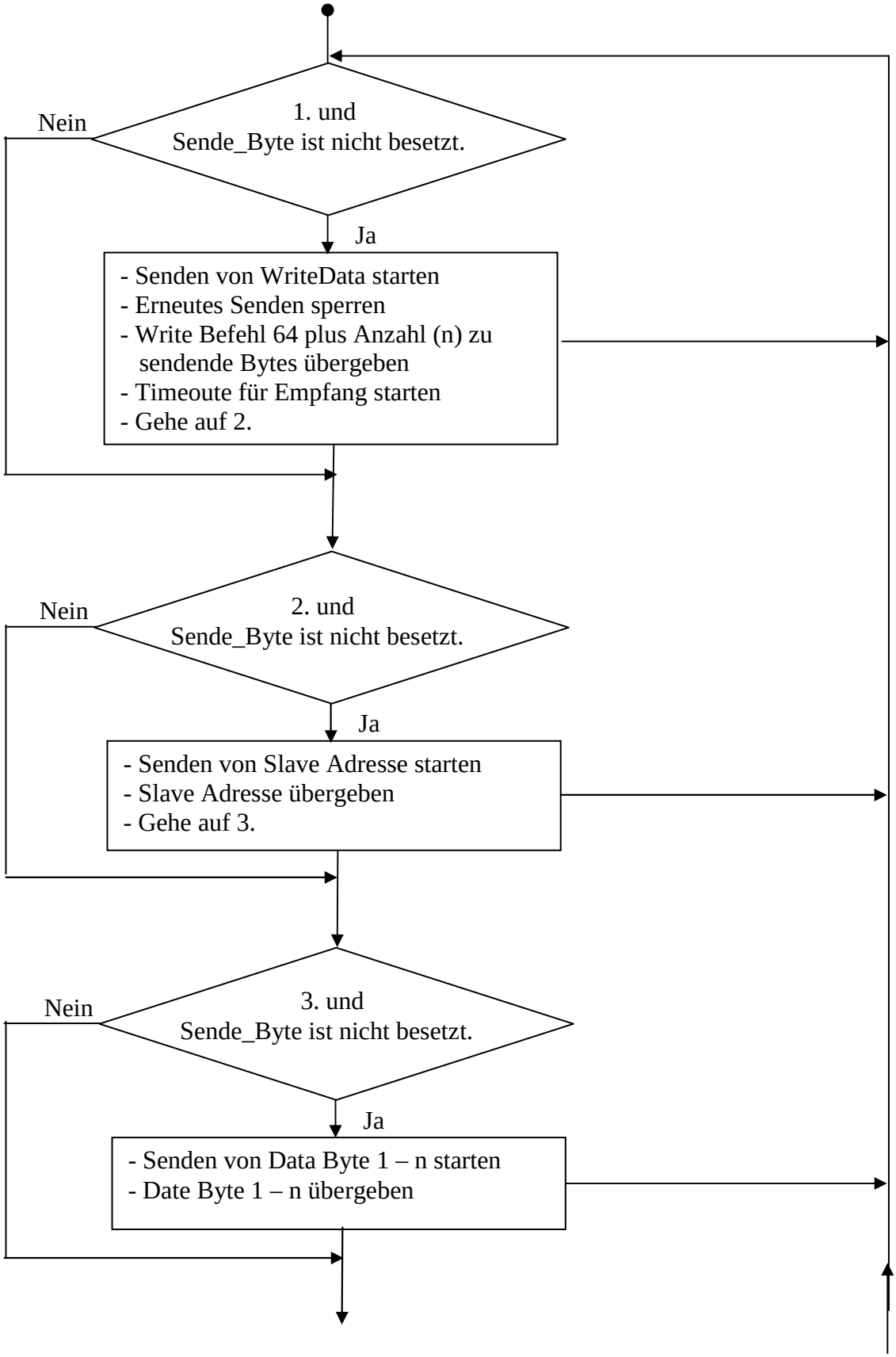
4.3.6 Ablaufdiagramm FB_GetVersion

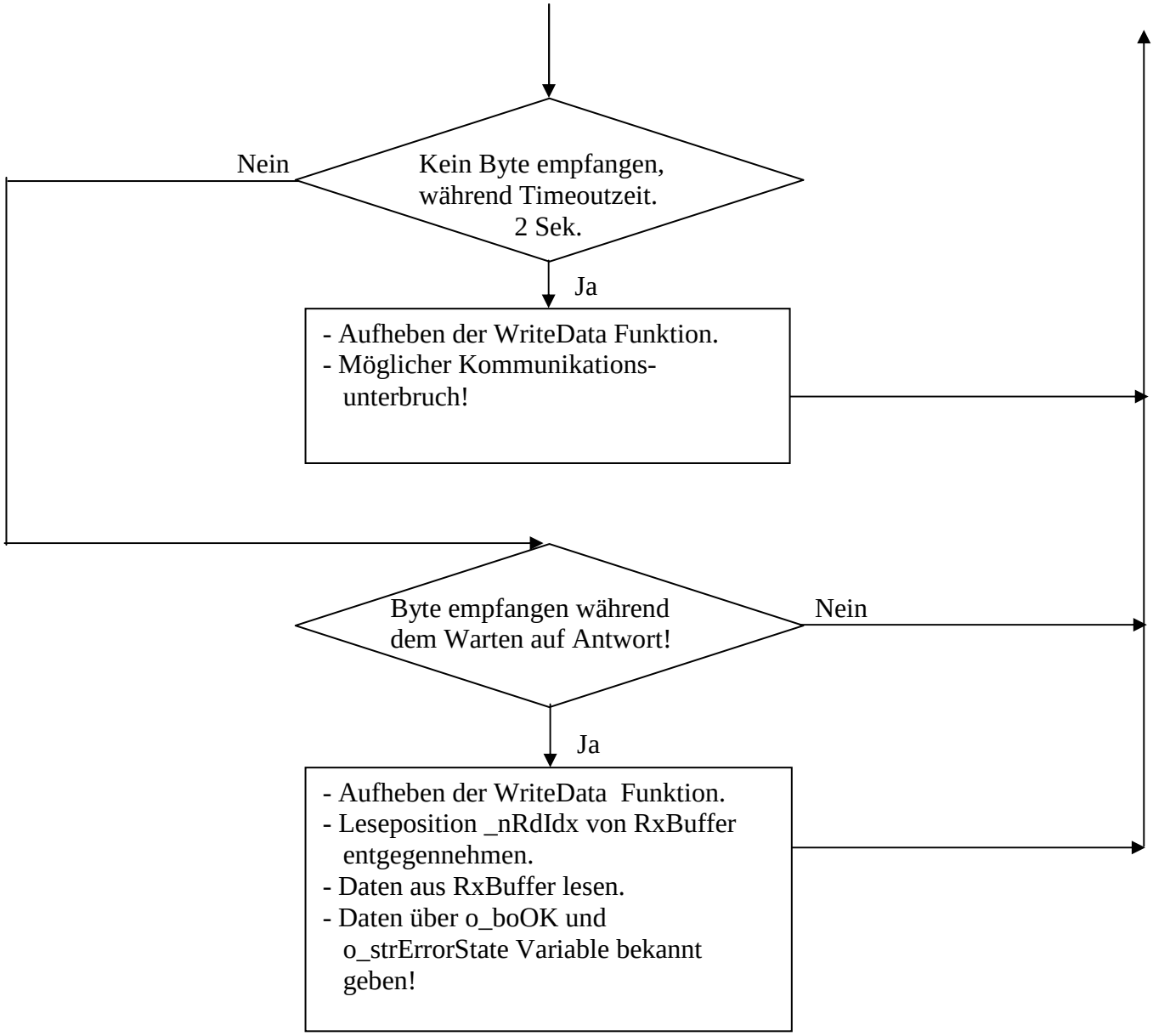
Bild 21:



4.3.7 Ablaufdiagramm FB_WriteData

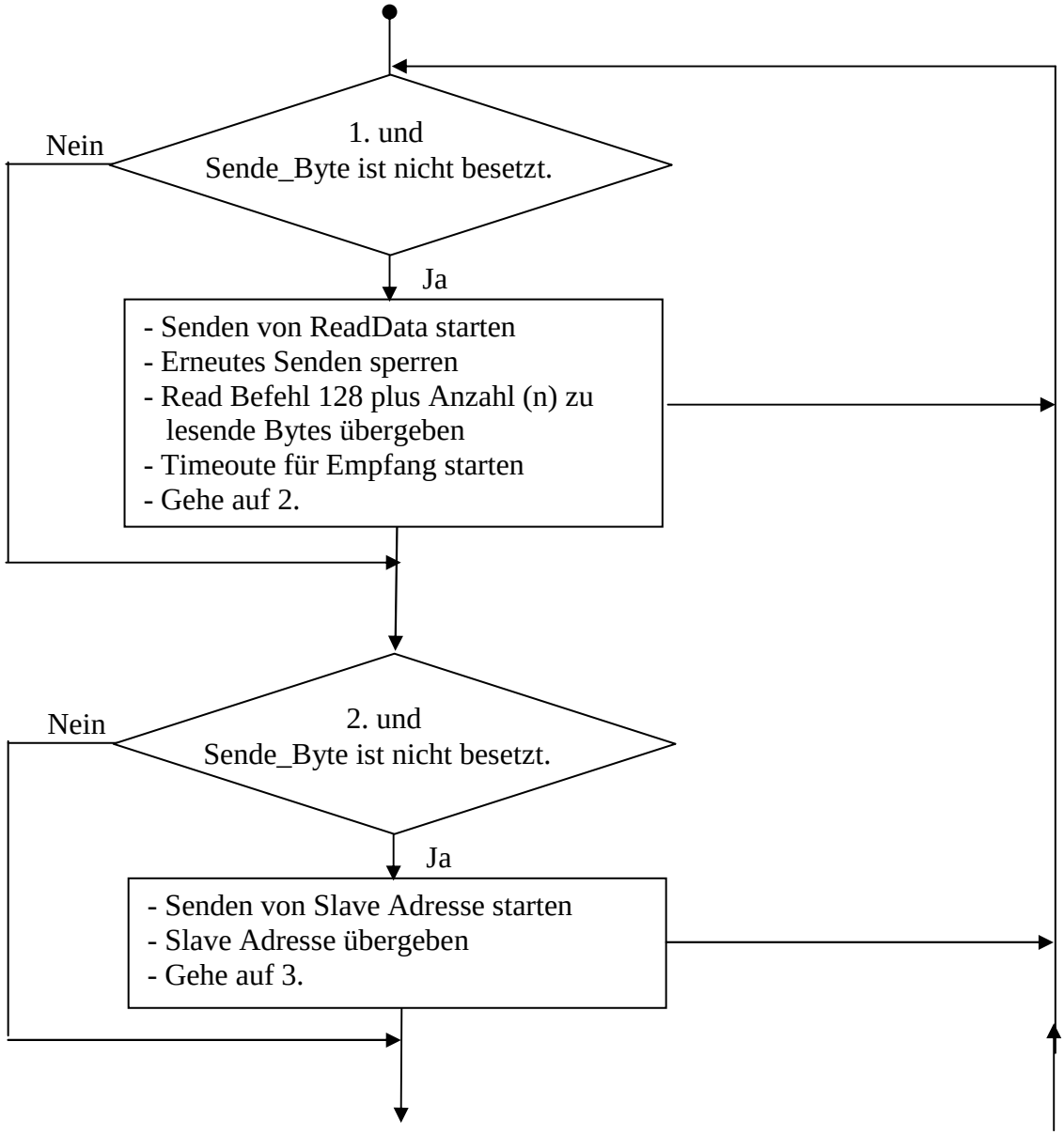
Bild 22:

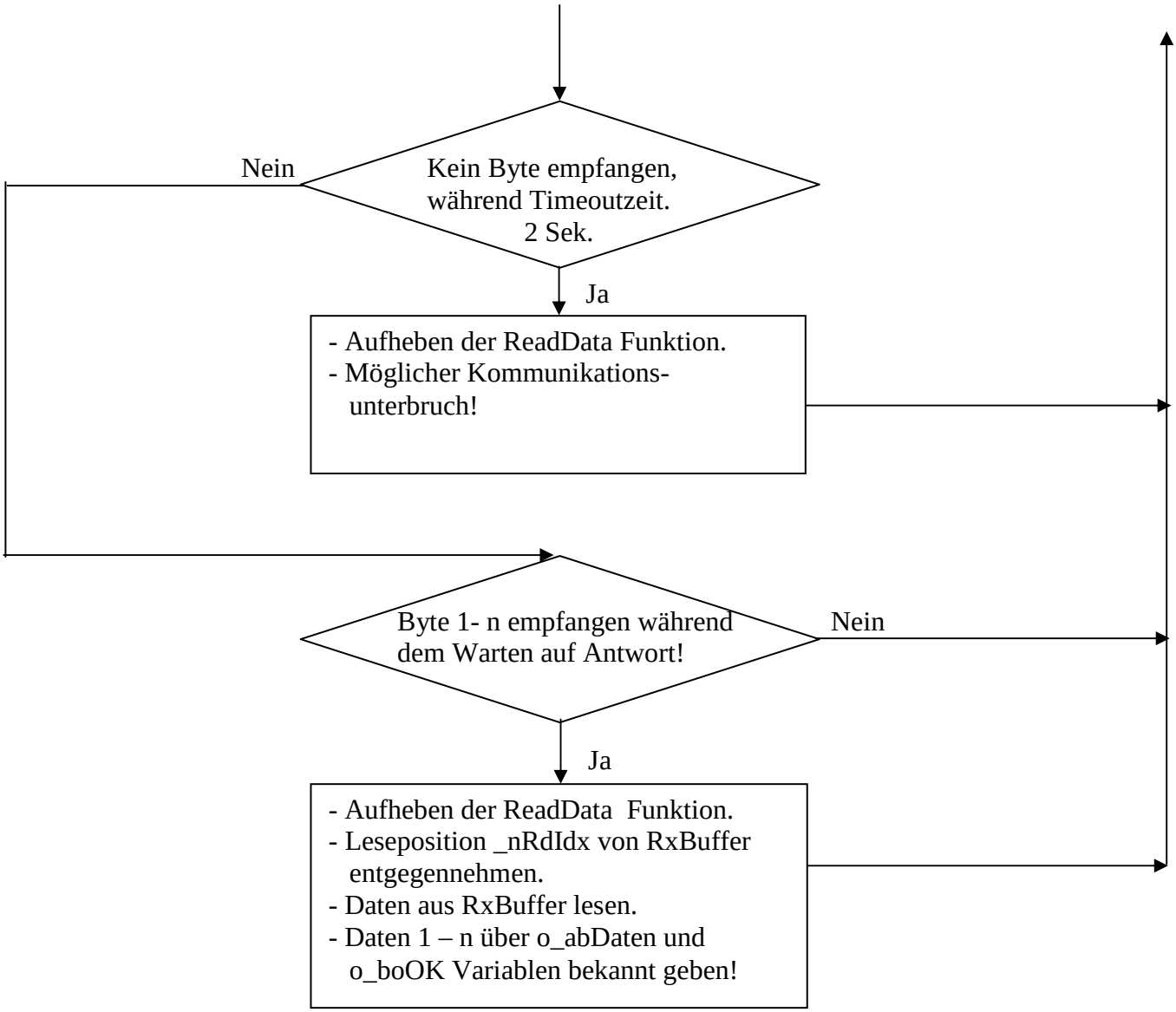



Nein
Kein Byte empfangen, während Timeoutzeit. 2 Sek.
Ja
- Aufheben der WriteData Funktion.
- Möglicher Kommunikations-unterbruch!
Byte empfangen während dem Warten auf Antwort!
Nein
Ja
- Aufheben der WriteData Funktion.
- Leseposition _nRdIdx von RxBuffer entgegennehmen.
- Daten aus RxBuffer lesen.
- Daten über o_boOK und o_strErrorState Variable bekannt geben!

4.3.8 Ablaufdiagramm FB_ReadData

Bild 23:




Nein
Kein Byte empfangen,
während Timeoutzeit.
2 Sek.
Ja
- Aufheben der ReadData Funktion.
- Möglicher Kommunikations-
unterbruch!
Byte 1- n empfangen während
dem Warten auf Antwort!
Nein
Ja
- Aufheben der ReadData Funktion.
- Leseposition _nRdIdx von RxBuffer
entgegennehmen.
- Daten aus RxBuffer lesen.
- Daten 1 – n über o_abDaten und
o_boOK Variablen bekannt geben!

4.3.9 Ergänzungen zu den Ablaufdiagrammen

Aus dem Ablaufdiagramm READ_WRITE_IO_CTRL sieht man, dass nur immer eine Modemanfrage in Bearbeitung ist. Die einzelnen Modemanfragen können nur gestartet werden, wenn auch keine andere Anfrage in Bearbeitung ist. Dies ist auch ersichtlich in den einzelnen Ablaufdiagrammen, wie z. Bsp. im FB_SetSpeed mit der Abfrage “Sende_Byte ist nicht besetzt”. Konnte eine Modemanfrage gestartet werden, dann wird diese für eine erneute Anfrage gesperrt, maximal während zwei Sekunden. Innerhalb dieser Zeit, muss mindestens eine Antwort erfolgt sein. Sonst besteht keine Modemverbindung, oder der angesprochene Teilnehmer mit der entsprechenden Adresse ist nicht an den I2C-Bus angeschlossen.

2.4. Testaufbau

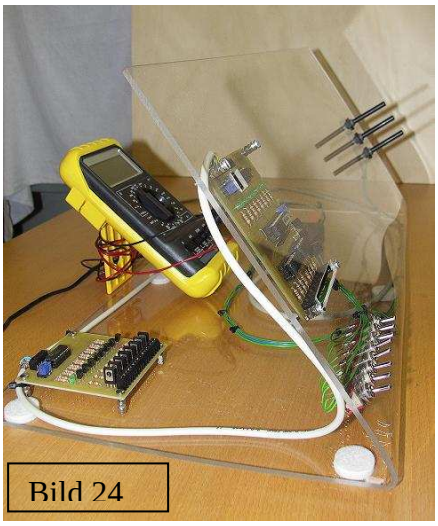


Bild 24

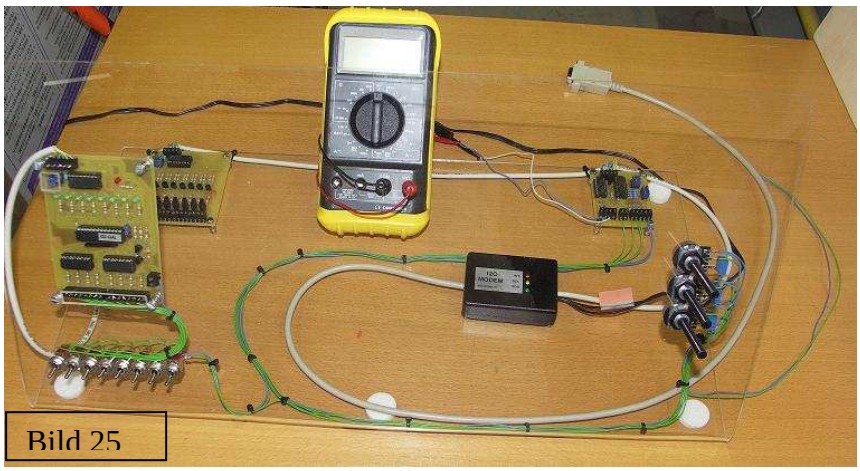


Bild 25

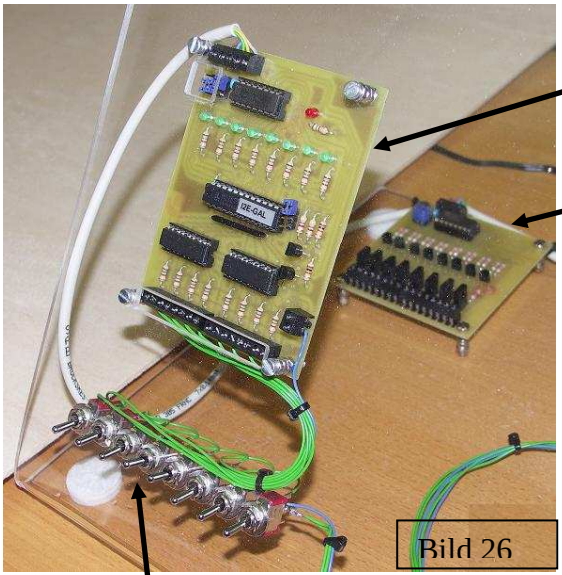


Bild 26

A1

A2

X1

A3

R1 – R3

R4 – R6 (blau)

S1 – S8

A4

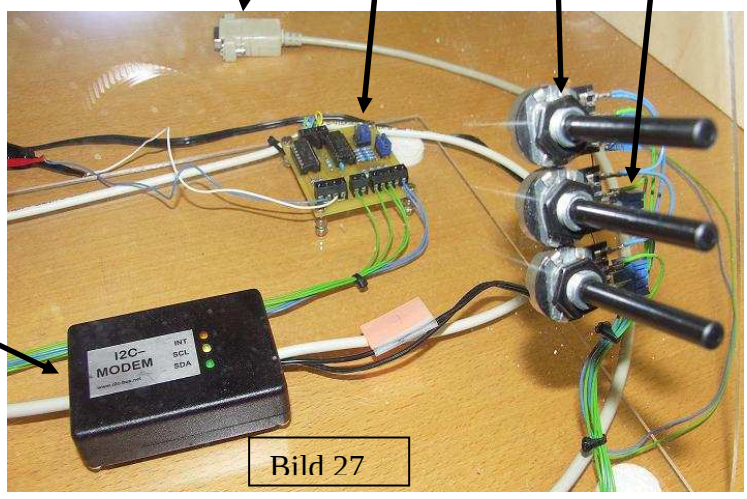
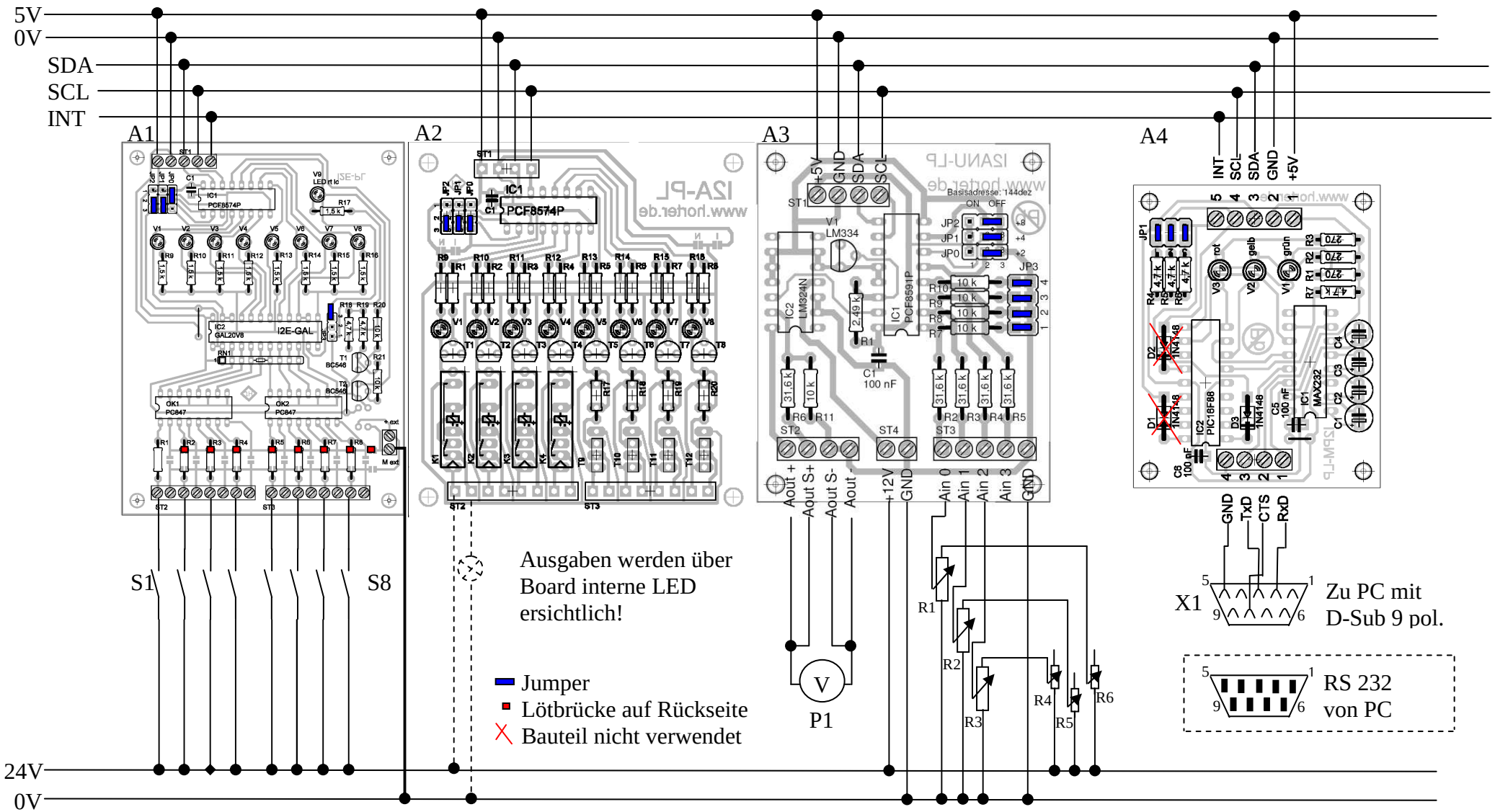


Bild 27

2.4.1. Stückliste

Artikel - Bezeichnung	Artikel - Nummer	Schema - Bezeichnung	Lieferant	Preis CHF
Digitale Eingabekarte	I2E-Bk (Adresse 64 – 78) I2E-PL (Print)	A1	Horter & Kalb	33.45
Digitale Ausgabekarte	I2A-BkT (Adresse 112 – 126) I2A-PL (Print)	A2	Horter & Kalb	25.25
Analogkarte	I2AU-Bk	A3	Horter & Kalb	26.00
I2C-Modem mit D-Sub 9 polig female	I2PM-Bk	A4 X1	Horter & Kalb	53.70
Gehäuse	I2PM-Gg			
Kippschalter	20 21 66	S1-S8	Distrelec	24.10
Potentiometer	74 27 46	R1-R3	Distrelec	10.00
Trimmpotentiometer	74 13 65	R4-R6	Distrelec	5.50
			Total:	**178.00**

2.4.2. Schema


5V
0V
SDA
SCL
INT
A1
A2
A3
A4
S1
S8
Ausgaben werden über Board interne LED ersichtlich!
Jumper
Lötbrücke auf Rückseite
Bauteil nicht verwendet
P1
R1
R2
R3
R4
R5
R6
X1
Zu PC mit D-Sub 9 pol.
RS 232 von PC
24V
0V
Digitale Eingabekarte (8) mit Schalter S1 – S8
Digitale Ausgabekarte (8)
Analogekarte (1 Output, 4 Input)
R1-R3 100kΩ Potentiometer für Analogeingänge (0-3)
R4-R6 100kΩ Trimmpotentiometer für genauen 10V Eingang
I2C-Modem

2.4.3. SW-Einstellungen

I2C-Modem:
Nach Datenblatt von Horter und Kalb arbeitet das I2C-Modem mit 19200 Baud 8N1. Das heisst, da die Symbolwertigkeit gleich 1 ist, dass das I2C-Modem die Fähigkeit hat, 19200Bit/Sekunde zu verarbeiten. Also für die Übertragung von einem Bit, wird 0.052 ms benötigt. 8N1 steht für 8 Datenbits, keine Parität und ein Stopbit.

TwinCAT Taskmanager:
Entsprechend den I2C-Modemdaten wird die Konfiguration eingestellt.

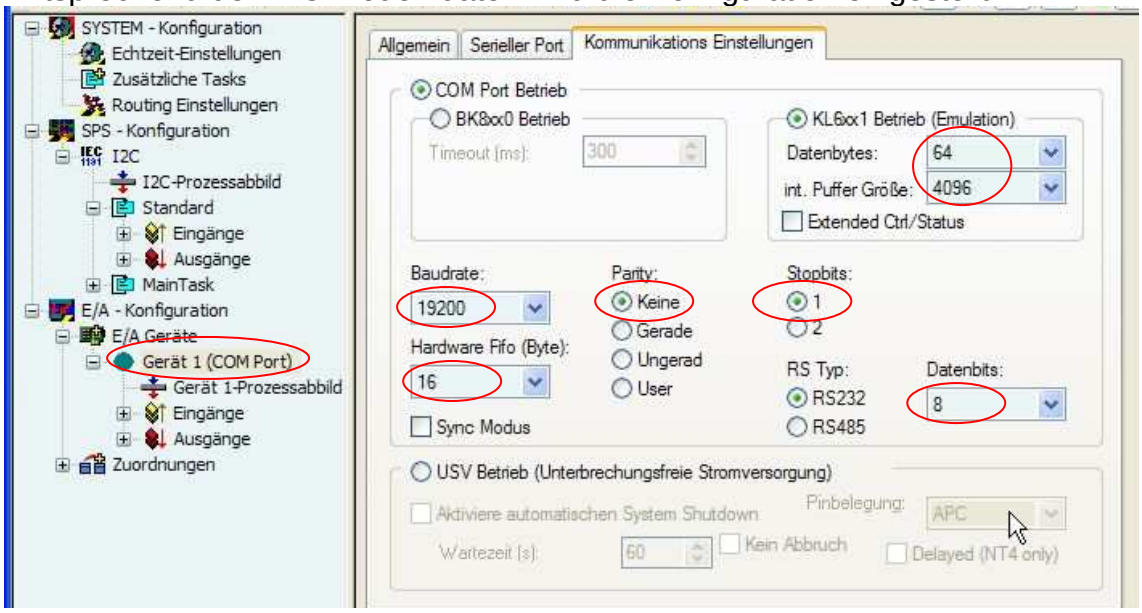


Bild 28:

Der Hardware Fifo steht auf mindestens 16, da das I2C-Modem in einem Schritt max. 16 Byte lesen oder schreiben kann.

2.5. Variante mit Einzeltest

Vorerst wurden die Modemanfragen im 10ms Task aufgerufen, wie es vom Beckhoff Beispiel vorgegeben wird. Diese Antwortzeiten habe ich als lang empfunden. So versuchte ich die Modemanfragen im 1ms Task aufzurufen.

Das Resultat war doch sehr beachtlich, denn die Antwortzeiten wurden dadurch um das zehnfache reduziert.

Funktion	Funktionsaufruf in	Antwortzeit	Antwort
fbGetVersion	Task 10ms	60ms	erfolgt korrekt
	Task 1ms	6ms	erfolgt korrekt
fbGetIdent	Task 10ms	40ms	erfolgt korrekt
	Task 1ms	4ms	erfolgt korrekt
fbSetSpeed	Task 10ms	50ms	erfolgt korrekt
	Task 1ms	5ms	erfolgt korrekt
fbGetStatus	Task 10ms	40ms	erfolgt korrekt
	Task 1ms	4ms	erfolgt korrekt
fbWriteData	Task 10ms	70ms	erfolgt korrekt
	Task 1ms	7ms	erfolgt korrekt
fbReadData	Task 10ms	170ms	erfolgt korrekt
	Task 1ms	17ms	erfolgt korrekt

fbWriteData und fbReadData wurden in dieser Messung nur mit den digitalen Ein- und Ausgabekarten getestet.

Zu berücksichtigen ist dann auch, dass pro Datenaustausch, Senden und Empfangen, drei Taskzyklen benötigt werden. Theoretisch müssten ein Sende- und ein Empfangsbyte innerhalb 3ms gesendet und empfangen sein, bei einer Taskzeit von 1ms. Die Praxis zeigt aber, dass es etwas länger dauert; 4ms.

Der Flaschenhals ist also die PC-RS232 Schnittstelle und das I2C-Modem.
Wieso das I2C-Modem? Nach der Berechnung ist die schnellste Übertragungszeit vom I2C-Modem für ein Bit 0.052ms. Stellt man aber den I2C-Bus auf die Taktrate von 43kHz, so ist ein Bit auf dem I2C-Bus in 0.023ms übertragen. Das I2C-Modem ist nur halb so schnell.

Auf Grund des obigen Resultats versuchte ich eine weitere Reduktion der Antwortzeit, in dem ich einen 0.5ms Task erzeugte. Leider funktionierte die Kommunikation danach nicht mehr. Es scheint, dass die PC-RS232 Schnittstelle maximal mit nur einem 1ms Task getrieben werden kann. Es wird auch im “Beckhoff Information System 12/2007” nur immer von einem 1ms Task gesprochen.

2.6. Gesamttest mit Validierung der Funktionsanforderung über Beispielprojekt

Als erstes habe ich ein Testprogramm geschrieben, wo die Interaktion zwischen Bediener und Anzeigen, um eins bis zwei Sekunden verzögert waren. Die Problematik dabei war, dass bei einer doch langandauernden Datenübertragung, die Interaktion zwischen Bediener und Anzeige erst sichtbar wurde, als der Kommunikationsteil vollständig abgearbeitet war. Diese Erkenntnis hat mich zum folgenden Konzept gebracht: Der Kommunikationsteil, die ganze I2C-Library, habe ich komplett in einem eigenen Task untergebracht, der alle 1ms aufgerufen wird. Der Programmteil, Anwenderprogramm, wird in einem separaten langsameren Task aufgerufen, der in diesem Test alle 3ms erfolgt.
Die Task laufen asynchron nebeneinander her, wobei der 1ms-Task die höhere Priorität hat.

4.6.1 Kommunikationszeit erfassen

Auf Grund der Kommunikationsprobleme die vorgängig gemacht wurden, habe ich im Programmteil “READ_WRITE_IO_CTRL” noch die Aktion “SaveTime” integriert, um die Kommunikationszeit erfassen zu können.

Mit der ersten Messung werden vier Modembefehle, eine digitale Eingangskarte, eine digitale Ausgangskarte und eine Analogkarte gemessen:

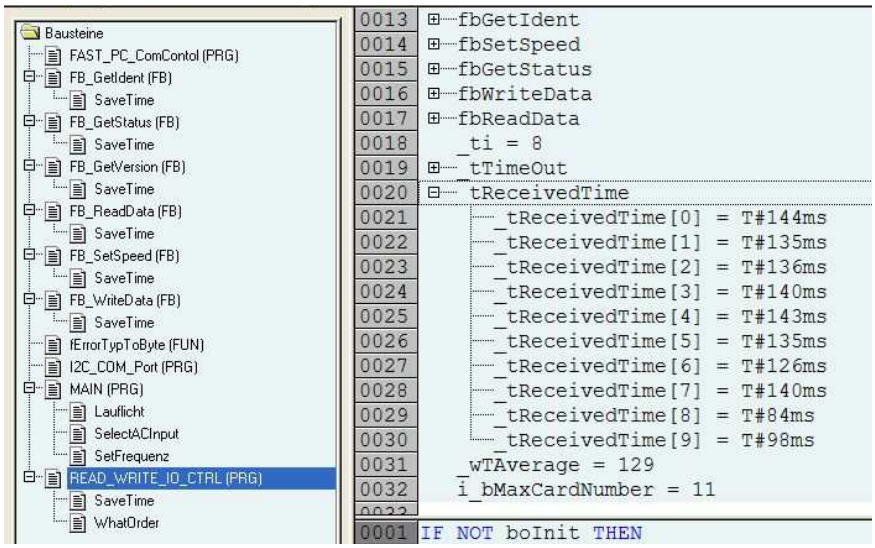


Bild 29:

Dabei sieht man, dass die durchschnittliche Kommunikationszeit “_wTAverage” 129ms beträgt.

Mit der zweiten Messung werden vier Modembefehle, zwei digitale Eingangskarten, zwei digitale Ausgangskarten und zwei Analogkarten gemessen:

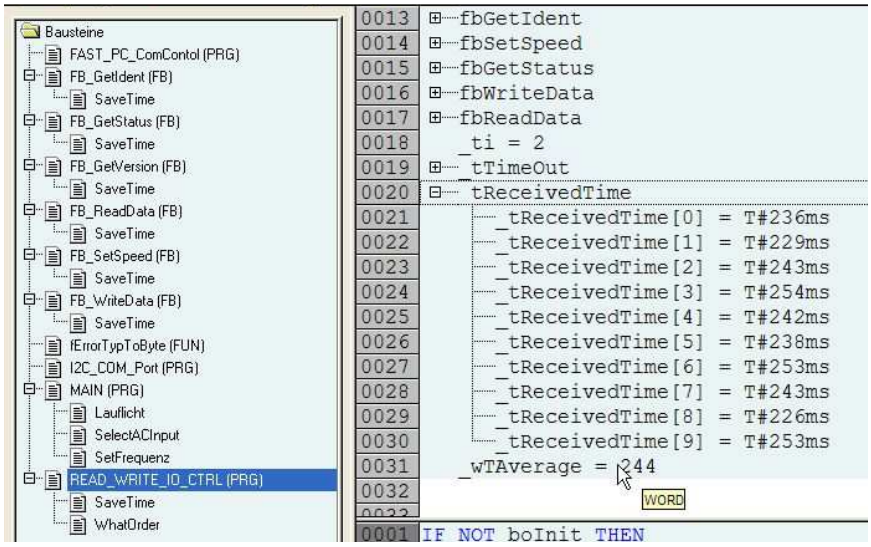


Bild 30:

Hier sieht man, dass die durchschnittliche Kommunikationszeit “_wTAverage” bereits 224ms beträgt.

Daraus ziehe ich die Konsequenz, dass die I2C-Library nicht mehr als 12 Modemanfragen unterstützt, da sonst die Kommunikationszeit zu lange dauert. Dies entspricht z. Bsp. 4 digitalen Eingangskarten, 4 digitalen Ausgangskarten und einer Analogkarte mit 4 analogen Eingängen und einem analogen Ausgang. Eine Analogkarte benötigt alleine vier Modemanfragen.

4.6.2 Validierung der geforderten Aufgabenstellung

Hier werden noch die Punkte, Programm-Beispiel für Schulungen oder Heimanwendungen und die geforderten Funktionen von der I2C-Library geprüft.

Zu prüfendes Element		**Funktion**	**Bemerkung**	**Resultat**
Programm-Beispiele	Lauflicht	Ausgänge beliebig setzen		OK
	SelectACInput	analoge Eingangswerte einlesen analoge Ausgangswerte nach Bedarf stellen		OK
	SetFrequenz	Ausgänge beliebig setzen		OK
	MAIN	Einfache Anbindung der I2C-Karten von Horter u. Kalb		OK
	VISU	Darstellen sämtlicher Werte		OK
I2C-Library		Einlesen von digitalen Eingängen Einlesen von analogen Eingängen Anfragen von Modeminformationen - Version - Modem vorhanden - SDA Leitung - SCL Leitung - INT Leitung Setzen von digitalen Ausgängen Setzen von analogen Ausgängen		OK OK OK OK OK OK OK OK OK
		Lesen eines bestimmten digitalen Einganges	Funktion ersetzt durch 8Bit-Portwert	Nicht realisiert, da immer ein Byte gesendet wird.
		Setzen eines	Funktion ersetzt durch	Nicht realisiert, da

		bestimmten digitalen Ausganges	8Bit-Portwert	immer ein Byte gesendet wird.
Schreiben auf LCD-Anzeige			Funktion nach Pflichtenheft nicht gefordert	Nicht realisiert, SPS-Visu kann Anzeige ersetzen.
Lesen von Temperatur-sensor			Funktion nach Pflichtenheft nicht gefordert	Nicht realisiert, kann mit analogem Eingang realisiert werden.
Beschreibung für Library-User			Siehe 4.8	OK

Das Beispielprojekt beinhaltet die Beispiele Lauflicht, Eingänge erfassen, Analogwerte selektieren und Frequenz setzen. Damit werden folgende Funktionsanforderungen abgedeckt:

Von der Soft-SPS können
- digitale Eingänge ausgewertet und dargestellt werden.
- digitale Ausgänge beliebig gesetzt werden.
- analoge Eingangswerte eingelesen und ebenfalls dargestellt werden.
- analoge Ausgangswerte nach Bedarf gestellt werden.

4.6.2 Frequenzmessung an Ausgang

Als weiteres interessiert, wie schnell ein Ausgang geschaltet werden kann.
Dazu habe ich eine Aktion SetFrequenz geschrieben. Mit dieser wird der Ausgang 0, der digitalen Ausgangskarte, in einer vorgegebenen Zeit ein- und ausgeschaltet.

Messaufbau:
Die durchschnittliche Kommunikationszeit "_wTAverage" beträgt 129ms.

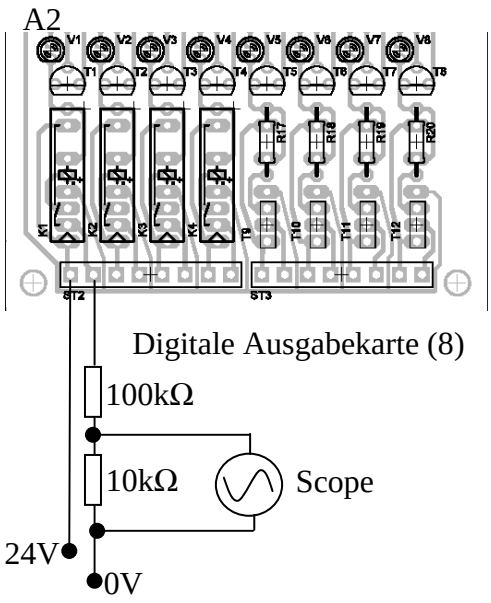


Bild 31:

Die erste Messung zeigt den Signalverlauf von 0,5Hz. Dabei wird das Bit 0, der Analogkarte, zyklisch 1000ms ein- und 1000ms lang ausgeschaltet.

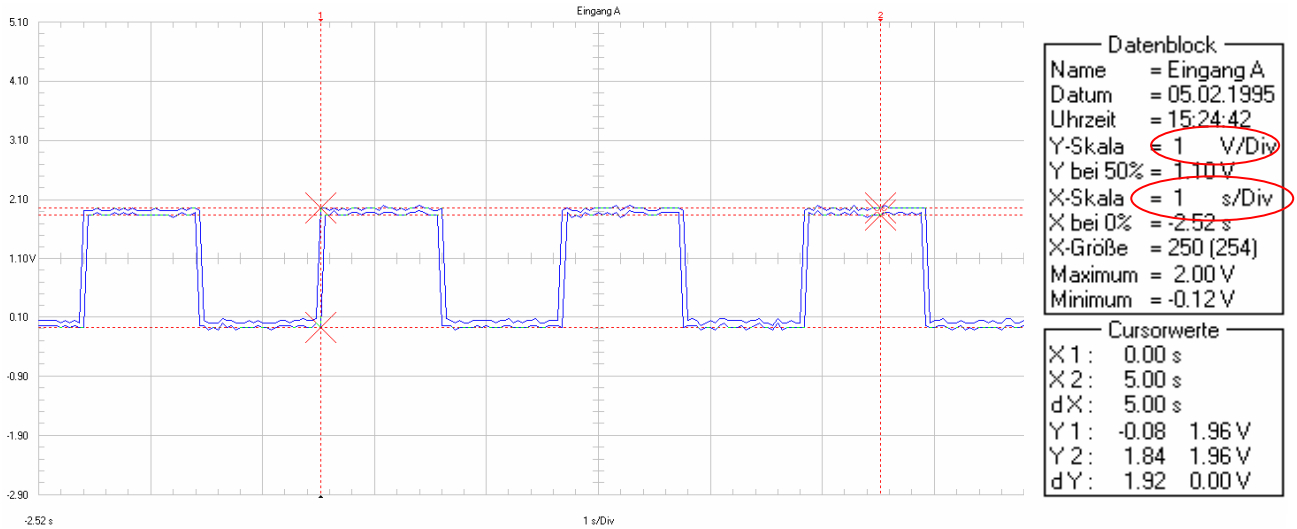


Bild 32:

Die zweite Messung zeigt den Signalverlauf von 2Hz. Dabei wird das Bit 0, der Analogkarte, zyklisch 250ms ein- und 250ms lang ausgeschaltet.

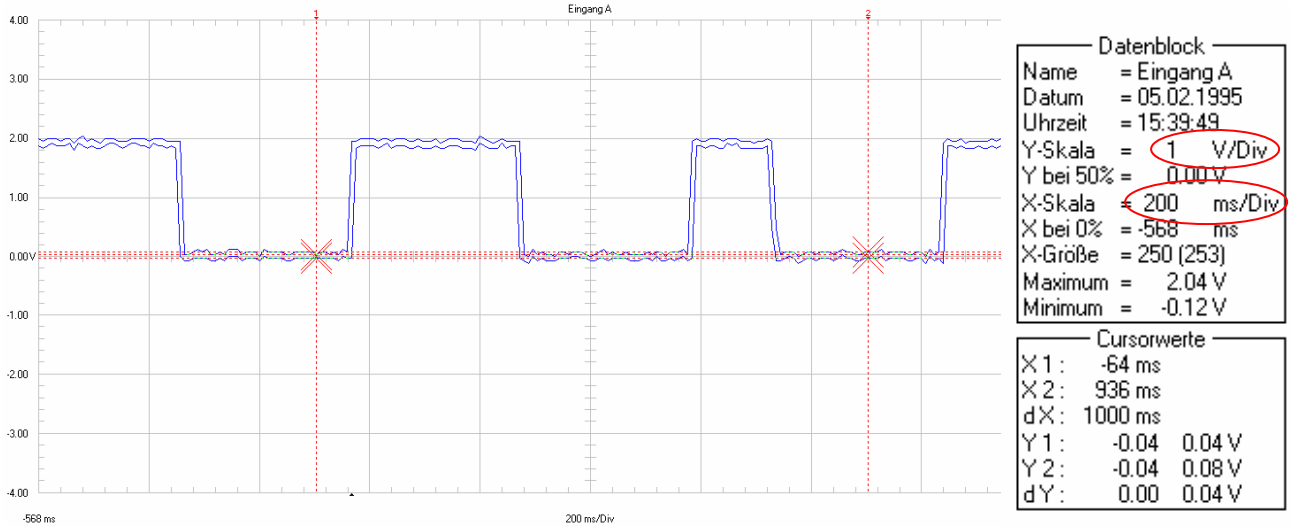


Bild 33:

Bei beiden Messungen wird ersichtlich, dass die Flanken- und Pausendauer ca. um die doppelte Kommunikationszeit (260ms) daneben liegen können. Dies gibt eine Toleranz von +/- ca. Kommunikationszeit. Dies wiederum heisst, je mehr Karten bedient werden, desto ungenauer wird die Frequenz.

Auch aus diesem Test ziehe ich die Konzequenz, dass die I2C-Library nicht mehr als 12 Modemanfragen unterstützt, weil die Ungenauigkeit sonst noch höher würde.

Weiter sind Schaltfrequenzen höher als 2Hz zu vermeiden!

4.6.2 Per USB-Serial Converter auf das I2C-Modem

Da leider in den neueren Notebooks und PC's keine RS232 Schnittstellen mehr zur Verfügung stehen, wollte ich wissen ob per USB-Serial Converter, auf das I2C-Modem zugegriffen werden kann.

Leider hat sich mein Versuch negativ gezeigt. Mit dem USB-Serial Converter von Maxxtro (Distrelec 69 23 82) konnte ich keine Kommunikation zum I2C-Modem aufbauen. Die Kommunikation mit dem FlukeScope per USB-Serial Converter funktionierte aber einwandfrei.

4.7 Technische Daten im Zusammenhang mit der I2C-Library

Daten für Anwenderprogramm:
12 Modemanfragen werden unterstützt, diese entsprechen der Ankopplung von

- 12 digitalen Eingangskarten (96 Eingänge) oder
- 12 digitalen Ausgangskarten (96 Ausgänge) oder
- 6 digitalen Eingangskarten und 6 digitalen Ausgangskarten oder
- 6 digitalen Eingangskarten und 2 digitalen Ausgangskarten und 1 Analogkarte oder
- 2 digitalen Eingangskarten und 6 digitalen Ausgangskarten und 1 Analogkarte oder
- 2 digitalen Eingangskarten und 2 digitalen Ausgangskarten und 2 Analogkarten usw.

Schaltfrequenzen höher als 2 Hz werden nicht empfohlen.

Die wichtigsten Daten von der Hardware:

Daten von digitaler Eingangskarte:
- Maximaler Eingangsstrom 50mA pro Eingang
- Pro Eingangskarte acht Eingänge
- Adressen sind mit Jumpern einstellbar ->
  Karte mit IC-Typ PCF8574: 65, 67, 69, 71, 73, 75, 77, 79
  Karte mit IC-Typ PCF8574A: 113, 115, 117, 119, 121, 123, 125, 127

Daten von digitaler Ausgangskarte:
- Schaltleistung 45V, 1,5A, Ptot 8W pro Transistor Ausgang
- Pro Ausgangskarte acht Ausgänge
- Adressen sind mit Jumpern einstellbar ->
  Karte mit IC-Typ PCF8574: 64, 66, 68, 70, 72, 74, 76, 78
  Karte mit IC-Typ PCF8574A: 112, 114, 116, 118, 120, 122, 126, 128

Daten von Analogkarte:
- Analogeingänge -> Spannungen von 0 - 10V
- Analogausgang -> Spannungen von 0 - 10V
- Pro Analogkarte vier analoge Eingänge und ein analoger Ausgang
- Adressen sind mit Jumpern einstellbar ->
  Karte mit IC-Typ PCF8591: 144, 146, 148, 150, 152, 154, 156, 158

Weitere Info kann dem Anhang entnommen werden.

4.8 Installation und Benutzeranleitung

Diese Anleitung zeigt, wie mit einem Beckhoff TwinCat-Projekt, per RS232 Schnittstelle, von einem PC, die I2C-Peripheriekomponenten von Horter und Kalb, über dessen I2C-Modem, gelesen und gesteuert werden können.

Der Hardwareaufbau bezieht sich auf das Schema unter 4.4.2 .

Folgende Softwarekomponenten sind zusätzlich zum Entwicklungssystem Beckhoff TwinCat erforderlich:
- Comlib.Lib von Beckhoff (Schnittstelle zum PC-RS232-Port)
- I2C.lib von Lichtensteiger (Schnittstelle zum I2C-Modem)

4.8.1 Projekterstellung mit TwinCat

Nach erfolgreicher Installation der Entwicklungsumgebung TwinCat (Beckhoff), ist das PLC Control wie folgt zu starten:

1. Rechter Mausklick, über dem TwinCat-Icon von der Taskleiste, am rechten Bildschirmecken.
2. Linker Mausklick auf die Auswahl “PLC Control”,

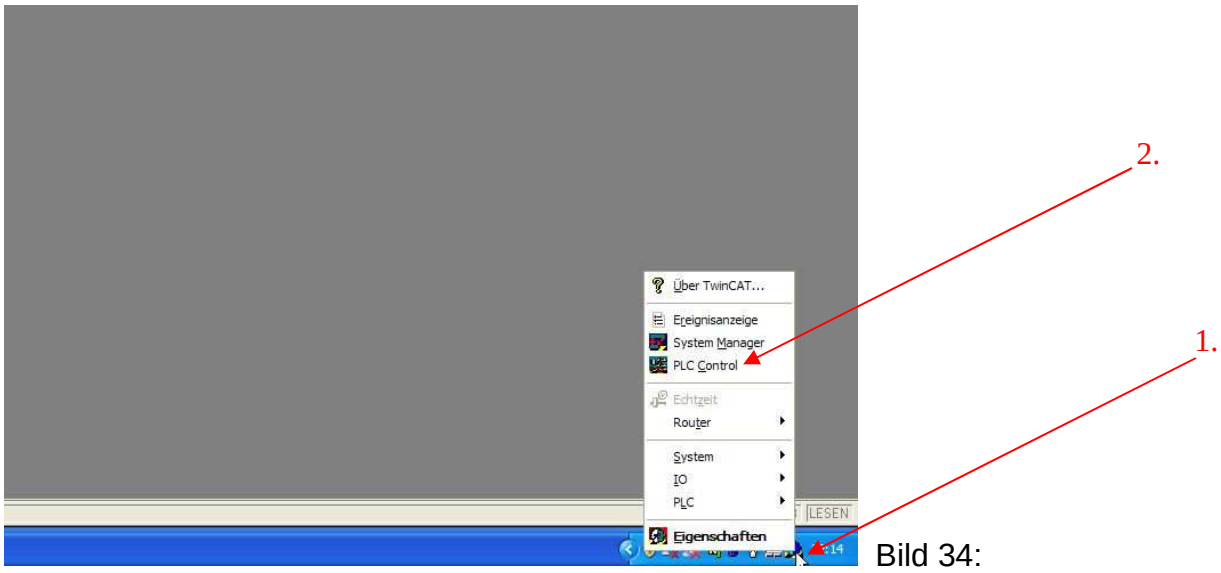


Bild 34:

dann öffnet sich das TwinCat PLC Control.

3. Mit Klick auf “Neu”, unter dem Menupunkt Datei, wird eine Auswahl gestartet.

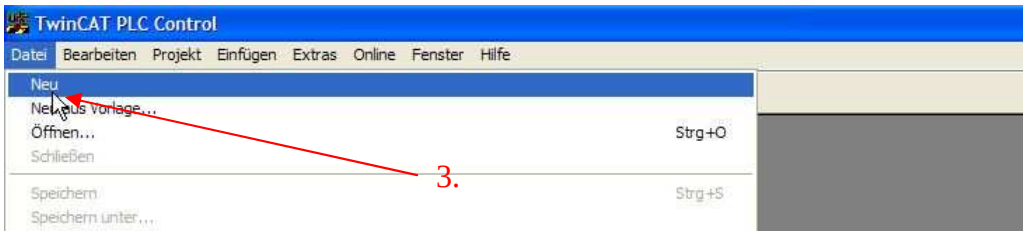


Bild 35:

4. Es ist die Auswahl “PC oder CX (x86)” mit OK zu bestätigen.

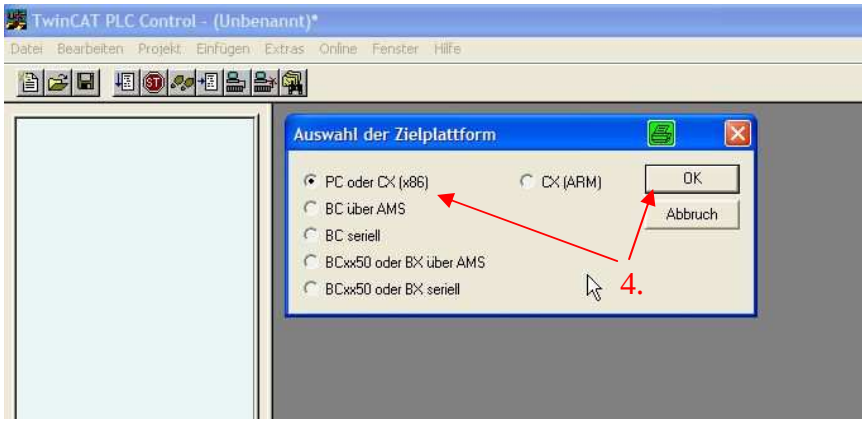


Bild 36:

5. Es wird für den ersten Baustein der Name “MAIN” vorgeschlagen, der so, wenn nichts dagegen spricht, mit OK übernommen werden kann.

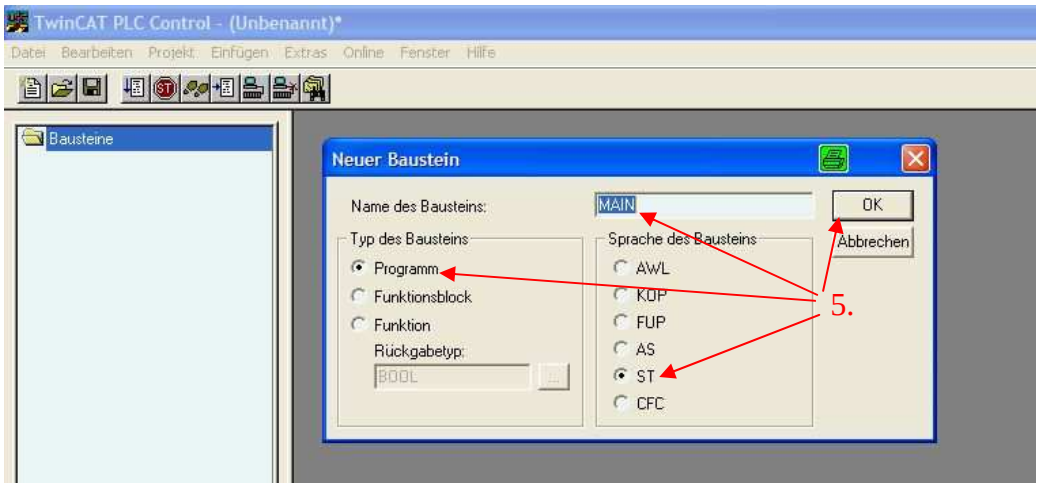


Bild 37:

Der Baustein MAIN hat den Typ Programm und wird in diesem Beispiel in der Sprache ST (Strukturierter Text) geschrieben.

6. Das neu eröffnete Projekt, muss nun noch unter einem Namen abgespeichert werden. Dies erfolgt über das Menu “Datei\Speichern unter ...” (BSP: I2C_Projekt).

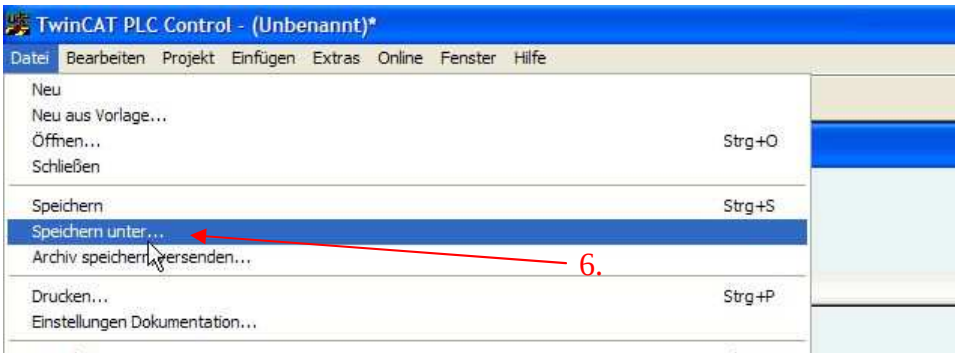


Bild 38:

4.8.2 Integration der COMlib.lib, in das I2C_Projekt

Die “TwinCAT PLC Serial Communication Library” ist bei Beckhoff zu bestellen und ist kostenpflichtig. Nach Erhalt und Entkomprimierung steht dem User ein File TcSerialCom.exe zur Verfügung, das vorerst installiert werden muss. Dies erfolgt mit einem Doppelklick auf das File.
Erst nach der erfolgreichen Installation steht die COMlib.lib Library zur Integration in das I2C_Projekt bereit.

Für die Integration der COMlib.lib sind folgende Schritte durchzuführen:

1. Über den Menupunkt “Fenster\Bibliotheksverwaltung” lässt sich der Bibliotheksverwalter öffnen.

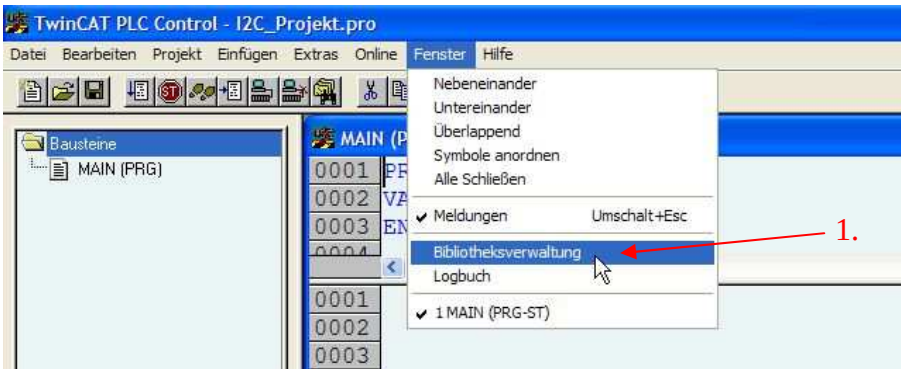


Bild 39:

2. Mit einem rechten Mausklick ins Fenster des Bibliothekverwalters, wird das Kontextmenu geöffnet, das den Menupunkt “Weitere Bibliothek ... Einfg” enthaltet.

3. Mit anwählen von “Weitere Bibliothek ... Einfg” erscheint eine grössere Auswahl an Bibliotheken. Unter diesen ist die COMlib.lib mit Doppelklick anzuwählen. Dadurch wird sie und damit auch die ChrAsc.lib ins Fenster des Bibliotheksverwalters und somit auch ins I2C_Projekt integriert.

   Falls die COMlib.lib Library in der Auswahl nicht zu finden ist, muss der Verzeichnis-pfad kontrolliert werden. Der korrekterweise, nach erfolgter Installation von TcSerialCom.exe, wie folgt lauten sollte:
   “Installationspfad TwinCat”\TwinCat\Plc\Lib\COMlib.lib
   z. Bsp: *“Installationspfad TwinCat”* = C:\Programme

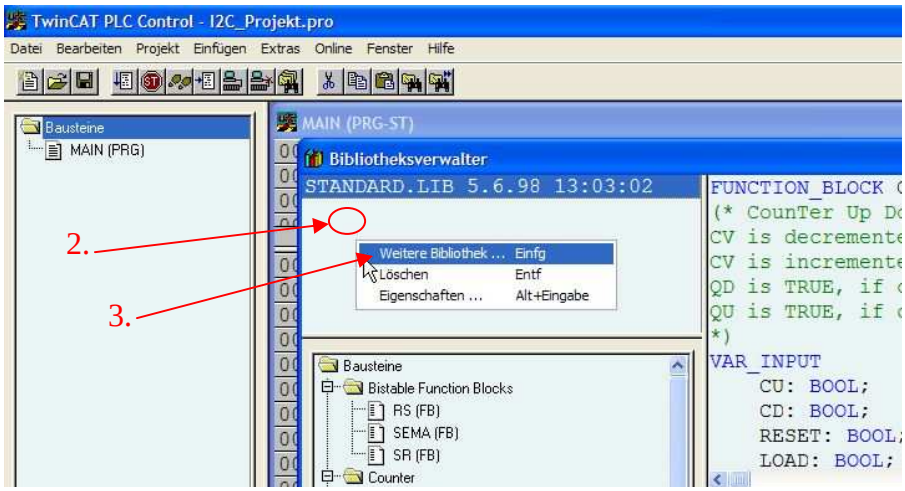


Bild 40:

4.8.3 Integration der I2C.lib, in das I2C_Projekt

Für die Integration der I2C.lib sind folgende Schritte durchzuführen:

1. Die zur Verfügung gestellte I2C.lib soll unter dem Verzeichnis *“Installationspfad TwinCat”*\TwinCat\Plc\Lib\ abgelegt werden. Dadurch ist die Integration gleich der Integration von COMlib.lib durchzuführen.

4.8.4 Taskerzeugung im I2C_Projekt

Es sind zwei Tasks zu erstellen, ein schneller und ein langsamerer.

Bei der Taskerstellung ist zu beachten, dass die CPU-Auslastung nicht an ihre Grenzen kommt. Wenn zum Beispiel ein Task alle 1ms zyklisch aufgerufen wird und sein zu bearbeitendes Programm 950us verbraucht, dann stehen einem weiteren Task nur noch 5% der CPU-Auslastung zur Verfügung, da mit 950us die CPU auf 95% ausgelastet ist.

Der langsamere Task soll zyklisch auf 10ms mit der Priorität von 1 und der schnelle Task soll zyklisch auf 1ms mit der Priorität von 0 eingestellt werden.

Mit dem langsamen Task soll das Main-Programm aufgerufen und mit dem schnellen Task soll das READ_WRITE_IO_CTRL-Programm von der I2C-Library aufgerufen werden.

Die Schritte zur Taskerzeugung sehen wie folgt aus:

1. Man wechselt auf das Register Ressourcen.
2. Mit einem Doppelklick auf die Taskkonfiguration, erscheint der entsprechende Dialog.

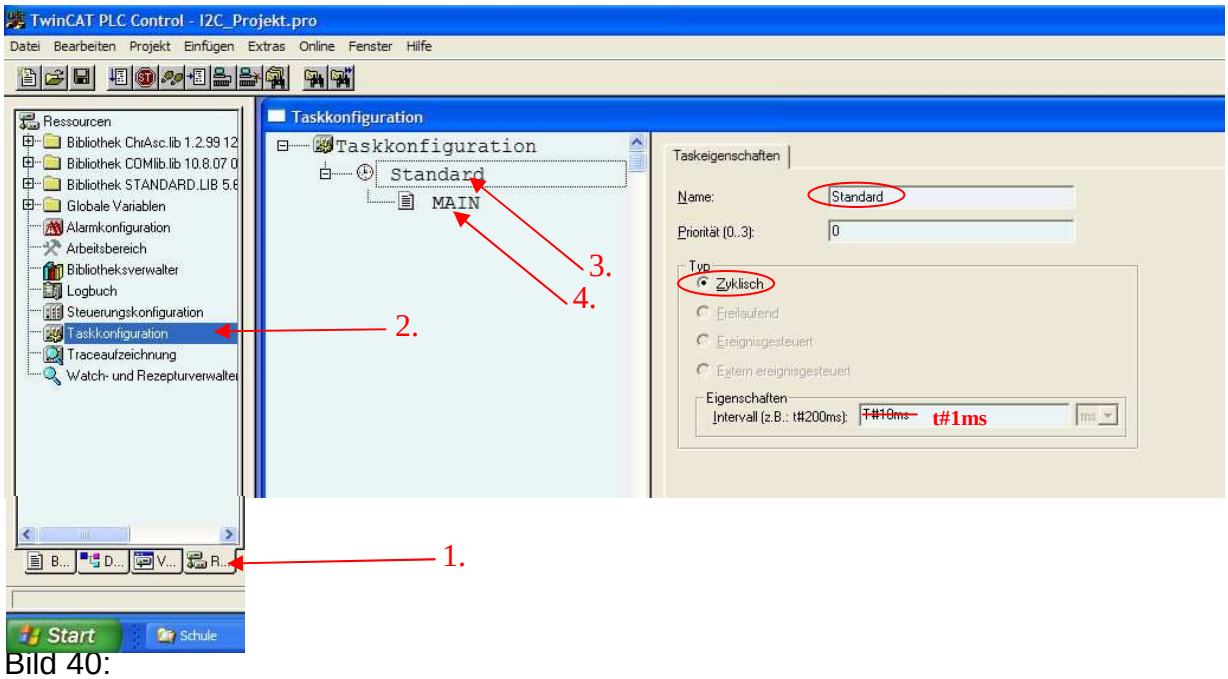


Bild 40:

Ein Standard Task ist bereits schon angelegt. Dieser ist aber zu modifizieren.

3. Das Anklicken von Standard ermöglicht eine Namensänderung, die hier mit FastTask erfolgen soll. Die Priorität ist auf null und der Intervall auf t#1ms zu setzen.

4. Das Anklicken von MAIN ermöglicht eine Namensänderung, die hier mit READ_WRITE_IO_CTRL erfolgen soll. Dies ist das Programm aus der I2C- Library, das der FastTask aufruft.

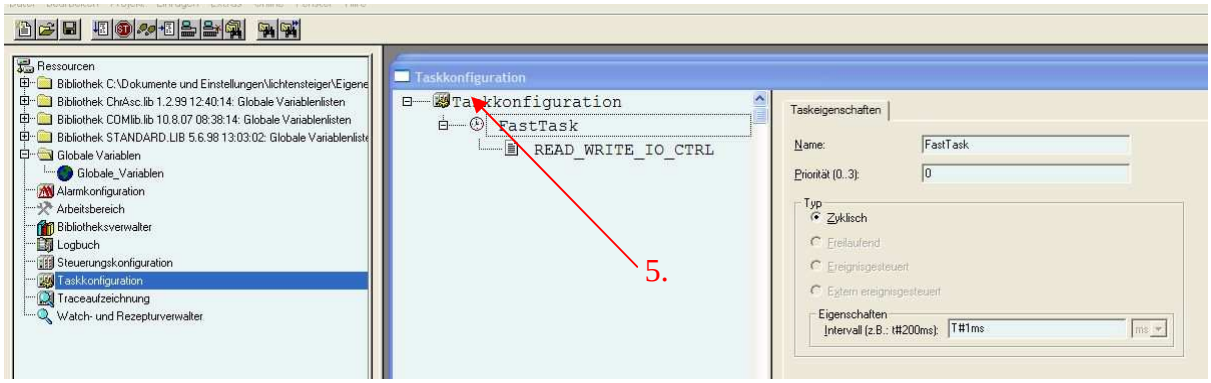


Bild 41:

5. Das Erstellen des langsamen Tasks wird erreicht, in dem mit der rechten Maustaste über dem Text Taskkonfiguration, das Kontextmenu geöffnet und mit dem Anklicken von "Task anhängen" der Task "NeueTask" erstellt wird.

6. Das Anklicken von NeueTask ermöglicht eine Namensänderung, die hier mit MAINTask erfolgen soll. Die Priorität ist auf eins und der Intervall auf t#10ms zu setzen.

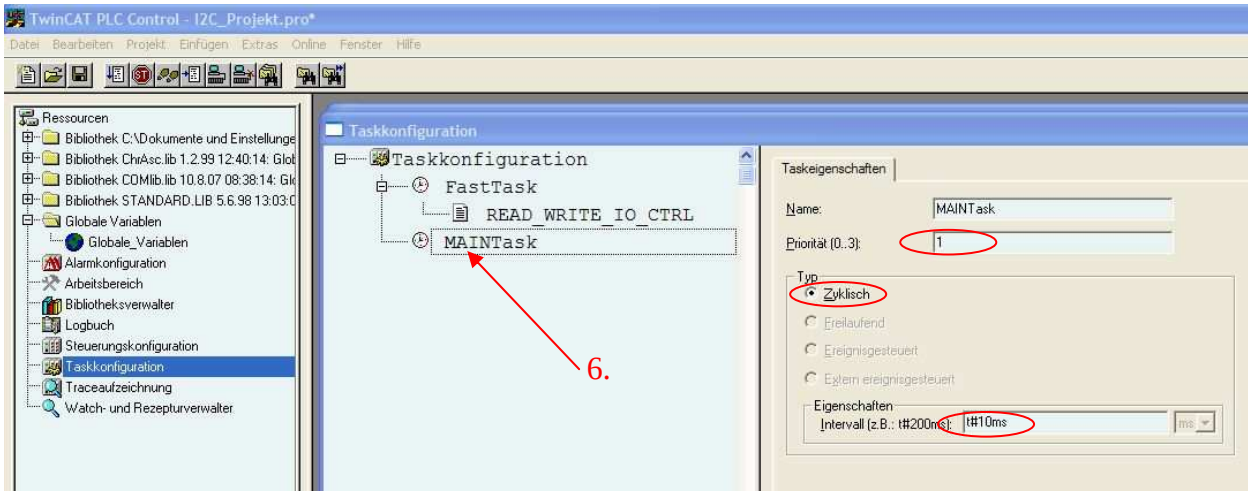


Bild 42:

7. Dem MAINTask muss nun noch über das Kontextmenu (rechte Maustaste), mit „Programmaufruf anhängen", das Programm MAIN eingetragen werden.

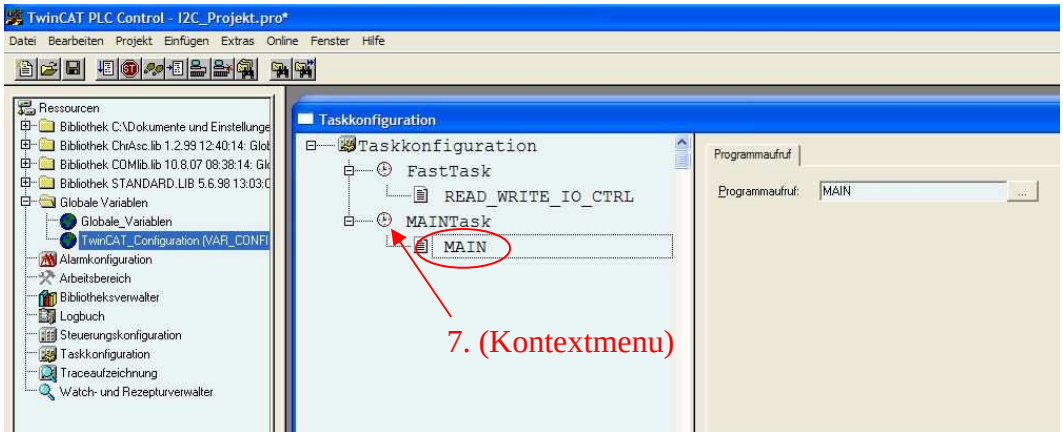


Bild 43:

4.8.5 Vorbereitung für erstes fehlerfreies Übersetzen des I2C_Projekt

Um das Übersetzen erfolgreich durchführen zu können, muss das MAIN-Programm eine minimale Anweisung enthalten. Diese minimale Anweisung ist ein “;” (Semikolon).

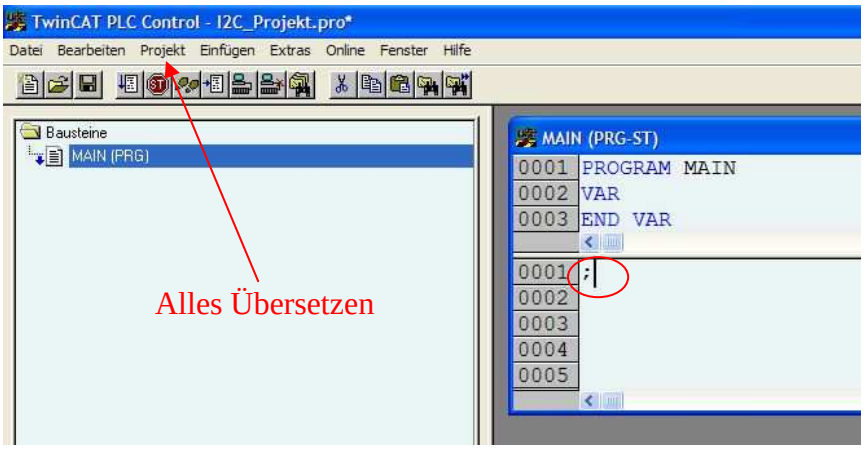


Bild 44:

Nach dem Ausführen der Übersetzung, zu starten unter dem Menu “Projekt\Alles Übersetzen”, zeigen sich im Meldefenster die folgenden Meldungen mit 0 Fehler und 4 Warnungen. Dieser Zustand ist zu diesem Zeitpunkt in Ordnung.

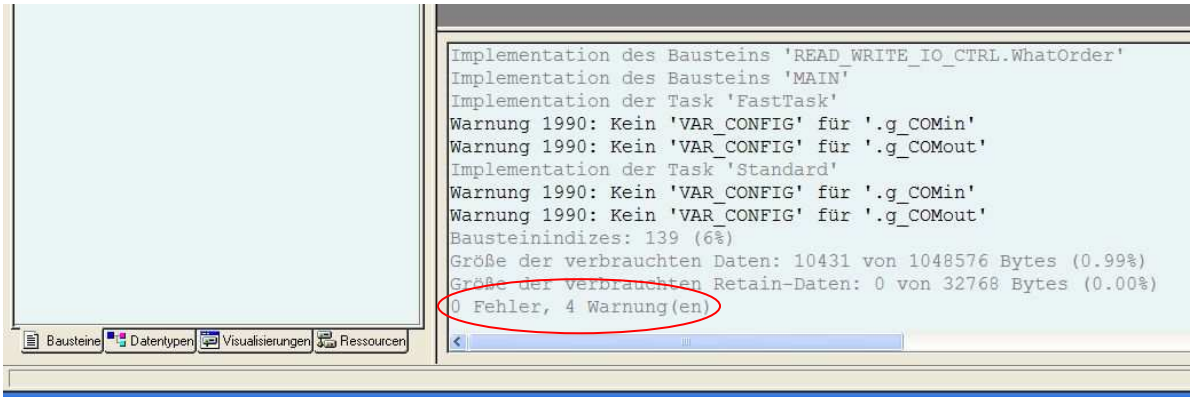


Bild 45:

4.8.6 Hardware mit Software verknüpfen, per System Manager

Die RS232-Schnittstelle vom PC muss noch mit der Software gekoppelt werden, dafür braucht es ein Konfigurationsfile. Dies wird mit dem System Manager von TwinCat erledigt.

1. Rechter Mausklick, über dem TwinCat-Icon von der Taskleiste, am rechten Bildschirmecken.
2. Linker Mausklick auf die Auswahl “System Manager”,

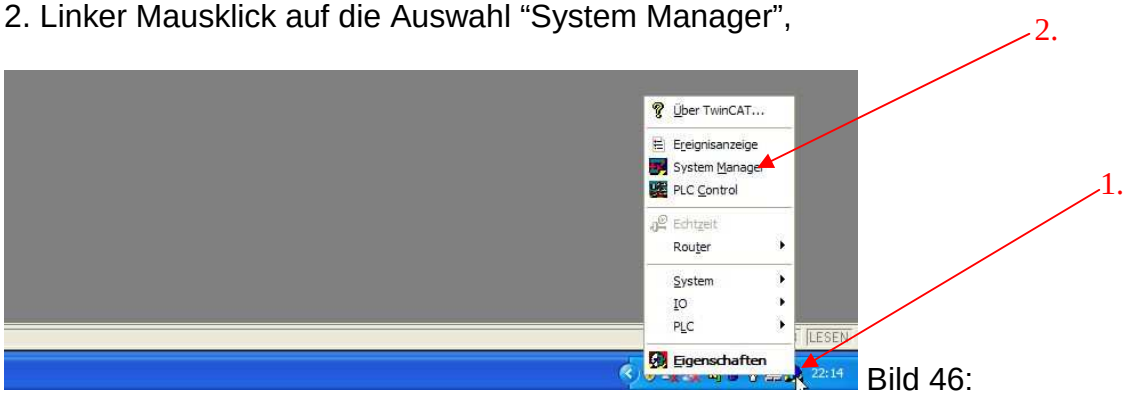


Bild 46:

dann öffnet sich der TwinCat System Manager.

3. Unter Datei\Neu wird eine neue Konfiguration erstellt, die unter einem neuen Namen “Datei\Speichern unter...” gespeichert werden soll,
   z. Bsp. I2C_ProjektKonf.tsm.

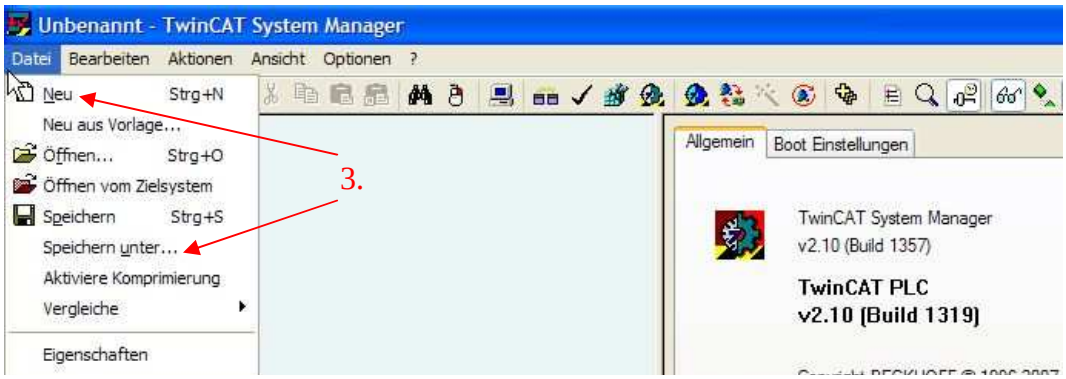


Bild 47:

4. Die Basiszeit, unter Echtzeit-Einstellungen, ist auf **100µs** einzustellen.

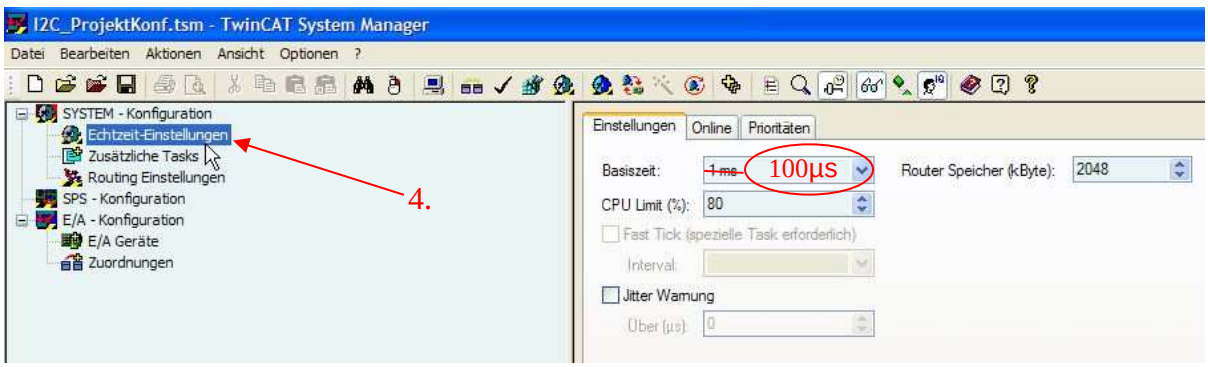


Bild 48:

5. Mit einem Klick auf die SPS-Konfiguration erreicht man das Register SPS-Einstellungen, das mindestens wie folgt eingestellt sein sollte:

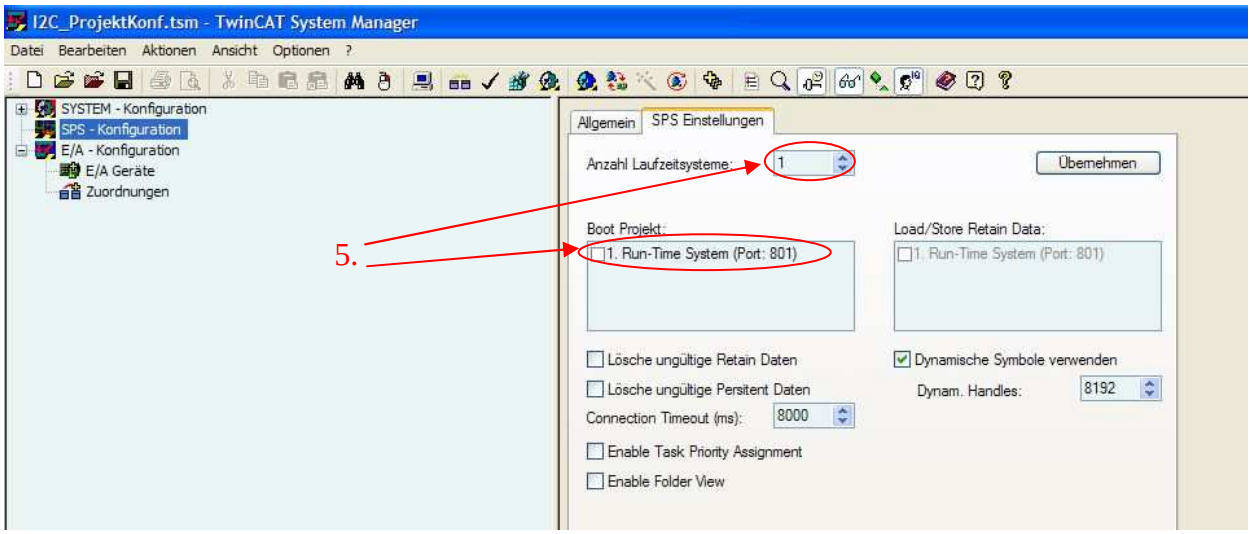


Bild 49:

Das Boot Projekt muss nur angewählt werden, wenn beim Starten der SPS das Projekt automatisch starten soll. Mit der Anzahl Laufzeitsysteme, wird angegeben wieviele SPS'en in Betrieb sein sollen.

6. Der SPS-Konfiguration muss das SPS-Projekt angefügt werden. Per Kontextmenu (rechte Maustaste) über der SPS-Konfiguration, lässt sich dies mit der Auswahl "SPS Projket Anfügen..." erledigen.
   Die Voraussetzung dafür ist aber, dass zuvor das SPS-Projekt fehlerfrei übersetzt werden konnte. Nur dann wird nämlich ein Filetyp mit .tpy auswählbar sein.

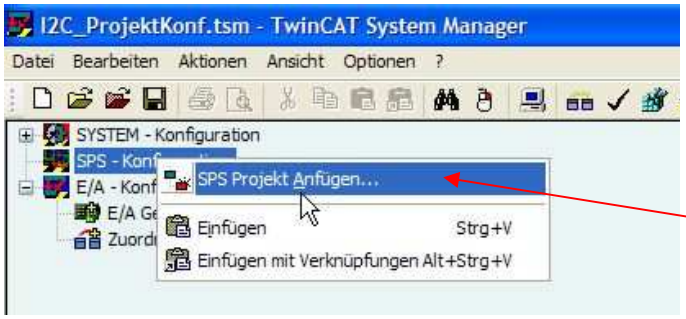


Bild 50:

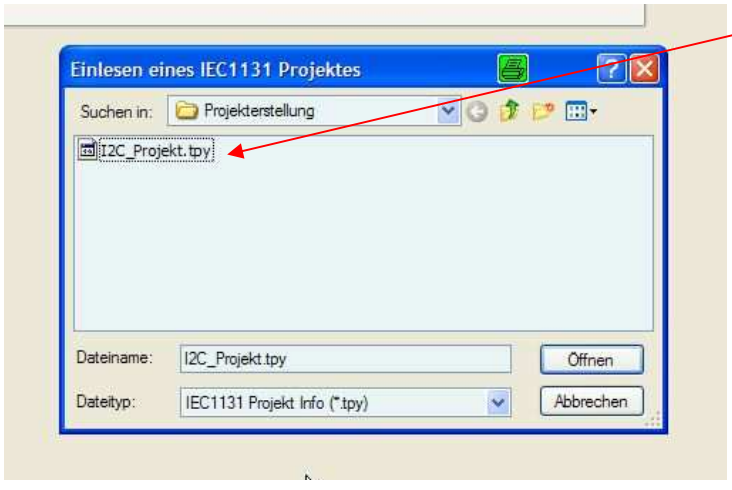


Bild 51:

Nun steht der SPS-Konfiguration die Softwarevariablen g_COMin und g_COMout zur Verfügung, siehe unten.

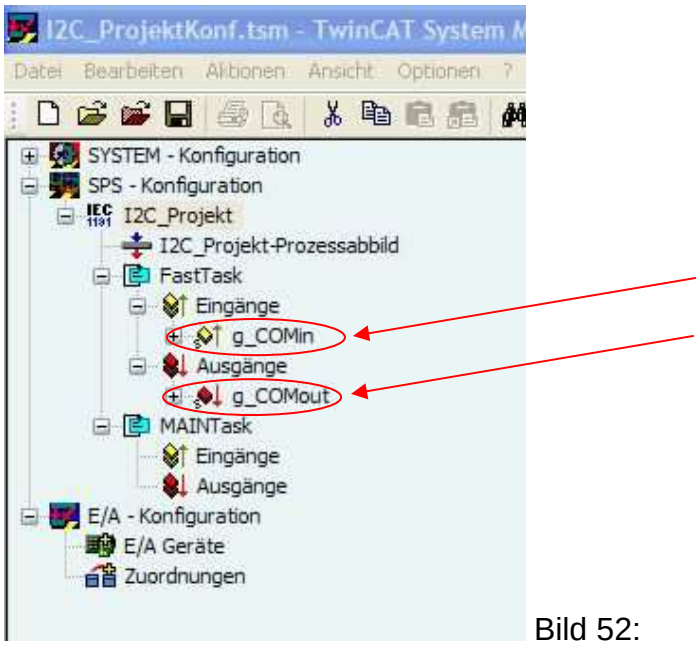


Bild 52:

7. Der E/A-Konfiguration muss die Serielle-Schnittstelle eingefügt werden, so dass dem Projekt die Hardware zu Verfügung steht. Per Kontextmenu (rechte Maustaste) über dem E/A Geräte, öffnet den Dialog “Einfügen eines E/A Geräts”.

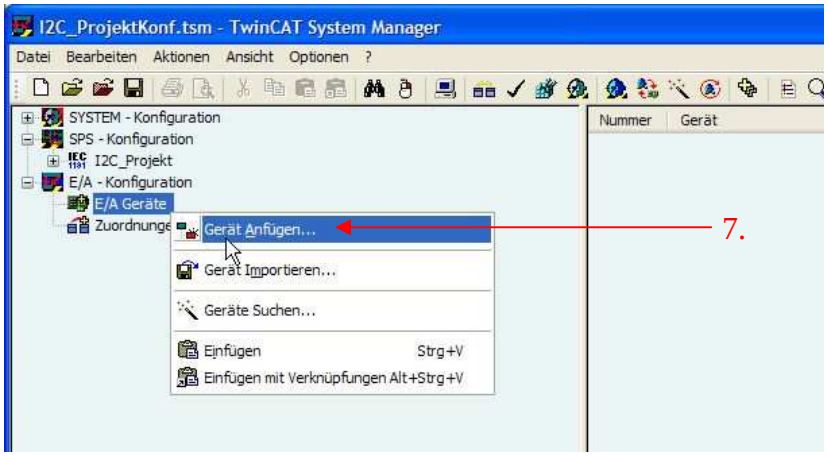


Bild 53:

8. Im Dialog “Einfügen eines E/A Geräts” ist unter dem Kapitel “Verschiedenes” die Serielle – Schnittstelle anzuwählen und mit OK zu bestätigen.

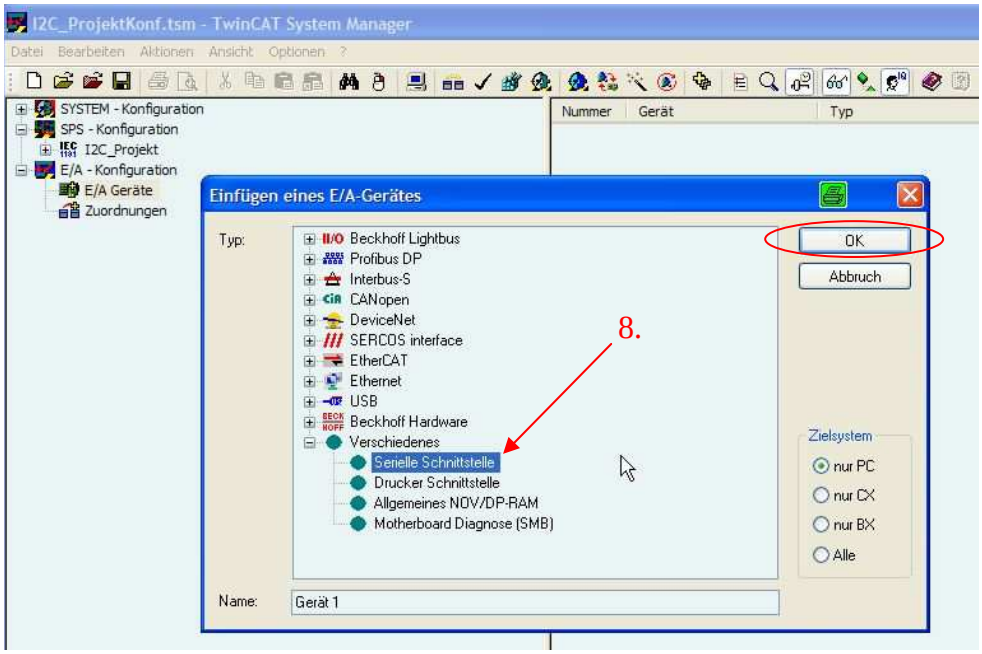


Bild 54:

9. Danach öffnet sich der Dialog “Gerät an Adresse gefunden”, der einfach mit “Abbruch” zu schliessen ist.

10. Es sind nun einige Einstellungen im Register “Serieller Port” und “Kommunikations Einstellungen” vorzunehmen, die unter dem Gerät 1 (Com Port) zu finden sind.

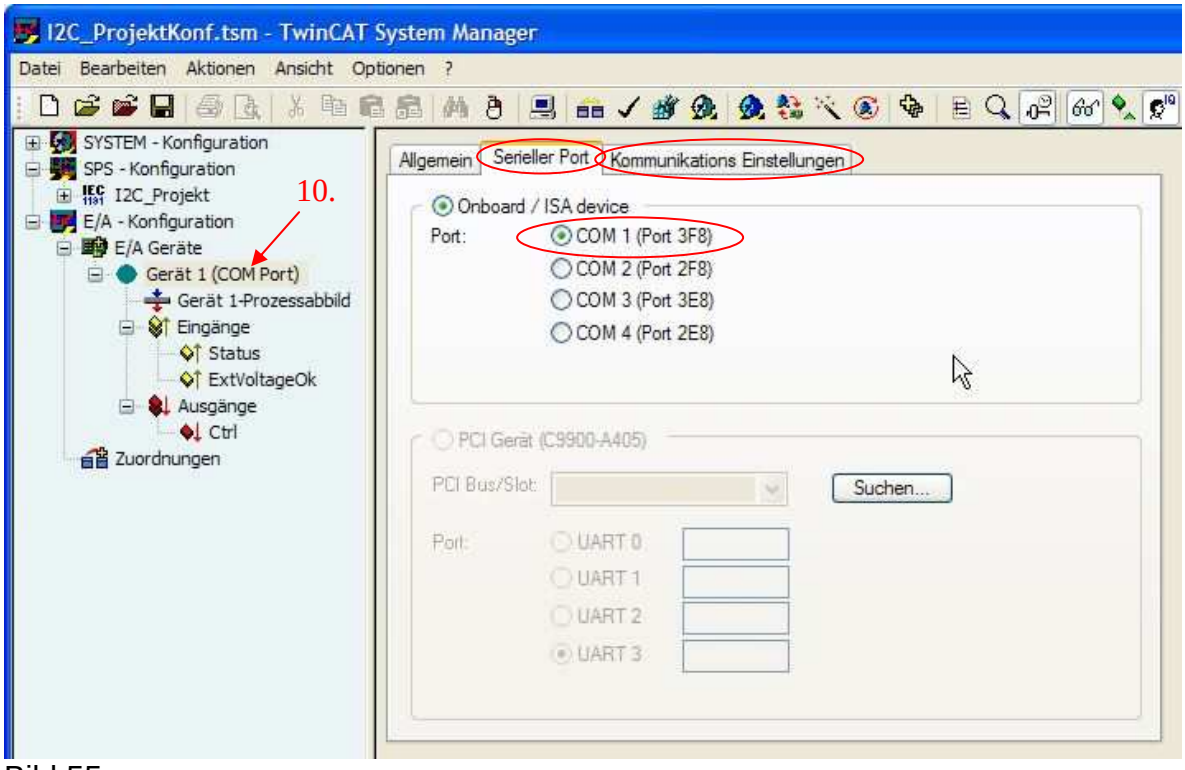


Bild 55:

11. Im Register “Kommunikations Einstellungen” sind die Kommunikations-konfigurationen so einzustellen, dass es mit dem I2C-Modem übereinstimmt, siehe Bild 56, die rot umkreisten Felder.

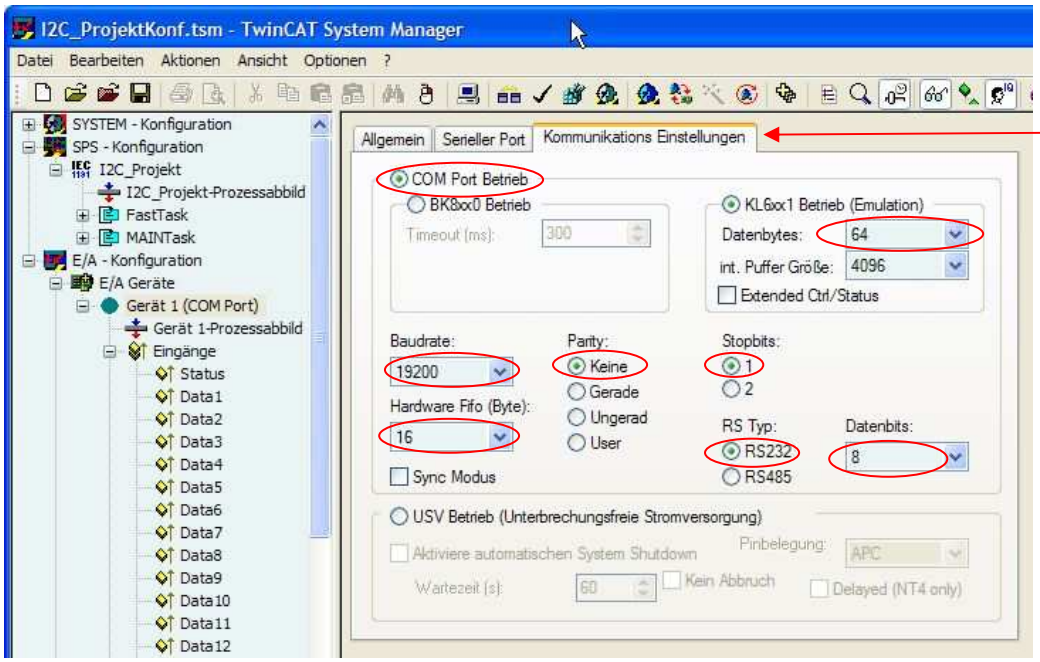


Bild 56:

Nach diesen Einstellungen wurden die Empfangs- (Eingänge) und Sendebuffer (Ausgänge) des Geräte 1 (Com1 Port) dargestellt.

12. Jetzt gilt es die Eingänge und Ausgänge vom Gerät 1 mit der SPS zu verknüpfen. Am einfachsten geht es, wenn man auf die einzelnen Eingänge vom Gerät 1, hier zum Bsp. auf “Status”, einen Doppelklick ausführt. Dann öffnet sich der Dialog “Variablenverknüpfung” und zeigt einem die SPS-Variable, die vom Typ her zu verknüpfen wäre.

13. Der Programmierer kann nun auf die passende SPS-Variable doppelklicken, was gerade eine Verknüpfung ausführt. Eine Verknüpfung wird speziell mit einem kleinen Pfeil gekennzeichnet, siehe Bild 58.

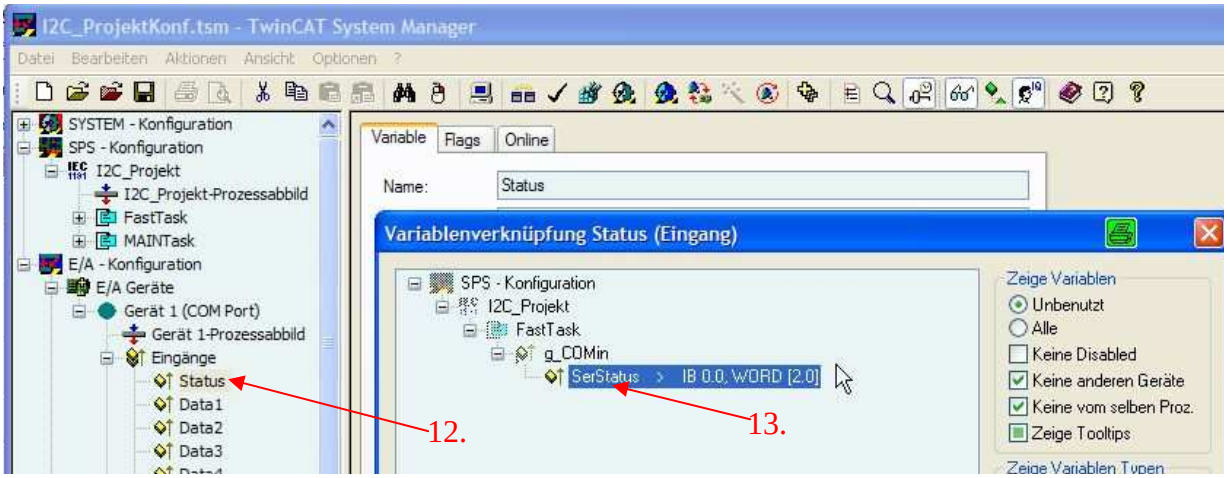


Bild 57:

14. Alle Eingänge (65) und alle Ausgänge (65) vom Gerät 1, sind mit den SPS-Variablen zu verknüpfen. Ab Data1 – Data64 werden mehrere SPS-Variablen vom gleichen Typ vorgeschlagen. Die Verknüpfung muss der Reihe nach erfolgen, Data1 verknüpfen mit D[0], Data2 verknüpfen mit D[1] usw..

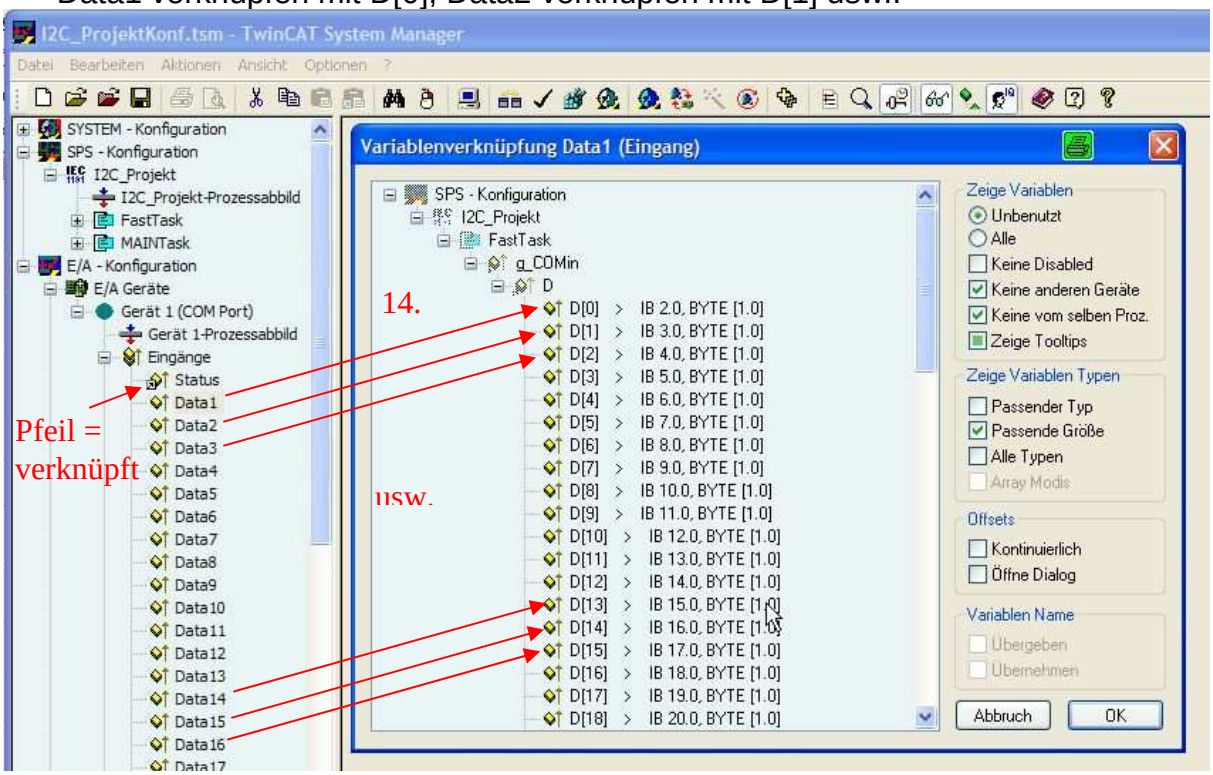


Bild 58:

15. Sind alle Verknüpfungen erstellt worden, muss die Zuordnung erfolgen. Dies wird erledigt, wenn per Kontextmenu über “Zuordnungen” das Feld “Zuordnung Erzeugen” angewählt wird.

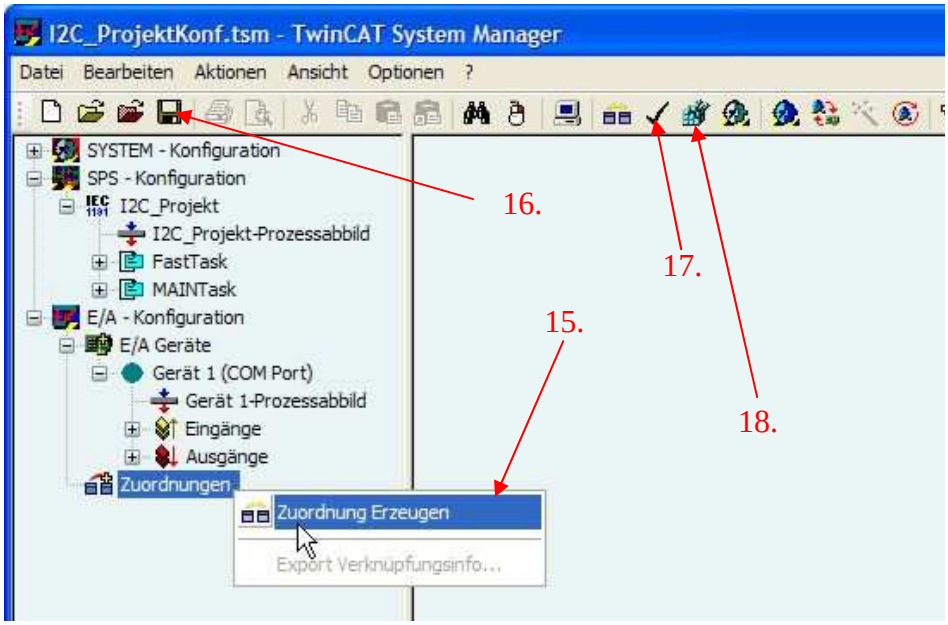


Bild 59:

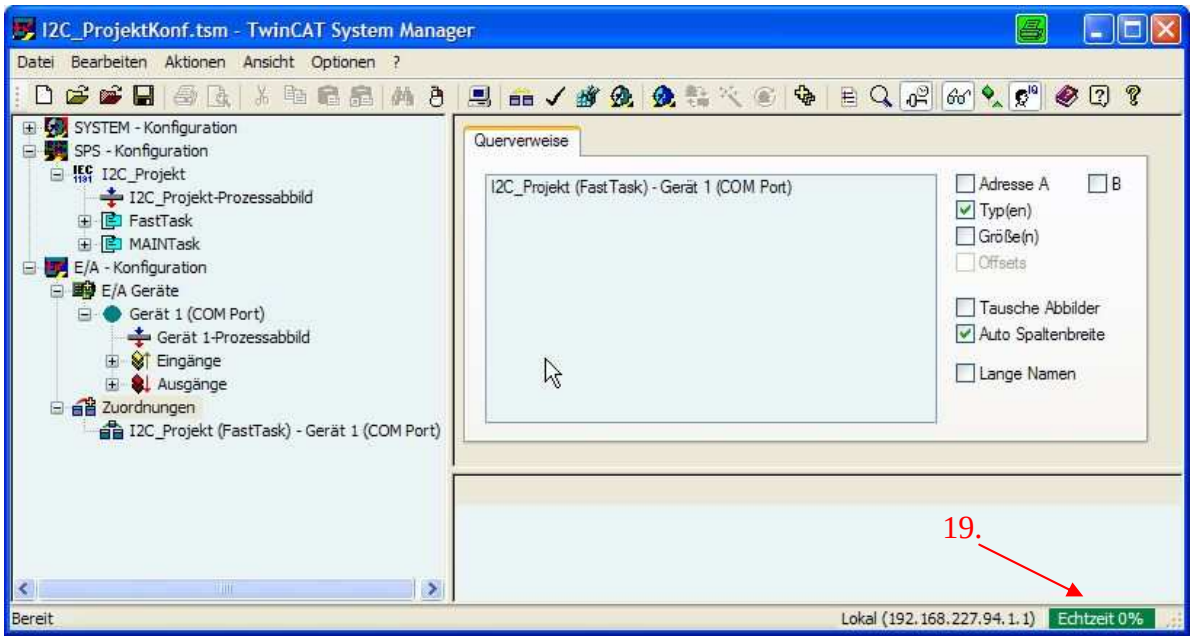
16. Spätestens jetzt sollte die Konfiguration erneut gespeichert werden.

17. Die Konfiguration muss manuell mit dem Feld “Häckchen” überprüft werden.

18. Um die Soft-SPS auf dem Notebook, für einen Programmdownload bereit zu stellen, muss die neu erstellte Konfiguration aktiviert werden. Diese aktiviert man mit dem Würfel. Der Dialog mit der Frage “? Neustart TwinCAT System in Run Modus” mit OK bestätigen.

19. Die Soft-SPS ist für den Programmdownload bereit, wenn das Feld Echtzeit %0 in grüner Farbe erscheint.

Bild 60:



4.8.8 Beispielprogramm ins I2C_Projekt importieren, übersetzen und downloaden

Im “TwinCAT PLC Control” ist nun noch das Anwenderprogramm zu erstellen.
Wir öffnen das I2C_Projekt, das unter 4.8.1 erstellt wurde.

1. Das Beispielprogramm steht als “MAIN.exp” File zum Import bereit. Der Import wird über das Menu Projekt\Importieren..., nach der Fileauswahl erfolgen.
   Verzeichnis: I2C_Projekt
   Es erscheint aber vorerst ein Dialog mit der Frage
   “Das Objekt MAIN existiert bereits. Wollen Sie es ersetzen?”.
   Beantworten Sie diese mit “Ja, Alle”.

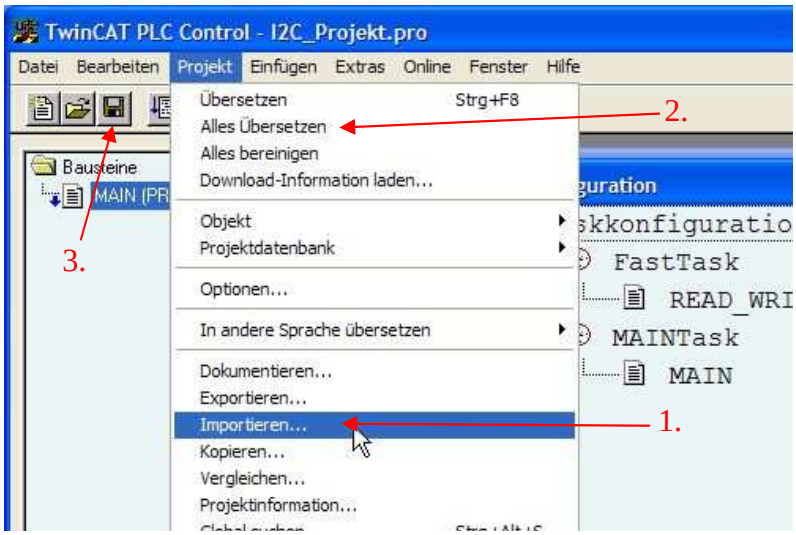


Bild 61:

2. Das Übersetzen ist unter dem Menu Projekt\Alles Übersetzen zu starten.

3. Mit dem Icon Diskette, muss das Projekt erneut gespeichert werden.

Nach erfolgreicher Übersetzung, mit null Fehlern und null Warnungen, muss das Programm noch in die Soft-SPS heruntergeladen werden.

4. Der Download erfolgt mit dem unten markierten Icon (F11) und dem nachträglichen Starten (F5) des Programmes.

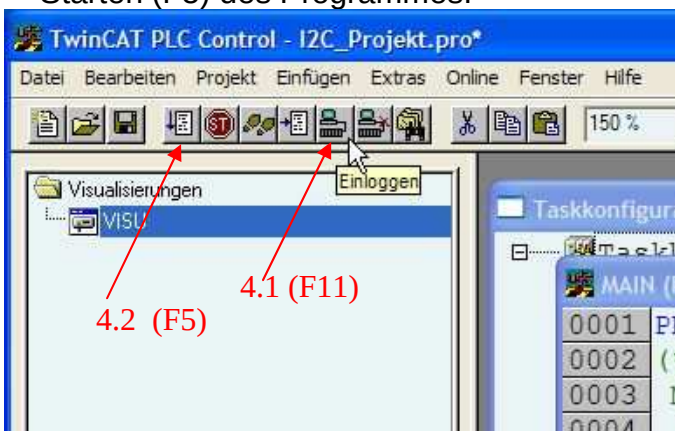


Bild 62:

Ist dann auch noch das I2C-Modem an die RS232 des PC’s oder des Notebooks angeschlossen, so müsste die Soft-SPS mit den I2C-Karten kommunizieren.
!!Speisung einschalten**!!**

4.8.9 Bedienung des Beispielprogramms

1. Mit der VISU, die unter der Registerkarte “Visualisierung” zu finden ist, kann die Testumgebung bedient werden.
2. Der Doppelklick auf VISU, öffnet dann auch das VISU-Fenster.

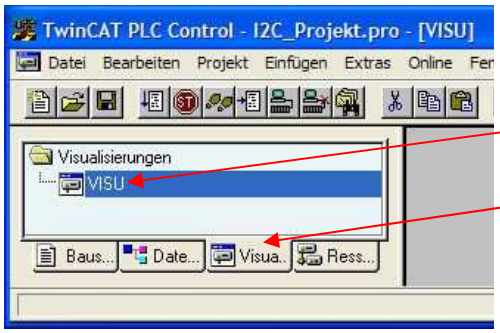


Bild 63:

Die Visualisierung
der Testumgebung, mit Beispielprogrammen präsentiert sich wie folgt:

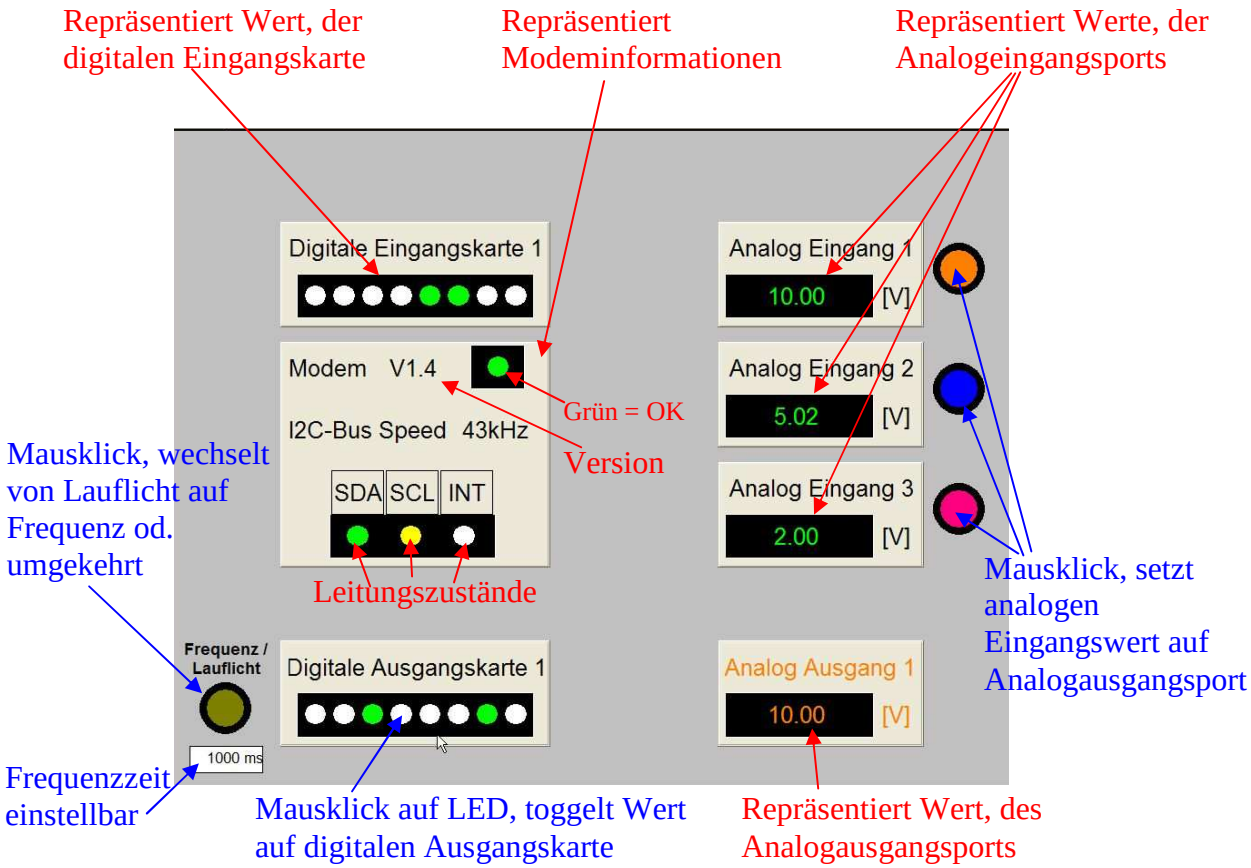
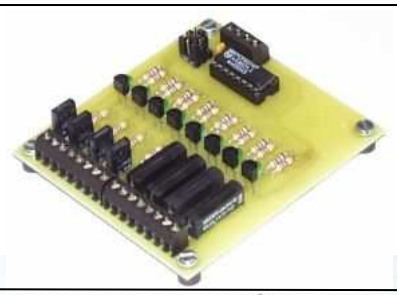


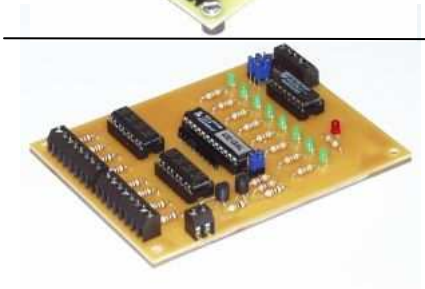
Bild 64:

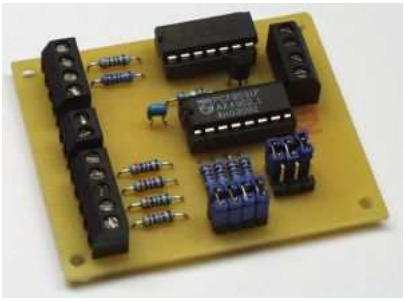
4.8.10 Erläuterungen zum Programmcode

Im MAIN-Programm, ab der Zeilennummer 43, findet man diese Codezeilen.
Hier wird anhand der vorhanden I2C-Peripherie, der ARRAY “g_aCommunication” abgefüllt und zugleich wieder abgefragt.

Vom Anwenderprogramm übergebene Daten sind rot gekennzeichnet.
Vom Anwenderprogramm entgegengenommene Daten sind blau gekennzeichnet.
Die restlichen Einträge vom und zum ARRAY “g_aCommunication”, sind für die Peripheriekonfiguration.







```
(*****************************************************************
    Unter den Feldern 1-12 von g_aCommunication
    werden I2C-Pheripherie Anfragen definiert.
    Jede digitale Ein- od. Ausgangskarte bekommt einen
    ARRAY-Platz aus g_aCommunication. Die analoge
    Ein-Ausgabekarte besetzt hingegen vier ARRAY-Plätze,
    da diese zu lesende, zu schreibende und zu konfigurierende
    Daten hat.
*****************************************************************)

(*Dig. Ausgabekarte 8 Bit*)
   g_aCommunication[1].i_abDaten[1] := bDigAusgabe1;
   g_aCommunication[1].i_bAnzByte := 1;       (*Anzahl Byte*)
   g_aCommunication[1].i_bModemCmd := g_WRITE; (*Modembefehl*)
   g_aCommunication[1].i_bSlaveAddress := 64;  (*Kartenadresse*)
   (*Mögliche Adressen 64, 66, 68, 70, 72, 74, 76, 78*)

(*Dig.Eingabekarte 8 Bit*)
   g_aCommunication[2].i_bAnzByte := 1;       (*Anzahl Byte*)
   g_aCommunication[2].i_bModemCmd := g_READ; (*Modembefehl*)
   g_aCommunication[2].i_bSlaveAddress := 67; (*Kartenadresse*)
   (*Mögliche Adressen 65, 67, 69, 71, 73, 75, 77, 79*)
   bDigEingabe1 := g_aCommunication[2].o_abDaten[1];

(*Analoge Karte Betriebsart setzen*)
   (*Kanal einstellen, -> 4 = alle analoge Eingaenge*)
   g_aCommunication[3].i_abDaten[1] := 64+4;
   g_aCommunication[3].i_bAnzByte := 1;       (*Anzahl Byte*)
   g_aCommunication[3].i_bModemCmd := g_WRITE;(*Modembefehl*)
   g_aCommunication[3].i_bSlaveAddress := 144;(*Kartenadresse*)
   (*Mögliche Adressen 144, 146, 148, 150, 152, 154, 156, 158*)

   (*Analoge Eingabekarte 4 Kanäle a 8 Bit, Altwerte verwerfen*)
   g_aCommunication[4].i_bAnzByte := 1;       (*Anzahl Byte*)
   g_aCommunication[4].i_bModemCmd := g_READ;   (*Modembefehl*)
   g_aCommunication[4].i_bSlaveAddress := 144+1;(*Kartenadresse*)
   (*Mögliche Adressen 145, 147, 149, 151, 153, 155, 157, 159*)

   (*Analoge Eingabekarte 4 Kanäle a 8 Bit lesen*)
   g_aCommunication[5].i_bAnzByte := 4;       (*Anzahl Byte*)
   g_aCommunication[5].i_bModemCmd := g_READ;   (*Modembefehl*)
   g_aCommunication[5].i_bSlaveAddress := 144+1;(*Kartenadresse*)
   (*Mögliche Adressen 145, 147, 149, 151, 153, 155, 157, 159*)
   rACEingang1 := g_aCommunication[5].o_abDaten[1] * 0.0392;
   rACEingang2 := g_aCommunication[5].o_abDaten[2] * 0.0392;
   rACEingang3 := g_aCommunication[5].o_abDaten[3] * 0.0392;

   (*Analoge Ausgabekarte 1 Kanal 8 Bit schreiben*)
   (*10V : 255 = 0.0392*)
   bDigAusgabe1 := REAL_TO_BYTE((rACAusgang1 / 0.0392));
   g_aCommunication[6].i_abDaten[1] := 64;(*AD-Wandler freigeben*)
   (*analoger Ausgabewert*)
   g_aCommunication[6].i_abDaten[2] := bACAusgang1;
   g_aCommunication[6].i_bAnzByte := 2;       (*Anzahl Byte*)
   g_aCommunication[6].i_bModemCmd := g_WRITE; (*Modembefehl*)
   g_aCommunication[6].i_bSlaveAddress := 144; (*Kartenadresse*)
```

Wie man also feststellt, benötigt die I2C-Analogkarte 4 ARRAY Plätze.

Wie kommt dies zustande?

Wenn man das Datenblatt vom PCF8591P zur Hand nimmt, erfahrt man, wie der IC aufgebaut ist und wie damit kommuniziert werden soll.

Es wird beschrieben, dass die Adresse als erstes Byte gesendet werden soll.
Das letzte Bit der Adresse kennzeichnet ob gelesen oder geschrieben wird.
Gelesen wird, wenn das letzte Bit der Adresse 1 ist.
In dem Programmcode wird das Lesen mit der Adressübergabe 144 + 1 erledigt.

Das zweite zu sendende Byte, wird im Controlregister gespeichert und bestimmt die Gerätefunktion. Mit 64 +4, im Programmcode, wird nun die Gerätefunktion wie folgt gelegt, siehe Datenblatt Fig. 4,:

- Analogausgang wird freigegeben
- Analogeingänge werden als vier einzelne Eingänge definiert
- Das Flag Autoincrement wurde auch gesetzt, was ein Inkrementieren des Kanals bei Wertänderung veranlässt.

Für den Analogausgang muss ein weiteres Byte gesendet werden, nämlich den digitalen Wert, der in ein Analogsignal gewandelt werden soll. Dieser wird im DAC Dataregister gespeichert. Im Programmcode wird dies mit der Variable “bDigAusgabe1” ausgeführt.

Das Erfassen der Analogeingänge wird mit einer gültigen Leseadresse (144+1) gestartet. Was ein nachfolgendes Senden der Digitalwerte, gewandelt aus den einzelnen Analogwerten, auslöst. Diese werden dann, im Programmcode, in den Variablen rACEingang1-n eingetragen.

4.9 Verbesserungsvorschläge

Nach den obigen Tests ist es relativ eindeudig, dass das dynamische Verhalten dieses Kommunikationssystems nicht beeindruckend ist. Man müsste mit der Antwortzeit mindestens 10 mal schneller werden.

Am einfachsten könnte man dies erreichen, wenn das TwinCAT System die Serielle-Schnittstelle auch mit kleineren Taskzeiten, als nur einer Millisekunde, treiben könnte.

Weiter wäre es zu überlegen, ob das Pollingsystem (zyklisches Anfragen) durch ein Interruptsystem ersetzt werden sollte. Das würde heissen, dass nur die Interruptleitung ständig angefragt und die Daten für die Ausgangskarten nur bei Änderung gesendet würden. Beim Erfassen des Interrupts, müssten dann nur alle Eingangskarten angefragt werden. Dies würde den Vorteil bringen, dass die Durchschnittliche Antwortzeit gesenkt werden könnte. Der Nachteil dabei wäre, man kann nicht erkennen, zu welchem Zeitpunkt die höchste Antwortzeit erfolgt, der so sogenannte Ausreisser, zum ungünstigsten Moment.

Das I2C-Modem hätte natürlich auch noch Potential. Z. Bsp. könnte man dieses so aufbauen, dass es alle I2C-Eingangskarten selbst anfragt und beim Erkennen einer Änderung das ganze Abbild aller Karten überträgt. Dies bedeutet natürlich, dass das I2C-Modem die I2C-Topologie kennen muss. Die TwinCAT Seriell-Schnittstelle kann pro drei Zyklen, 64 Byte empfangen und versenden. Bei einem 1ms PLC-Taskzyklus könnte dann innerhalb 3ms alle I2C-Karten bedient und angefragt werden.

4.10 Rückblick mit Erkenntnissen

Diese Aufgabe hat mir wieder einmal gezeigt, dass sich trotz vorgängiger Analyse und Konzepterstellung noch Unbekannte verbergen können. Vielleicht hätte man noch genauer analysieren müssen, aber der Zeitdruck drängt einem weitere Schritte anzugehen, den der Endtermin ist ja nicht zu verschieben.

Mit der einen Unbekannten meine ich, das zeitliche Verhalten der Datenübertragung, die ich durch mangelnde Erfahrung in der Kommunikationsprogrammierung unterschätzt habe. Die Abschwächung der Problemmatik ist unter anderm auch daher gekommen, dass die I2C-Buszeiten mit 43kHz und die Modemzeiten von 19200Baud ziemlich schnell sind.

Dann habe ich mich noch mit einer Unbekannten herumgeschlagen, die dem TwinCAT PLC-System zuzuschreiben war. Die Timerfunktionen waren ca. 4 mal langsamer. Da hatte ich einen Timer von 1000ms definiert, aber er brauchte vier Sekunden bis er abgelaufen war. Nach unzähligen Versuchen dieses Problem zu lösen, ist es mir doch noch gelungen, die Nadel im Heuhaufen zu finden. Erst als die Basiszeit der SPS-Konfiguration auf 100µs gestellt hatte, funktionierte der Timer wieder, wie man es auch erwarten würde.

Rückblickend kann ich aber sagen, dass das Konzept der Anbindung von I2C-Karten in die Soft-SPS aus Entwicklersicht einfach gehalten ist. Wie das die Anwender empfinden, wird sich noch zeigen, aber da bin ich zuversichtlich.

Die Variante mit dem I2C-Modem ist doch für einfache Heimanwendungen und evt. auch für Lehrlinge eine günstige Alternative. Besonders dann, wenn das TwinCAT-System als Studentenversion zur Verfügung steht und damit die Programmiersoftwarekosten wegfallen.

5 Anhang

5.6 Quellenverzeichnis

- Beckhoff
  TwinCAT Information System -> Hilfe von Entwicklungsumgebung
- Beckhoff Online Support -> http://www.beckhoff.de/
- Horter & Kalb -> http://www.horter.de/
- Philips Datenblätter -> siehe Anhang
- Internet -> http://www.google.ch/

5.7 Programmlisting

5.7.2 Hauptprogramm

```
(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '' *)
(* @SYMFILEFLAGS := '2048' *)
PROGRAM MAIN (*MAINTask, Prioritaet 1, Zyklisch 10ms*)
(*****************************************************************
 MAIN Programm

 Funktion: Beispiel fuer die Anbindung von
        I2C-Komponenten ( www.Horter.de )

 Hardware: Fuer dieses Beispiel sind mindestens folgende
        Komponenten an das I2C-Modem zu schalten:
        - digitale Eingangskarte, I2C-Bus Adresse 65
        - digitale Ausgangskarte, I2C-Bus Adresse 64
        - analoge Ein-Ausgangskarte, I2C-Bus Adresse 144

 Autor:    R. Lichtensteiger

 *****************************************************************)
VAR
  rACEingang1:  REAL;  (*siehe VISU, Analogeingang*)
  rACEingang2:  REAL;  (*siehe VISU, Analogeingang*)
  rACEingang3:  REAL;  (*siehe VISU, Analogeingang*)
  rACAusgang1:  REAL;  (*siehe VISU, Analogausgang*)
  prACAusgang1: POINTER TO REAL; (*Zeigervariable*)
  bACAusgang1:  BYTE;  (*Analogausgang in Byte*)
  bDigAusgabe1: BYTE;  (*siehe VISU, Digitalausgang*)
  bDigAusgabe1Last: BYTE;(*letzter Wert*)
  bDigEingabe1: BYTE;  (*siehe VISU, Digitaleingang*)
  strSpeed:     STRING;(*siehe VISU, Speedwert*)

  t_LEDwechsel: TON; (*Timer Lauflicht*)
  t_Frequenz:   TON; (*Timer Frequenz*)
  wFrequenz:    WORD := 1000; (*default Zeitbasis ms*)
  boLauflicht:  BOOL;  (*siehe VISU, Schalter ein / aus*)
  bACInput:     BYTE := 1;(*siehe VISU, Schalter AC-Input Wechsel*)
  bACInputLast: BYTE;  (*siehe SelectACInput*)
  dwFarbeRGB:   DWORD; (*siehe SelectACInput*)
END_VAR
(* @END_DECLARATION := '0' *)
(*Die hier vorgegebene Struktur muss eingehalten werden,
  wenn mit der I2C-Library die I2C-Karten von Horter & Kalb
  gesteuert werden sollen.*)

(*******Schreiben Sie hier Ihre I2C Applikation******************)

Lauflicht();     (*Lauflicht erscheint auf digitaler Ausgabekarte*)
SelectACInput(); (*definiert welcher AC Eingang der AC-Karte
                   auf den AC Ausgang geschalten werden soll*)
SetFrequenz();   (*gibt eine Frequenz auf Bit 0 der
                   digitalen Ausgabekarte aus, wenn Lauflicht aus-
                   geschalten ist*)

(*Die gleich unten stehenden Variablen, werden von den
  obigen Aktionen verwendet.

  (*In diese Variable schreibt das Lauflicht den Wert
```

```
    der auf der digitalen Ausgangskarte erscheinen soll.*)

  Variable: bDigAusgabe1

  (*Von dieser Variable bekommt die Visualisierung den darzu-
    stellenden Wert, der auf der digitalen Eingangskarte ansteht.*)

  Variable: bDigEingabe1

  (*Von dieser Variable bekommt die Visualisierung die darzu-
    stellenden Analogwerte, die an den analogen Eingangsports
    anstehen.
    PS. Es koennten vier Eingangsports angefragt werden.*)

  Variable: rACEingang1
  Variable: rACEingang2
  Variable: rACEingang3

  (*In diese Variable schreibt die Aktion SelectACInput den Wert,
    der auf dem analogen Ausgangsport erscheinen soll.*)

  Variable: bACAusgang1

Siehe unten, da sehen Sie wie die Variablen verwendet werden*)

(*****************************************************************
     Unter den Feldern 1-12 von g_aCommunication
     werden I2C-Peripherie Anfragen definiert.
     Jede digitale Ein- od. Ausgangskarte bekommt einen
     ARRAY-Platz aus g_aCommunication. Die analoge
     Ein-Ausgabekarte besetzt hingegen vier ARRAY-Plätze,
     da diese zu lesende, zu schreibende und zu konfigurierende
     Daten hat.
*****************************************************************)

(*Dig. Ausgabekarte 8 Bit*)
   g_aCommunication[1].i_abDaten[1] := bDigAusgabe1;
   g_aCommunication[1].i_bAnzByte := 1;       (*Anzahl Byte*)
   g_aCommunication[1].i_bModemCmd := g_WRITE; (*Modembefehl*)
   g_aCommunication[1].i_bSlaveAddress := 64;  (*Kartenadresse*)
   (*Mögliche Adressen 64, 66, 68, 70, 72, 74, 76, 78*)

(*Dig.Eingabekarte 8 Bit*)
   g_aCommunication[2].i_bAnzByte := 1;      (*Anzahl Byte*)
   g_aCommunication[2].i_bModemCmd := g_READ; (*Modembefehl*)
   g_aCommunication[2].i_bSlaveAddress := 67; (*Kartenadresse*)
   (*Mögliche Adressen 65, 67, 69, 71, 73, 75, 77, 79*)
   bDigEingabe1 := g_aCommunication[2].o_abDaten[1];

(*Analoge Karte Betriebsart setzen*)
   (*Kanal einstellen, -> 4 = alle analoge Eingaenge*)
   g_aCommunication[3].i_abDaten[1] := 64+4;
   g_aCommunication[3].i_bAnzByte := 1;      (*Anzahl Byte*)
   g_aCommunication[3].i_bModemCmd := g_WRITE;(*Modembefehl*)
   g_aCommunication[3].i_bSlaveAddress := 144;(*Kartenadresse*)
   (*Mögliche Adressen 144, 146, 148, 150, 152, 154, 156, 158*)

   (*Analoge Eingabekarte 4 Kanäle a 8 Bit, Altwerte verwerfen*)
   g_aCommunication[4].i_bAnzByte := 1;       (*Anzahl Byte*)
   g_aCommunication[4].i_bModemCmd := g_READ;   (*Modembefehl*)
   g_aCommunication[4].i_bSlaveAddress := 144+1;(*Kartenadresse*)
   (*Mögliche Adressen 145, 147, 149, 151, 153, 155, 157, 159*)
```

```
   (*Analoge Eingabekarte 4 Kanäle a 8 Bit lesen*)
   g_aCommunication[5].i_bAnzByte := 4;        (*Anzahl Byte*)
   g_aCommunication[5].i_bModemCmd := g_READ;   (*Modembefehl*)
   g_aCommunication[5].i_bSlaveAddress := 144+1;(*Kartenadresse*)
   (*Mögliche Adressen 145, 147, 149, 151, 153, 155, 157, 159*)
   rACEingang1 := g_aCommunication[5].o_abDaten[1] * 0.0392;
   rACEingang2 := g_aCommunication[5].o_abDaten[2] * 0.0392;
   rACEingang3 := g_aCommunication[5].o_abDaten[3] * 0.0392;

   (*Analoge Ausgabekarte 1 Kanal 8 Bit schreiben*)
   (*10V : 255 = 0.0392*)
   bACAusgang1 := REAL_TO_BYTE((rACAusgang1 / 0.0392));
   g_aCommunication[6].i_abDaten[1] := 64;(*AD-Wandler freigeben*)
   (*analoger Ausgabewert*)
   g_aCommunication[6].i_abDaten[2] := bACAusgang1;
   g_aCommunication[6].i_bAnzByte := 2;        (*Anzahl Byte*)
   g_aCommunication[6].i_bModemCmd := g_WRITE; (*Modembefehl*)
   g_aCommunication[6].i_bSlaveAddress := 144; (*Kartenadresse*)

(*weitere Karten *)
   (*     .
          .
          .
          .                                *)
   (*bis 12 I/O Karten können definiert werden*)

   (* BSP:
   g_aCommunication[7].i_abDaten  :=
   g_aCommunication[7].i_bAnzByte  :=
   g_aCommunication[7].i_bModemCmd :=
   g_aCommunication[7].i_bSlaveAddress :=
          .
          .
          .
   g_aCommunication[12].i_abDaten  :=
   g_aCommunication[12].i_bAnzByte  :=
   g_aCommunication[12].i_bModemCmd :=
   g_aCommunication[12].i_bSlaveAddress :=

Ende bei 12 Karten oder Modembefehlen *)

(*Unter dem Feld 13 werden I2C-Bus Speed Daten abgelegt*)
(*Uebergeben Sie hier den entsperechenden Speedwert*)
  g_aCommunication[13].i_bModemCmd := g_SPEED_43KHz;
  (************************************************)
    CASE g_aCommunication[13].i_bModemCmd OF
     g_SPEED_43KHz: strSpeed := '43kHz';(*Defaultwert in Library*)
     g_SPEED_28KHz: strSpeed := '28kHz';
     g_SPEED_17KHz: strSpeed := '17kHz';
     g_SPEED_9KHz:  strSpeed := '9kHz';
     g_SPEED_5KHz:  strSpeed := '5kHz';
     g_SPEED_2500Hz: strSpeed := '2,5kHz';
     g_SPEED_1300Hz: strSpeed := '1,3kHz';
    END_CASE

(*Die Modemdaten werden von der Visualisierung dargestellt. Diese
  entnimmt die Daten wie folgt:
 strSpeed                    (*I2C-Bus Speed*)
 g_aCommunication[14].o_boOK      (*Modem vorhanden bei TRUE*)
 g_aCommunication[15].o_boSCL     (*Modem Taktleitung*)
 g_aCommunication[15].o_boINT     (*Modem Interruptleitung*)
```

```
 g_aCommunication[15].o_boSDA      (*Modem Datenleitung*)
 g_aCommunication[16].o_strVersion (*Modemversion*)            *)

(************Ende Ihrer I2C Applikation*************************)

(*INFO:*)
(*I2C-Library nutzt die Felder 13-16 von g_aCommunication.*)
(*Unter dem Feld 14 werden Modem Status Daten abgelegt*)
(*Unter dem Feld 15 werden I2C-Bus Leitungs Daten abgelegt*)
(*Unter dem Feld 16 werden Modem Versions Daten abgelegt*)
END_PROGRAM
```

ACTION Lauflicht:

```
(*****************************************************************

 Aktion:   Diese Aktion erzeugt auf der digitalen Ausgabekarte
           ein Lauflicht, wenn die boLauflicht Variable auf TRUE
           geht.

 Autor:    R. Lichtensteiger

 *****************************************************************)
t_LEDwechsel(IN := TRUE, PT := t#1000ms);
IF boLauflicht AND t_LEDwechsel.Q THEN
  t_LEDwechsel(IN := FALSE);
  t_LEDwechsel(IN := TRUE);
  IF bDigAusgabe1 = 0 THEN
    bDigAusgabe1 := 1;
  END_IF
  bDigAusgabe1 :=  ROL(bDigAusgabe1, 1);
END_IF
END_ACTION
```

ACTION SelectACInput:

```
(*****************************************************************

 Aktion:   Diese Aktion definiert anhand der bACInput Variable,
           welcher analoge Eingang auf der Analogkarte auf den
           digitalen Ausgang geschalten werden soll.

 Autor:    R. Lichtensteiger

 *****************************************************************)

IF bACInput.0 AND NOT bACInputLast.0 THEN
  dwFarbeRGB := 16#000080FF;   (*Farbe fuer AC Ausgang1*)
  prACAusgang1 := ADR(rACEingang1);  (*AC Eingang1 zu AC Ausgang1*)
  bACInput.1:= FALSE;
  bACInput.2:= FALSE;
END_IF
IF bACInput.1 AND NOT bACInputLast.1  THEN
  dwFarbeRGB := 16#00FF0000;   (*Farbe fuer AC Ausgang1*)
  prACAusgang1 := ADR(rACEingang2);  (*AC Eingang1 zu AC Ausgang1*)
  bACInput.0:= FALSE;
  bACInput.2:= FALSE;
END_IF
IF bACInput.2 AND NOT bACInputLast.2  THEN
  dwFarbeRGB := 16#008000FF;   (*Farbe fuer AC Ausgang1*)
  prACAusgang1 := ADR(rACEingang3);  (*AC Eingang1 zu AC Ausgang1*)
  bACInput.0:= FALSE;
  bACInput.1:= FALSE;
END_IF
```

```
bACInputLast.0 := bACInput.0;
bACInputLast.1 := bACInput.1;
bACInputLast.2 := bACInput.2;

rACAusgang1 := prACAusgang1^;

(*
BSP:
  ---/Blue/Green/Red (RGB)
16#00  FF   00    FF
*)
END_ACTION
```

ACTION SetFrequenz:

```
(*****************************************************************

 Aktion:   Diese Aktion erzeugt auf der digitalen Ausgabekarte
           eine Frequenz auf dem Bit 0, wenn die boLauflicht
           Variable auf FALSE geht.

 Autor:    R. Lichtensteiger

 *****************************************************************)
t_Frequenz.PT := WORD_TO_TIME(wFrequenz);
t_Frequenz(IN := TRUE);
IF NOT boLauflicht AND t_Frequenz.Q THEN
  t_Frequenz(IN := FALSE);
  t_Frequenz(IN := TRUE);
  bDigAusgabe1Last := bDigAusgabe1;
  IF (bDigAusgabe1 AND 2#00000001) = 0 THEN
    bDigAusgabe1 := bDigAusgabe1Last OR 2#00000001;
  ELSIF (bDigAusgabe1 AND 2#00000001) = 1 THEN
    bDigAusgabe1 := bDigAusgabe1Last AND 2#11111110;
  END_IF
END_IF
END_ACTION
```

5.7.3 I2C-Library

```
(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
PROGRAM FAST_PC_ComContol (*Fast-Task, Prioritaet 0, Zyklisch 1ms*)
(*****************************************************************

 Funktion: Dieses Programm treibt die RS232-Hardware.
          Dies wird von der Beckhoff TwinCat Library COMlib.lib
          unterstuetzt. Das Beispiel von TwinCAT, schlägt die
          Einbindung von PcComControl wie unten implementiert vor.

 Autor:    R. Lichtensteiger

 *****************************************************************)
VAR
          (* background communication with the COM port device *)
          COMportControl: PcComControl;
END_VAR
(* @END_DECLARATION := '0' *)
(*=================================================================
          Background communication with the COM port device.
          The SerialLineControl function block is supposed to be called
   in every PLC cycle. It communicates with the serial line
   hardware device and transmits or receives data. The
   SerialLineControl can be called in the standard task or in a
   separate fast task as well.          A fast separate task will be
   necessary at high baud rates.
*)
COMportControl(
 COMin:= g_COMin,          (* I/O data; see global variables *)
 ComOut:= g_COMout,                (* I/O data; see global variables *)
 TxBuffer:= g_TxBuffer,    (* transmit buffer; see global variables *)
 RxBuffer:= g_RxBuffer);(* receive buffer; see global variables  *)
END_PROGRAM


(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
FUNCTION_BLOCK FB_GetIdent
(*****************************************************************

 Funktion: Dieser Funktionsblock setzt den Modembefehl IDENT ab.

 Autor:    R. Lichtensteiger


 INDENT: (Befehlsnummer 16dez)
 Modem sendet I2C-OK. Dieser Befehl kann dazu verwendet werden,
 das Modem an der RS232 zu erkennen.
 Empfängt Das I2C-Modem diesen Befehl, wird ein Datenwort
 generiert, in dem die Bits 7 und 6 gesetzt sind.

 Befehl : Bit  7 6 5 4 3 2 1 0
        Wert 0 0 0 1 0 0 0 0 -> 16dez
 Antwort: Bit  7 6 5 4 3 2 1 0
        Wert 1 1 0 0 0 0 0 0 -> 192dez

 *****************************************************************)
```

```
VAR CONSTANT
  C_1BYTE:       BYTE :=1;(* In der Antwort erwartete BYTE *)
END_VAR
VAR_INPUT
  i_pboSend:     POINTER TO BOOL;  (* Senden bei i_boSend = TRUE *)
END_VAR
VAR_OUTPUT
  o_boModemOn:   BOOL;  (* Modem vorhanden, o_boModemOn = TRUE*)
  o_boDone:      BOOL := TRUE; (*Anfrage erfolgt*)
END_VAR
VAR (*ZG = Zeiger*)
  _pSende_Byte:     POINTER TO SendByte; (*ZG auf Sende_Byte*)
  _pReceive_Byte:   POINTER TO ReceiveByte;(*ZG auf Receive_Byte*)
  _bByte1ofIdent:   BYTE; (*Antwort von Modem*)
  _nRdIdx:          INT;  (*Leseposition*)
  _nReceivedByte:   INT;  (*Anzahl empfangener Bytes*)
  _nArrayIndex:     INT;  (*Leseposition von g_RxBuffer.Buffer*)
  _boWaitForAnswer_Ident: BOOL; (*Info ob Antwort erwartet wird*)
  _bCountByteReceived:    BYTE; (*Zaehler fuer empfangene Bytes*)
  _tTimeOut:        TON;  (*Antwortzeit Timer*)
  _tReceivedTime:   ARRAY [0..9] OF TIME;(*Speicher Antwortzeiten*)
  _ti:              INT;  (*Speicherplatz Index für Antwortzeiten*)
  _wTAverage:       WORD; (*Durchschnitt der Antwortzeiten*)
END_VAR
(* @END_DECLARATION := '0' *)
IF i_pboSend = 0 THEN (*Pointeradresse prüfen*)
  RETURN;
END_IF
(*ZG = Zeiger, Adr. = Adresse*)
_pSende_Byte := ADR(I2C_COM_Port.Sende_Byte); (*ZG Adr. zuweisen*)
_pReceive_Byte := ADR(I2C_COM_Port.Receive_Byte);(*     dito    *)
_tTimeOut(); (*Aufruf von Antwortzeit Timer*)
IF NOT _pSende_Byte^.Busy AND (*Sendestart abwarten*)
                (i_pboSend^ AND NOT _boWaitForAnswer_Ident) THEN
 g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

 (*zurücksetzen der Startvariable, fuer einmaligen Start*)
 i_pboSend^ := FALSE;
 _pSende_Byte^.SendByte := g_IDENT; (*Befehl an Sende_Byte*)

 _boWaitForAnswer_Ident := TRUE; (*es wird auf Antwort gewartet*)
 _bCountByteReceived := 0; (*Wert rücksetzen*)
 (*Timer mit Zeitlimit starten*)
 _tTimeOut(IN := TRUE, PT := t#2000ms);
 o_boDone := FALSE; (*Wert rücksetzen*)
END_IF
IF _pReceive_Byte^.ByteReceived THEN  (*Empfangene Bytes*)
 _bCountByteReceived :=  _bCountByteReceived + 1; (*zählen*)
END_IF
IF _tTimeOut.Q THEN (*Antwortzeit abgelaufen*)
  (*Wird kein BYTE innerhalb 2Sek empfangen,
    so wird der FB freigegeben*)
  _tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
  _boWaitForAnswer_Ident := FALSE;  (*Wert rücksetzen*)
  o_boModemOn := FALSE; (*Funktionsrueckgabewert*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
(*Ueberpruefen ob Antwort erfolgte*)
IF _bCountByteReceived = C_1BYTE AND _boWaitForAnswer_Ident THEN
  _boWaitForAnswer_Ident := FALSE; (*Wert rücksetzen*)

  _nRdIdx := g_RxBuffer.RdIdx; (*Leseposition von RxBuffer*)
```

```
  _nReceivedByte := C_1BYTE;   (*1 Byte für IDENT*)

  (*Finden des auszulesenden ARRAY's vom RxBuffer*)
  _nArrayIndex := ((BUFFERSIZE_DEBUGBUFFER + _nRdIdx) -
                   _nReceivedByte) MOD BUFFERSIZE_DEBUGBUFFER;
  IF _nArrayIndex >= 0 AND (*ARRAY Zugriffspruefung*)
    (_nArrayIndex + (_nReceivedByte - 1)) <= 300
  THEN
    _bByte1ofIdent := g_RxBuffer.Buffer[_nArrayIndex];(*auslesen*)
    IF _bByte1ofIdent = g_OK THEN (*Modem Antwort OK*)
      o_boModemOn := TRUE; (*Funktionsrueckgabewert*)
    ELSE
      o_boModemOn := FALSE;(*Funktionsrueckgabewert*)
    END_IF
  END_IF

  SaveTime();       (*Aufruf fuer Antwortzeiterfassung*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
END_FUNCTION_BLOCK

ACTION SaveTime:
(*****************************************************************

 Funktion: Diese Aktion SaveTime speichert 10 Antwortzeiten und
           berechnet die durchschnittliche Antwortzeit
           eines Modembefehls.

 Autor:    R. Lichtensteiger

 *****************************************************************)
IF _ti >= 0 AND _ti <= 9 THEN (*ARRAY Zugriffspruefung*)
  _tReceivedTime[_ti] := _tTimeOut.ET; (*Zeit erfassen*)
END_IF
_ti := _ti + 1; (*ARRAY Index erhöhen*)

IF _ti = 10 THEN
(*nach 10 Eintraegen Durchschnitt rechnen*)
_wTAverage := ( TIME_TO_WORD(_tReceivedTime[0]) +
               TIME_TO_WORD(_tReceivedTime[1]) +
               TIME_TO_WORD(_tReceivedTime[2]) +
               TIME_TO_WORD(_tReceivedTime[3]) +
               TIME_TO_WORD(_tReceivedTime[4]) +
               TIME_TO_WORD(_tReceivedTime[5]) +
               TIME_TO_WORD(_tReceivedTime[6]) +
               TIME_TO_WORD(_tReceivedTime[7]) +
               TIME_TO_WORD(_tReceivedTime[8]) +
               TIME_TO_WORD(_tReceivedTime[9])   ) / 10;
END_IF

_ti := _ti MOD 10; (*ARRAY Index, bei 10 nullen*)

_tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
END_ACTION
```

```
(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
FUNCTION_BLOCK FB_GetStatus
(*****************************************************************

 Funktion: Dieser Funktionsblock setzt den Modembefehl STATUS ab.

 Autor:    R. Lichtensteiger


 STATUS: (Befehlsnummer 48dez)
 Nachdem Das I2C-Modem diesen Befehl erhalten hat, liest er die
 Zustände der Leitungen SDA, SCL und INT aus. Diese werden zusammen
 mit einem OK in einem Byte an den PC zurückgesendet.

 Befehl : Bit  7 6 5 4 3 2 1 0
        Wert 0 0 1 1 0 0 0 0 -> 48dez
 Antwort: Bit  7 6 5 4 3 2 1 0
        Wert 1 1 0 0 0 I C D -> 192dez + max. 7 fuer ICD

 D = 1 wenn SDA auf High liegt
 C = 1 wenn SCL auf High liegt
 I = 1 wenn INT auf High liegt

 *****************************************************************)
VAR CONSTANT
  C_1BYTE:       BYTE :=1;(* In der Antwort erwartete BYTE *)
END_VAR
VAR_INPUT
  i_pboSend:     POINTER TO BOOL;  (* Senden bei i_boSend = TRUE *)
END_VAR
VAR_OUTPUT
  o_boSDA:       BOOL;  (* Datenleitung High, o_boSDA = TRUE*)
  o_boSCL:       BOOL;  (* Taktleitung  High, o_boSCL = TRUE*)
  o_boINT:       BOOL;  (* Infoleitung  High, o_boINT = TRUE*)
  o_boDone:      BOOL := TRUE;(*Anfrage erfolgt*)
END_VAR
VAR (*ZG = Zeiger*)
  _pSende_Byte:     POINTER TO SendByte; (*ZG auf Sende_Byte*)
  _pReceive_Byte:   POINTER TO ReceiveByte;(*ZG auf Receive_Byte*)
  _bByte1ofStatus:  BYTE;(*Antwort von Modem*)
  _nRdIdx:          INT; (*Leseposition*)
  _nReceivedByte:   INT; (*Anzahl empfangener Bytes*)
  _nArrayIndex:     INT; (*Leseposition von g_RxBuffer.Buffer*)
  _boWaitForAnswer_Status: BOOL; (*Info ob Antwort erwartet wird*)
  _bCountByteReceived:      BYTE;(*Zaehler fuer empfangene Bytes*)
  _tTimeOut:        TON; (*Antwortzeit Timer*)
  _tReceivedTime:   ARRAY [0..9] OF TIME;(*Speicher Antwortzeiten*)
  _ti:              INT; (*Speicherplatz Index für Antwortzeiten*)
  _wTAverage:       WORD;(*Durchschnitt der Antwortzeiten*)
END_VAR
(* @END_DECLARATION := '0' *)
IF i_pboSend = 0 THEN (*Pointeradresse prüfen*)
  RETURN;
END_IF
(*ZG = Zeiger, Adr. = Adresse*)
_pSende_Byte := ADR(I2C_COM_Port.Sende_Byte); (*ZG Adr. zuweisen*)
_pReceive_Byte := ADR(I2C_COM_Port.Receive_Byte);(*     dito    *)
_tTimeOut(); (*Aufruf von Antwortzeit Timer*)
IF NOT _pSende_Byte^.Busy AND (*Sendestart abwarten*)
              (i_pboSend^ AND NOT _boWaitForAnswer_Status) THEN
```

```
 g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

 (*zurücksetzen der Startvariable, fuer einmaligen Start*)
 i_pboSend^ := FALSE;
 _pSende_Byte^.SendByte := g_STATUS; (*Befehl an Sende_Byte*)

 _boWaitForAnswer_Status := TRUE; (*es wird auf Antwort gewartet*)
 _bCountByteReceived := 0;(*Wert rücksetzen*)
 (*Timer mit Zeitlimit starten*)
 _tTimeOut(IN := TRUE, PT := t#2000ms);
 o_boDone := FALSE; (*Wert rücksetzen*)
END_IF
IF _pReceive_Byte^.ByteReceived THEN  (*Empfangene Bytes*)
 _bCountByteReceived := _bCountByteReceived + 1;  (*zählen*)
END_IF
IF _tTimeOut.Q THEN  (*Antwortzeit abgelaufen*)
  (*Wird kein BYTE innerhalb 2Sek empfangen,
    so wird der FB freigegeben*)
  _tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
  _boWaitForAnswer_Status := FALSE;   (*Wert rücksetzen*)
  o_boSDA := FALSE; (*Funktionsrueckgabewert*)
  o_boSCL := FALSE; (*Funktionsrueckgabewert*)
  o_boINT := FALSE; (*Funktionsrueckgabewert*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
(*Ueberpruefen ob Antwort erfolgte*)
IF _bCountByteReceived = C_1BYTE AND _boWaitForAnswer_Status THEN
  _boWaitForAnswer_Status := FALSE; (*Wert rücksetzen*)

  _nRdIdx := g_RxBuffer.RdIdx; (*Leseposition von RxBuffer*)
  _nReceivedByte := C_1BYTE;   (*1 Byte für Speed*)

  (*Finden des auszulesenden ARRAY's vom RxBuffer*)
  _nArrayIndex := ((BUFFERSIZE_DEBUGBUFFER + _nRdIdx) -
                   _nReceivedByte) MOD BUFFERSIZE_DEBUGBUFFER;

  IF _nArrayIndex >= 0 AND (*ARRAY Zugriffspruefung*)
     (_nArrayIndex + (_nReceivedByte - 1)) <= 300
  THEN
    _bByte1ofStatus := g_RxBuffer.Buffer[_nArrayIndex];(*auslesen*)
    IF (_bByte1ofStatus AND g_MaskSDA) = g_MaskSDA THEN
      o_boSDA := TRUE;  (*Modem Antwort SDA*)
    ELSE
      o_boSDA := FALSE; (*Modem Antwort SDA*)
    END_IF
    IF (_bByte1ofStatus AND g_MaskSCL) = g_MaskSCL THEN
      o_boSCL := TRUE;  (*Modem Antwort SCL*)
    ELSE
      o_boSCL := FALSE; (*Modem Antwort SCL*)
    END_IF
    IF (_bByte1ofStatus AND g_MaskINT) = g_MaskINT THEN
      o_boINT := TRUE;  (*Modem Antwort INT*)
    ELSE
      o_boINT := FALSE;  (*Modem Antwort INT*)
    END_IF
  END_IF

  SaveTime();       (*Aufruf fuer Antwortzeiterfassung*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
END_FUNCTION_BLOCK
```

```
ACTION SaveTime:
(*************************************************************

 Funktion: Diese Aktion SaveTime speichert 10 Antwortzeiten und
           berechnet die durchschnittliche Antwortzeit
           eines Modembefehls.

 Autor:    R. Lichtensteiger

 *************************************************************)
IF _ti >= 0 AND _ti <= 9 THEN (*ARRAY Zugriffspruefung*)
  _tReceivedTime[_ti] := _tTimeOut.ET; (*Zeit erfassen*)
END_IF
_ti := _ti + 1; (*ARRAY Index erhöhen*)

IF _ti = 10 THEN
(*nach 10 Eintraegen Durchschnitt rechnen*)
_wTAverage := ( TIME_TO_WORD(_tReceivedTime[0]) +
                TIME_TO_WORD(_tReceivedTime[1]) +
                TIME_TO_WORD(_tReceivedTime[2]) +
                TIME_TO_WORD(_tReceivedTime[3]) +
                TIME_TO_WORD(_tReceivedTime[4]) +
                TIME_TO_WORD(_tReceivedTime[5]) +
                TIME_TO_WORD(_tReceivedTime[6]) +
                TIME_TO_WORD(_tReceivedTime[7]) +
                TIME_TO_WORD(_tReceivedTime[8]) +
                TIME_TO_WORD(_tReceivedTime[9])   ) / 10;
END_IF

_ti := _ti MOD 10; (*ARRAY Index, bei 10 nullen*)

_tTimeOut(IN := FALSE); (*Time wird geloescht*)
END_ACTION




(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\/I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
FUNCTION_BLOCK FB_GetVersion
(*************************************************************

 Funktion: Dieser Funktionsblock setzt den Modembefehl VERSION ab.

 Autor:    R. Lichtensteiger


 VERSION: (Befehlsnummer 80dez)
 Das I2C-Modem antwortet mit zwei Byte. Werden die Bytes in der
 Reihenfolge zusammengesetzt so ergibt sich die aktuelle
 Versionsnummer der geladenen Firmware

 Befehl : Bit   7 6 5 4 3 2 1 0
          Wert  0 1 0 1 0 0 0 0 -> 80dez

                Byte 1          Byte 2
 Antwort: Bit  7 6 5 4 3 2 1 0  7 6 5 4 3 2 1 0
          Wert 0 0 0 0 0 0 0 1  0 0 0 0 0 1 1 0
               erste Stelle     zweite Stelle

 im oberen Beispiel bedeutet das:
```

```
 Byte 1 = 1 Byte 2 = 6 ==> Version = 1,6

 ***************************************************************)
VAR CONSTANT
  C_2BYTE:       BYTE :=2;(* In der Antwort erwartete BYTE *)
END_VAR
VAR_INPUT
  i_pboSend:     POINTER TO BOOL;  (* Senden bei i_boSend = TRUE *)
END_VAR
VAR_OUTPUT
  o_strVersion:  STRING;(* Modemversion, o_strVersion = 'x.y'*)
  o_boDone:      BOOL := TRUE; (*Anfrage erfolgt*)
END_VAR
VAR (*ZG = Zeiger*)
  _pSende_Byte:     POINTER TO SendByte; (*ZG auf Sende_Byte*)
  _pReceive_Byte:   POINTER TO ReceiveByte;(*ZG auf Receive_Byte*)
  _bByte1ofVersion: BYTE; (*Antwort von Modem*)
  _bByte2ofVersion: BYTE; (*Antwort von Modem*)
  _nRdIdx:          INT;  (*Leseposition*)
  _nReceivedByte:   INT;  (*Anzahl empfangener Bytes*)
  _nArrayIndex:     INT;  (*Leseposition von g_RxBuffer.Buffer*)
  _boWaitForAnswer_Version: BOOL; (*Info ob Antwort erwartet wird*)
  _bCountByteReceived:      BYTE; (*Zaehler fuer empfangene Bytes*)
  _tTimeOut:        TON;  (*Antwortzeit Timer*)
  _tReceivedTime:   ARRAY [0..9] OF TIME;(*Speicher Antwortzeiten*)
  _ti:              INT;  (*Speicherplatz Index für Antwortzeiten*)
  _wTAverage:       WORD; (*Durchschnitt der Antwortzeiten*)
END_VAR

(* @END_DECLARATION := '0' *)
IF i_pboSend = 0 THEN (*Pointeradresse prüfen*)
  RETURN;
END_IF
(*ZG = Zeiger, Adr. = Adresse*)
_pSende_Byte := ADR(I2C_COM_Port.Sende_Byte); (*ZG Adr. zuweisen*)
_pReceive_Byte := ADR(I2C_COM_Port.Receive_Byte);(*     dito    *)
_tTimeOut(); (*Aufruf von Antwortzeit Timer*)
IF NOT _pSende_Byte^.Busy AND (*Sendestart abwarten*)
               (i_pboSend^ AND NOT _boWaitForAnswer_Version) THEN
 g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

 (*zurücksetzen der Startvariable, fuer einmaligen Start*)
 i_pboSend^ := FALSE;
 _pSende_Byte^.SendByte := g_VERSION; (*Befehl an Sende_Byte*)

 _boWaitForAnswer_Version := TRUE; (*es wird auf Antwort gewartet*)
 _bCountByteReceived := 0; (*Wert rücksetzen*)
 (*Timer mit Zeitlimit starten*)
 _tTimeOut(IN := TRUE, PT := t#2000ms);
 o_boDone := FALSE; (*Wert rücksetzen*)
END_IF
IF _pReceive_Byte^.ByteReceived THEN  (*Empfangene Bytes*)
 _bCountByteReceived :=  _bCountByteReceived + 1; (*zählen*)
END_IF
IF _tTimeOut.Q THEN (*Antwortzeit abgelaufen*)
  (*Wird kein BYTE innerhalb 2Sek empfangen,
    so wird der FB freigegeben*)
  _tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
  _boWaitForAnswer_Version := FALSE;  (*Wert rücksetzen*)
  _bByte1ofVersion := 0;   (*Wert rücksetzen*)
  _bByte2ofVersion := 0;   (*Wert rücksetzen*)
  o_strVersion := 'keine Verbindung';  (*Funktionsrueckgabewert*)
```

```
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
(*Ueberpruefen ob Antwort erfolgte*)
IF _bCountByteReceived = C_2BYTE AND _boWaitForAnswer_Version THEN
  _boWaitForAnswer_Version := FALSE; (*Wert rücksetzen*)

  _nRdIdx := g_RxBuffer.RdIdx; (*Leseposition von RxBuffer*)
  _nReceivedByte := C_2BYTE;   (*2 Byte für Version*)

  (*Finden des auszulesenden ARRAY's vom RxBuffer*)
  _nArrayIndex := ((BUFFERSIZE_DEBUGBUFFER + _nRdIdx) -
                   _nReceivedByte) MOD BUFFERSIZE_DEBUGBUFFER;

  IF _nArrayIndex >= 0 AND (*ARRAY Zugriffspruefung*)
    (_nArrayIndex + (_nReceivedByte - 1)) <= 300
  THEN
    (*auslesen*)
    _bByte1ofVersion := g_RxBuffer.Buffer[_nArrayIndex];
    _bByte2ofVersion := g_RxBuffer.Buffer[(BUFFERSIZE_DEBUGBUFFER
                + _nArrayIndex + 1) MOD BUFFERSIZE_DEBUGBUFFER];
    (*Modem Antwort*)  (*Funktionsrueckgabewert*)
    o_strVersion := CONCAT(BYTE_TO_STRING(_bByte1ofVersion), '.');
    o_strVersion := CONCAT(o_strVersion,
                      BYTE_TO_STRING(_bByte2ofVersion));
  END_IF

  SaveTime();       (*Aufruf fuer Antwortzeiterfassung*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
END_FUNCTION_BLOCK

ACTION SaveTime:
(*****************************************************************

 Funktion: Diese Aktion SaveTime speichert 10 Antwortzeiten und
           berechnet die durchschnittliche Antwortzeit
           eines Modembefehls.

 Autor:    R. Lichtensteiger

 *****************************************************************)
IF _ti >= 0 AND _ti <= 9 THEN (*ARRAY Zugriffspruefung*)
  _tReceivedTime[_ti] := _tTimeOut.ET; (*Zeit erfassen*)
END_IF
_ti := _ti + 1; (*ARRAY Index erhöhen*)

IF _ti = 10 THEN
(*nach 10 Eintraegen Durchschnitt rechnen*)
_wTAverage := ( TIME_TO_WORD(_tReceivedTime[0]) +
               TIME_TO_WORD(_tReceivedTime[1]) +
               TIME_TO_WORD(_tReceivedTime[2]) +
               TIME_TO_WORD(_tReceivedTime[3]) +
               TIME_TO_WORD(_tReceivedTime[4]) +
               TIME_TO_WORD(_tReceivedTime[5]) +
               TIME_TO_WORD(_tReceivedTime[6]) +
               TIME_TO_WORD(_tReceivedTime[7]) +
               TIME_TO_WORD(_tReceivedTime[8]) +
               TIME_TO_WORD(_tReceivedTime[9])   ) / 10;
END_IF

_ti := _ti MOD 10; (*ARRAY Index, bei 10 nullen*)
```

```
_tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
END_ACTION


(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\/I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
FUNCTION_BLOCK FB_ReadData
(*****************************************************************

 Funktion: Dieser Funktionsblock setzt den Modembefehl READ ab.

 Autor:    R. Lichtensteiger


 Das I2C-Modem liest ein oder mehrere Bytes aus dem Slave.
 Die Anzahl der zu lesenden Bytes wird durch das untere
 Nibbel (4 Bit) bestimmt. In einem Schritt können maximal 16 Bytes
 gelesen werden. Nachdem das Kommando READ vom PC zum Modem über-
 tragen wurde, muss die Slaveadresse übertragen werden. Das
 I2C-Modem antwortet mit einem I2C-OK und sendet dann die
 empfangenen Daten. Dabei wird das Datum das zuerst auf dem I2C-Bus
 empfangen wurde auch zuerst auf den PC übertragen.

 ACHTUNG:
 Es wird immer ein Byte mehr übertragen als im unteren Nibbel
 als Zahlenwert steht!!!

 Beispiel: (Befehlsnummer)
 Kommando 128 überträgt ein Byte
 Kommando 129 überträgt zwei Byte
 usw. -> 128dez + max. 15 fuer nnnn

             Byte 1          Byte 2
 Befehl : Bit  7 6 5 4 3 2 1 0  7 6 5 4 3 2 1 0
         Wert 1 0 0 0 n n n n  x n n n n n n n
             Kommando        I2C-Slaveadresse

             Byte 1          Byte 2 – 17
 Antwort: Bit  7 6 5 4 3 2 1 0  7 6 5 4 3 2 1 0
         Wert 1 1 0 0 0 0 0 0  n n n n n n n n
             OK -> 192dez     Daten 0-16
 *****************************************************************)
VAR CONSTANT
  C_bMaskForOffset:     BYTE :=15; (*Ergaenzung Anzahl Bytes *)
END_VAR
VAR_INPUT
  i_pboSend:      POINTER TO BOOL; (* Senden bei i_boSend = TRUE *)
  i_bAnzByte:       BYTE;  (*0 = Wirte 1 Byte,
                     usw. bis 15 = Wirte 16 Byte*)
  i_bSlaveAddress:   BYTE;  (*I2C-Kartenadresse*)
END_VAR
VAR_OUTPUT
  o_strErrorState:   STRING;(*Fehlerinfo als String *)
  o_boOK:           BOOL;  (* Antwort erfolgreich, o_boOK = TRUE*)
  o_abDaten:         ARRAY [1..16] OF BYTE; (*Daten aus I2C-Karte*)
  o_boDone:          BOOL := TRUE; (*Anfrage erfolgt*)
END_VAR
VAR (*ZG = Zeiger*)
  _pSende_Byte:      POINTER TO SendByte; (*ZG auf Sende_Byte*)
  _pReceive_Byte:    POINTER TO ReceiveByte;(*ZG auf Receive_Byte*)
  _bByte1ofWrite:    BYTE; (*Antwort von Modem*)
```

```
  _nRdIdx:          INT;  (*Leseposition*)
  _nReceivedByte:   INT;  (*Anzahl empfangener Bytes*)
  _nArrayIndex:     INT;  (*Leseposition von g_RxBuffer.Buffer*)
  _boWaitForAnswer_Read:BOOL; (*Info ob Antwort erwartet wird*)
  _bCountByteReceived:BYTE; (*Zaehler fuer empfangene Bytes*)
  _bCountByteSend:  BYTE := 0; (*Zaehler fuer gesendete Bytes*)
  _tTimeOut:        TON;  (*Antwortzeit Timer*)
  _bStep:           BYTE := 1; (*Sende Schrittnummer*)
  _bTempSendByte:   BYTE; (*zusammengesetzter Modembefehl*)
  _bIdx:            BYTE; (*Index von Ausgabearray o_abDaten*)
  _tReceivedTime:   ARRAY [0..9] OF TIME;(*Speicher Antwortzeiten*)
  _ti:              INT;  (*Speicherplatz Index für Antwortzeiten*)
  _wTAverage:       WORD; (*Durchschnitt der Antwortzeiten*)
END_VAR
(* @END_DECLARATION := '0' *)
IF i_pboSend = 0 THEN (*Pointeradresse prüfen*)
  RETURN;
END_IF
(*ZG = Zeiger, Adr. = Adresse*)
_pSende_Byte := ADR(I2C_COM_Port.Sende_Byte); (*ZG Adr. zuweisen*)
_pReceive_Byte := ADR(I2C_COM_Port.Receive_Byte);(*     dito    *)
_tTimeOut(); (*Aufruf von Antwortzeit Timer*)
CASE _bStep OF
  1: (*Sende Schritt 1*)
    IF NOT _pSende_Byte^.Busy AND (*Sendestart abwarten*)
                (i_pboSend^ AND NOT _boWaitForAnswer_Read) THEN
     g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

     (*zurücksetzen der Startvariable, fuer einmaligen Start*)
     i_pboSend^ := FALSE;
     IF i_bAnzByte >=1 AND i_bAnzByte <= 16 THEN (*Wertpruefung*)
       _bTempSendByte:= g_READ OR
                   (C_bMaskForOffset AND (i_bAnzByte-1));
       (*Befehl READ mit der Anzahl zu lesenden Bytes*)
       _pSende_Byte^.SendByte := _bTempSendByte; (*an Sende_Byte*)
     ELSE
       o_strErrorState := 'Wrong amount off byte'; (*Fehler*)
       RETURN;
     END_IF

     _boWaitForAnswer_Read := TRUE;(*es wird auf Antwort gewartet*)
     _bCountByteReceived := 0; (*Wert rücksetzen*)
     o_strErrorState := ''; (*Wert rücksetzen*)
    (*Timer mit Zeitlimit starten*)
     _tTimeOut(IN := TRUE, PT := t#2000ms);
     _bCountByteSend := 0; (*Wert rücksetzen*)
     _bStep := 2;  (*Sende Schritt 2 aktivieren*)
     o_boDone := FALSE;  (*Wert rücksetzen*)
    END_IF

  2: (*Sende Schritt 2*)
    IF NOT _pSende_Byte^.Busy AND _boWaitForAnswer_Read THEN
     _bCountByteSend := _bCountByteSend +1; (*zählen*)
     g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

    (*Send Slave Address*);
     _pSende_Byte^.SendByte := i_bSlaveAddress;
     _bStep := 1;              (*Sende Schritt 1 aktivieren*)
    END_IF

  ELSE
    _bCountByteSend := 0; (*Wert rücksetzen*)
```

```
    _bStep := 1;         (*Sende Schritt 1 aktivieren*)
END_CASE

IF _pReceive_Byte^.ByteReceived THEN  (*Empfangene Bytes*)
 _bCountByteReceived :=  _bCountByteReceived + 1; (*zählen*)
END_IF
IF _tTimeOut.Q THEN  (*Antwortzeit abgelaufen*)
  (*Wird kein BYTE innerhalb 2Sek empfangen,
    so wird der FB freigegeben*)
  _tTimeOut(IN := FALSE);  (*Timer wird gestoppt*)
  _boWaitForAnswer_Read := FALSE;  (*Wert rücksetzen*)
  _bStep := 1; (*Sende Schritt 1 aktivieren*)
  o_boOK := FALSE; (*Funktionsrueckgabewert*)
  o_strErrorState := 'Timeout'; (*Fehler als String ausgeben*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
(*Ueberpruefen ob Antwort erfolgte*)
IF _bCountByteReceived = (i_bAnzByte +1) AND
   _boWaitForAnswer_Read
THEN
  _boWaitForAnswer_Read := FALSE; (*Wert rücksetzen*)

  _nRdIdx := g_RxBuffer.RdIdx;     (*Leseposition von RxBuffer*)
  _nReceivedByte := i_bAnzByte +1; (*i_bAnzByte Byte +
                                    OK_Byte für Read*)

  (*Finden des ersten auszulesenden ARRAY's vom RxBuffer*)
  _nArrayIndex := ((BUFFERSIZE_DEBUGBUFFER + _nRdIdx) -
                   _nReceivedByte) MOD BUFFERSIZE_DEBUGBUFFER;

  IF _nArrayIndex >= 0 AND  (*ARRAY Zugriffspruefung*)
    (_nArrayIndex + (_nReceivedByte - 1)) <= 300
  THEN
    _bByte1ofWrite := g_RxBuffer.Buffer[_nArrayIndex];
    IF _bByte1ofWrite = g_OK THEN (*Modem Antwort OK*)
      o_boOK := TRUE; (*Funktionsrueckgabewert*)
      o_strErrorState := ''; (*Fehler als String ausgeben*)
      FOR _bIdx := 1 TO i_bAnzByte DO
        (*Empfangene Daten abfuellen*)
        o_abDaten[_bIdx] := (*Funktionsrueckgabewert*)
             g_RxBuffer.Buffer[(BUFFERSIZE_DEBUGBUFFER +
             _nArrayIndex + _bIdx) MOD BUFFERSIZE_DEBUGBUFFER];
      END_FOR
    END_IF
  END_IF

  SaveTime();       (*Aufruf fuer Antwortzeiterfassung*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
END_FUNCTION_BLOCK

ACTION SaveTime:
(*****************************************************************

 Funktion: Diese Aktion SaveTime speichert 10 Antwortzeiten und
           berechnet die durchschnittliche Antwortzeit
           eines Modembefehls.

 Autor:    R. Lichtensteiger

 ****************************************************************)
IF _ti >= 0 AND _ti <= 9 THEN  (*ARRAY Zugriffspruefung*)
```

```
  _tReceivedTime[_ti] := _tTimeOut.ET; (*Zeit erfassen*)
END_IF
_ti := _ti + 1; (*ARRAY Index erhöhen*)

IF _ti = 10 THEN
(*nach 10 Eintraegen Durchschnitt rechnen*)
_wTAverage := ( TIME_TO_WORD(_tReceivedTime[0]) +
               TIME_TO_WORD(_tReceivedTime[1]) +
               TIME_TO_WORD(_tReceivedTime[2]) +
               TIME_TO_WORD(_tReceivedTime[3]) +
               TIME_TO_WORD(_tReceivedTime[4]) +
               TIME_TO_WORD(_tReceivedTime[5]) +
               TIME_TO_WORD(_tReceivedTime[6]) +
               TIME_TO_WORD(_tReceivedTime[7]) +
               TIME_TO_WORD(_tReceivedTime[8]) +
               TIME_TO_WORD(_tReceivedTime[9])   ) / 10;
END_IF

_ti := _ti MOD 10; (*ARRAY Index, bei 10 nullen*)

_tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
END_ACTION

(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\/I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
FUNCTION_BLOCK FB_SetSpeed
(*****************************************************************

 Funktion: Dieser Funktionsblock setzt den Modembefehl SPEED ab.

 Autor:    R. Lichtensteiger


 SPEED:(Befehlsnummer 32dez)
 Die maximale Taktrate auf dem I2C-Bus kann mit diesem Befehl ein-
 gestellt werden. Konnte der neue Bustakt eingestellt werden, so
 antwortet Das I2C-Modem mit OK.

            Byte 1
 Befehl : Bit  7 6 5 4 3 2 1 0
        Wert 0 0 1 0 0 n n n  -> 32dez + max. 6 fuer nnn
            Kommando

            Byte 1
 Antwort: Bit  7 6 5 4 3 2 1 0
        Wert 1 1 0 0 0 0 0 0 -> 192dez
            OK

 Für n n n sind folgende dezimale Wert erlaubt:
 0 => I2C Busspeed = 43 KHz
 1 => I2C Busspeed = 28 KHz
 2 => I2C Busspeed = 17 KHz
 3 => I2C Busspeed = 9 KHz
 4 => I2C Busspeed = 5 KHz
 5 => I2C Busspeed = 2,5 KHz
 6 => I2C Busspeed = 1,3 KHz
 *****************************************************************)
VAR CONSTANT
  C_1BYTE:       BYTE :=1;(* In der Antwort erwartete BYTE *)
END_VAR
VAR_INPUT
```

```
 i_pboSend:      POINTER TO BOOL;  (* Senden bei i_boSend = TRUE *)
 i_bSpeed:       BYTE;             (* Geschwindigkeit I2C-Bus*)
END_VAR
VAR_OUTPUT
  o_boOK:        BOOL;  (* Antwort erfolgreich, o_boOK = TRUE*)
  o_boDone:      BOOL := TRUE;  (*Anfrage erfolgt*)
END_VAR
VAR (*ZG = Zeiger*)
  _pSende_Byte:     POINTER TO SendByte; (*ZG auf Sende_Byte*)
  _pReceive_Byte:   POINTER TO ReceiveByte;(*ZG auf Receive_Byte*)
  _bByte1ofSpeed:   BYTE; (*Antwort von Modem*)
  _nRdIdx:          INT;  (*Leseposition*)
  _nReceivedByte:   INT;  (*Anzahl empfangener Bytes*)
  _nArrayIndex:     INT;  (*Leseposition von g_RxBuffer.Buffer*)
  _boWaitForAnswer_Speed: BOOL; (*Info ob Antwort erwartet wird*)
  _bCountByteReceived:      BYTE; (*Zaehler fuer empfangene Bytes*)
  _tTimeOut:        TON;  (*Antwortzeit Timer*)
  _tReceivedTime:   ARRAY [0..9] OF TIME;(*Speicher Antwortzeiten*)
  _ti:              INT;  (*Speicherplatz Index für Antwortzeiten*)
  _wTAverage:       WORD; (*Durchschnitt der Antwortzeiten*)
END_VAR
(* @END_DECLARATION := '0' *)
IF i_pboSend = 0 THEN (*Pointeradresse prüfen*)
  RETURN;
END_IF
(*ZG = Zeiger, Adr. = Adresse*)
_pSende_Byte := ADR(I2C_COM_Port.Sende_Byte); (*ZG Adr. zuweisen*)
_pReceive_Byte := ADR(I2C_COM_Port.Receive_Byte);(*     dito    *)
_tTimeOut(); (*Aufruf von Antwortzeit Timer*)
IF NOT _pSende_Byte^.Busy AND (*Sendestart abwarten*)
                (i_pboSend^ AND NOT _boWaitForAnswer_Speed) THEN
 g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

 (*zurücksetzen der Startvariable, fuer einmaligen Start*)
 i_pboSend^ := FALSE;
 _pSende_Byte^.SendByte := i_bSpeed; (*Befehl an Sende_Byte*)

 _boWaitForAnswer_Speed := TRUE; (*es wird auf Antwort gewartet*)
 _bCountByteReceived := 0; (*Wert rücksetzen*)
 (*Timer mit Zeitlimit starten*)
 _tTimeOut(IN := TRUE, PT := t#2000ms);
 o_boDone := FALSE; (*Wert rücksetzen*)
END_IF
IF _pReceive_Byte^.ByteReceived THEN   (*Empfangene Bytes*)
 _bCountByteReceived := _bCountByteReceived + 1; (*zählen*)
END_IF
IF _tTimeOut.Q THEN (*Antwortzeit abgelaufen*)
  (*Wird kein BYTE innerhalb 2Sek empfangen,
    so wird der FB freigegeben*)
  _tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
  _boWaitForAnswer_Speed := FALSE; (*Wert rücksetzen*)
  o_boOK := FALSE; (*Funktionsrueckgabewert*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
(*Ueberpruefen ob Antwort erfolgte*)
IF _bCountByteReceived = C_1BYTE AND _boWaitForAnswer_Speed THEN
  _boWaitForAnswer_Speed := FALSE; (*Wert rücksetzen*)

  _nRdIdx := g_RxBuffer.RdIdx; (*Leseposition von RxBuffer*)
  _nReceivedByte := C_1BYTE;   (*1 Byte für Speed*)

  (*Finden des auszulesenden ARRAY's vom RxBuffer*)
```

```
  _nArrayIndex := ((BUFFERSIZE_DEBUGBUFFER + _nRdIdx) -
                   _nReceivedByte) MOD BUFFERSIZE_DEBUGBUFFER;

  IF _nArrayIndex >= 0 AND (*ARRAY Zugriffspruefung*)
    (_nArrayIndex + (_nReceivedByte - 1)) <= 300
  THEN
    _bByte1ofSpeed := g_RxBuffer.Buffer[_nArrayIndex];
    IF _bByte1ofSpeed = g_OK THEN (*Modem Antwort OK*)
      o_boOK := TRUE; (*Funktionsrueckgabewert*)
    ELSE
      o_boOK := FALSE; (*Funktionsrueckgabewert*)
    END_IF
  END_IF

  SaveTime();       (*Aufruf fuer Antwortzeiterfassung*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
END_FUNCTION_BLOCK

ACTION SaveTime:
(*********************************************************************

 Funktion: Diese Aktion SaveTime speichert 10 Antwortzeiten und
           berechnet die durchschnittliche Antwortzeit
           eines Modembefehls.

 Autor:    R. Lichtensteiger

 *********************************************************************)
IF _ti >= 0 AND _ti <= 9 THEN (*ARRAY Zugriffspruefung*)
  _tReceivedTime[_ti] := _tTimeOut.ET; (*Zeit erfassen*)
END_IF
_ti := _ti + 1; (*ARRAY Index erhöhen*)

IF _ti = 10 THEN
(*nach 10 Eintraegen Durchschnitt rechnen*)
_wTAverage := ( TIME_TO_WORD(_tReceivedTime[0]) +
                TIME_TO_WORD(_tReceivedTime[1]) +
                TIME_TO_WORD(_tReceivedTime[2]) +
                TIME_TO_WORD(_tReceivedTime[3]) +
                TIME_TO_WORD(_tReceivedTime[4]) +
                TIME_TO_WORD(_tReceivedTime[5]) +
                TIME_TO_WORD(_tReceivedTime[6]) +
                TIME_TO_WORD(_tReceivedTime[7]) +
                TIME_TO_WORD(_tReceivedTime[8]) +
                TIME_TO_WORD(_tReceivedTime[9])   ) / 10;
END_IF

_ti := _ti MOD 10; (*ARRAY Index, bei 10 nullen*)

_tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
END_ACTION
```

```
(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\/I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
FUNCTION_BLOCK FB_WriteData
(*****************************************************************

 Funktion: Dieser Funktionsblock setzt den Modembefehl WRITE ab.

 Autor:    R. Lichtensteiger


 Das I2C-Modem schreibt ein oder mehrere Bytes in den Slave. Die
 Anzahl der zu schreibenden Bytes wird durch das untere
 Nibbel (4 Bit) bestimmt. Mit einem WRITE können maximal 16 Byte
 übertragen werden. Nachdem das Kommando WRITE vom PC zum Modem
 übertragen wurde, müssen die Slaveadresse und die Daten übertragen
 werden. Das I2C-Modem antwortet mit einem I2C-OK.
 Beim Senden auf dem I2C-Bus wird das Datum das zuerst übertragen,
 das auch zuerst vom PC empfangen wurde.

 ACHTUNG:
 Es wird immer ein Byte mehr gelesen als im unteren Nibbel als
 Zahlenwert steht!!!

 Beispiel:
 Kommando 64 liest ein Byte
 Kommando 65 liest zwei Byte
 usw. -> 64dez + max. 15 fuer nnnn

             Byte 1          Byte 2          Byte 3 - 18
 Befehl : Bit  7 6 5 4 3 2 1 0   7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0
         Wert 0 1 0 0 n n n n   x n n n n n n n    n n n n n n n n
             Kommando         I2C-Slaveadresse   Daten 0-15

             Byte 1
 Antwort: Bit  7 6 5 4 3 2 1 0
         Wert 1 1 0 0 0 0 0 0
             OK -> 192dez

 I2C-NOT-OK:
 Diese Antwort wird statt I2C-OK gesendet, wenn bei der Übertagung
 ein Fehler aufgetreten ist.

             Byte 1
 Antwort: Bit  7 6 5 4 3 2 1 0
         Wert 0 0 0 0 c d a 0
                 NOT - OK

 Bedeutung der Bits:
 A: Fehler beim senden der I2C-Adresse. Dieses Bit wirdgesetzt,
    wenn der Slave die Adresse nicht quittiert hat.
 D: Fehler beim senden der I2C-Daten. Dieses Bit wird gesetzt,
    wenn der Slave den Empfang eines Bytes nicht quittiert hat.
 C: Das I2C-Modem hat das gesendete Kommando nicht erkannt.
 *****************************************************************)

VAR CONSTANT
  C_1BYTE:          BYTE :=1;(* In der Antwort erwartete BYTE *)
  C_bMaskForOffset:     BYTE :=15; (*Ergaenzung Anzahl Bytes *)
  C_bMaskFailedAddress: BYTE :=2;  (*Filter Erkennung Fehlerart *)
  C_bMaskByteLost:      BYTE :=4;  (*Filter Erkennung Fehlerart *)
  C_bMaskCommandUnknown:BYTE :=16; (*Filter Erkennung Fehlerart *)
```

```
END_VAR
VAR_INPUT
  i_pboSend:     POINTER TO BOOL;  (* Senden bei i_boSend = TRUE *)
  i_bAnzByte:    BYTE;  (*0 = Wirte 1 Byte,
                          usw. bis 15 = Wirte 16 Byte*)
  i_bSlaveAddress:    BYTE;  (*I2C-Kartenadresse*)
  i_abDaten:          ARRAY [1..16] OF BYTE;(*Daten zur I2C-Karte*)
END_VAR
VAR_OUTPUT
  o_strErrorState:     STRING;(*Fehlerinfo als String *)
  o_boOK:              BOOL;(* Antwort erfolgreich, o_boOK = TRUE*)
  o_boDone:            BOOL := TRUE; (*Anfrage erfolgt*)
END_VAR
VAR (*ZG = Zeiger*)
  _pSende_Byte:      POINTER TO SendByte; (*ZG auf Sende_Byte*)
  _pReceive_Byte:    POINTER TO ReceiveByte;(*ZG auf Receive_Byte*)
  _bByte1ofWrite:    BYTE; (*Antwort von Modem*)
  _nRdIdx:           INT;  (*Leseposition*)
  _nReceivedByte:    INT;  (*Anzahl empfangener Bytes*)
  _nArrayIndex:      INT;  (*Leseposition von g_RxBuffer.Buffer*)
  _boWaitForAnswer_Write:BOOL; (*Info ob Antwort erwartet wird*)
  _bCountByteReceived:  BYTE;  (*Zaehler fuer empfangene Bytes*)
  _bCountByteSend:      BYTE := 0; (*Zaehler fuer gesendete Bytes*)
  _tTimeOut:            TON;  (*Antwortzeit Timer*)
  _bStep:               BYTE := 1; (*Sende Schrittnummer*)
  _bTempSendByte:       BYTE; (*zusammengesetzter Modembefehl*)
  _tReceivedTime:   ARRAY [0..9] OF TIME;(*Speicher Antwortzeiten*)
  _ti:              INT;  (*Speicherplatz Index für Antwortzeiten*)
  _wTAverage:       WORD; (*Durchschnitt der Antwortzeiten*)
END_VAR
(* @END_DECLARATION := '0' *)
IF i_pboSend = 0 THEN (*Pointeradresse prüfen*)
  RETURN;
END_IF
(*ZG = Zeiger, Adr. = Adresse*)
_pSende_Byte := ADR(I2C_COM_Port.Sende_Byte); (*ZG Adr. zuweisen*)
_pReceive_Byte := ADR(I2C_COM_Port.Receive_Byte);(*     dito    *)
_tTimeOut(); (*Aufruf von Antwortzeit Timer*)
CASE _bStep OF
  1: (*Sende Schritt 1*)
    IF NOT _pSende_Byte^.Busy AND (*Sendestart abwarten*)
               (i_pboSend^ AND NOT _boWaitForAnswer_Write) THEN
     g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

     (*zurücksetzen der Startvariable, fuer einmaligen Start*)
     i_pboSend^ := FALSE;
     IF i_bAnzByte >=1 AND i_bAnzByte <= 16 THEN (*Wertpruefung*)
       (*Befehl WRITE mit der Anzahl zu lesenden Bytes*)
       _bTempSendByte := g_WRITE OR
                    (C_bMaskForOffset AND (i_bAnzByte-1));
       (*Befehl an Sende_Byte*)
       _pSende_Byte^.SendByte := _bTempSendByte;
     ELSE
       o_strErrorState := 'Wrong amount off byte'; (*Fehler*)
       RETURN;
     END_IF

     _boWaitForAnswer_Write := TRUE;(*auf Antwort warten*)
     _bCountByteReceived := 0; (*Wert rücksetzen*)
    (*Timer mit Zeitlimit starten*)
     _tTimeOut(IN := TRUE, PT := t#2000ms);
     _bCountByteSend := 0; (*Wert rücksetzen*)
```

```
    _bStep := 2;  (*Sende Schritt 2 aktivieren*)
    o_boDone := FALSE;   (*Wert rücksetzen*)
   END_IF

  2: (*Sende Schritt 2*)
   IF NOT _pSende_Byte^.Busy AND _boWaitForAnswer_Write THEN
    _bCountByteSend := _bCountByteSend +1;  (*zählen*)
    g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

   (*Send Slave Address*);
    _pSende_Byte^.SendByte := i_bSlaveAddress;
                        (*Send Slave Address*);
    _bStep := 3;  (*Sende Schritt 3 aktivieren*)
   END_IF

  3: (*Sende Schritt 3*)
   IF NOT _pSende_Byte^.Busy AND _boWaitForAnswer_Write AND
      _bCountByteSend < (i_bAnzByte +1)(*pruefen*)
   THEN
    _bCountByteSend := _bCountByteSend +1;  (*zählen*)
    g_boSend := TRUE; (* siehe Globale Variablen GV_I2C_Lib *)

    IF _bCountByteSend >= 2 THEN (*Schritt 1-2 erfolgt*)
      _pSende_Byte^.SendByte := i_abDaten[_bCountByteSend -1];
                        (*Send Data to Slave*)
    END_IF
   ELSE
    _bStep := 1; (*alles gesendet*) (*Sende Schritt 3 aktivieren*)
   END_IF

  ELSE
    _bCountByteSend := 0; (*Wert rücksetzen*)
    _bStep := 1;         (*Sende Schritt 1 aktivieren*)
END_CASE

IF _pReceive_Byte^.ByteReceived THEN  (*Empfangene Bytes*)
 _bCountByteReceived :=  _bCountByteReceived + 1;(*zählen*)
END_IF
IF _tTimeOut.Q THEN  (*Antwortzeit abgelaufen*)
  (*Wird kein BYTE innerhalb 2Sek empfangen,
    so wird der FB freigegeben*)
  _tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
  _boWaitForAnswer_Write := FALSE;  (*Wert rücksetzen*)
  _bStep := 1; (*Sende Schritt 1 aktivieren*)
  o_strErrorState := 'TimeOut'; (*Fehler als String ausgeben*)
  o_boOK := FALSE;(*Funktionsrueckgabewert*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
(*Ueberpruefen ob Antwort erfolgte*)
IF _bCountByteReceived = C_1BYTE AND _boWaitForAnswer_Write THEN
  _boWaitForAnswer_Write := FALSE; (*Wert rücksetzen*)

  _nRdIdx := g_RxBuffer.RdIdx; (*Leseposition von RxBuffer*)
  _nReceivedByte := C_1BYTE;   (*1 Byte für Write*)

  (*Finden des auszulesenden ARRAY's vom RxBuffer*)
  _nArrayIndex := ((BUFFERSIZE_DEBUGBUFFER + _nRdIdx) -
                  _nReceivedByte) MOD BUFFERSIZE_DEBUGBUFFER;

  IF _nArrayIndex >= 0 AND  (*ARRAY Zugriffspruefung*)
    (_nArrayIndex + (_nReceivedByte - 1)) <= 300
  THEN
```

```
    _bByte1ofWrite := g_RxBuffer.Buffer[_nArrayIndex];
    IF _bByte1ofWrite = g_OK THEN (*Modem Antwort OK*)
      o_boOK := TRUE; (*Funktionsrueckgabewert*)
      o_strErrorState := ''; (*Fehler als String ausgeben*)
    ELSE
      IF (_bByte1ofWrite AND C_bMaskFailedAddress) =
                                C_bMaskFailedAddress THEN
        o_strErrorState := 'Address failed';  (*Fehler String*)
      ELSIF (_bByte1ofWrite AND C_bMaskByteLost) =
                                     C_bMaskByteLost THEN
        o_strErrorState := 'Byte lost';       (*Fehler String*)
      ELSIF (_bByte1ofWrite AND C_bMaskCommandUnknown) =
                                  C_bMaskCommandUnknown THEN
        o_strErrorState := 'Command unknown'; (*Fehler String*)
      ELSE
        o_strErrorState := CONCAT('Unknown',' '); (*Fehler String*)
        o_strErrorState := CONCAT('o_strErrorState',
                        BYTE_TO_STRING(_bByte1ofWrite));
      END_IF
    END_IF
  END_IF

  SaveTime();       (*Aufruf fuer Antwortzeiterfassung*)
  o_boDone := TRUE; (*Info setzen fuer erfolgte Anfrage*)
END_IF
END_FUNCTION_BLOCK

ACTION SaveTime:
(*****************************************************************

 Funktion: Diese Aktion SaveTime speichert 10 Antwortzeiten und
           berechnet die durchschnittliche Antwortzeit
           eines Modembefehls.

 Autor:    R. Lichtensteiger

 *****************************************************************)
IF _ti >= 0 AND _ti <= 9 THEN (*ARRAY Zugriffspruefung*)
  _tReceivedTime[_ti] := _tTimeOut.ET; (*Zeit erfassen*)
END_IF
_ti := _ti + 1; (*ARRAY Index erhöhen*)

IF _ti = 10 THEN
(*nach 10 Eintraegen Durchschnitt rechnen*)
_wTAverage := ( TIME_TO_WORD(_tReceivedTime[0]) +
               TIME_TO_WORD(_tReceivedTime[1]) +
               TIME_TO_WORD(_tReceivedTime[2]) +
               TIME_TO_WORD(_tReceivedTime[3]) +
               TIME_TO_WORD(_tReceivedTime[4]) +
               TIME_TO_WORD(_tReceivedTime[5]) +
               TIME_TO_WORD(_tReceivedTime[6]) +
               TIME_TO_WORD(_tReceivedTime[7]) +
               TIME_TO_WORD(_tReceivedTime[8]) +
               TIME_TO_WORD(_tReceivedTime[9])   ) / 10;
END_IF

_ti := _ti MOD 10; (*ARRAY Index, bei 10 nullen*)

_tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
END_ACTION
```

```
(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
FUNCTION fErrorTypToByte : BYTE
(*****************************************************************

 Funktion: Diese Funktion setzt die Com-Fehler in Nummern um.
 Zweck:    Nummern sind leichter zu vergleichen.

 Autor:    R. Lichtensteiger

 *****************************************************************)

VAR_INPUT
 i_TError:    ComError_t;
END_VAR
VAR
END_VAR
(* @END_DECLARATION := '0' *)
IF COMERROR_NOERROR = i_TError THEN
  fErrorTypToByte:= 0;
ELSIF COMERROR_PARAMETERCHANGED = i_TError THEN
  fErrorTypToByte:= 1;
ELSIF COMERROR_TXBUFFOVERRUN = i_TError THEN
  fErrorTypToByte:= 2;
ELSIF COMERROR_STRINGOVERRUN = i_TError THEN
  fErrorTypToByte:= 10;
ELSIF COMERROR_ZEROCHARINVALID = i_TError THEN
  fErrorTypToByte:= 11;
ELSIF COMERROR_INVALIDPOINTER = i_TError THEN
  fErrorTypToByte:= 20;
ELSIF COMERROR_INVALIDRXPOINTER = i_TError THEN
  fErrorTypToByte:= 21;
ELSIF COMERROR_INVALIDRXLENGTH = i_TError THEN
  fErrorTypToByte:= 22;
ELSIF COMERROR_DATASIZEOVERRUN = i_TError THEN
  fErrorTypToByte:= 23;
ELSE
  fErrorTypToByte:= 255;
END_IF
END_FUNCTION


(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
PROGRAM I2C_COM_Port
(*****************************************************************

 Funktion: Dieses Programm treibt die SendByte- und ReceiveByte-
           Bausteine. Diese werden von der Beckhoff TwinCat
           Library COMlib.lib zur Verfügung gestellt.

 Autor:    R. Lichtensteiger

 *****************************************************************)

VAR
  Sende_Byte:   SendByte;   (*Bytesender, Instanz bilden*)
  SendBusy:     BOOL;       (*meldet Funktion in Arbeit*)
  SendErrorID:  ComError_t; (*Fehlerwertstruktur*)
```

```
  Receive_Byte: ReceiveByte;(*Byteempfaenger, Instanz bilden*)
  ByteReceived: BOOL;       (*Info, dass Byte emfangen wurde*)
  ReceivedByte: BYTE;       (*Anzahl empfangener Bytes*)
  LastReceivedByte: BYTE;   (*Hilfsvariable*)

  ReceiveErrorID:   ComError_t;(*Fehlerwertstruktur*)
  ReceiveCounter:   UDINT;  (*Empfangszaehler*)
END_VAR

VAR_OUTPUT
  o_bErrorTx:       BYTE; (*TxCom-Fehler in Nummern*)
  o_bErrorRx:       BYTE; (*RxCom-Fehler in Nummern*)
END_VAR


(* @END_DECLARATION := '0' *)
(*=============================================================
 Sende_Byte
 Den Sende Funktionsblock jeden Zyklus aufrufen, um die Ueber-
 tragung zu treiben. Also so lange der FB Busy meldet, bis die
 Uebertragung beendet ist. Die zu uebertragenden Daten werden im
 g_TxBuffer abgelegt.
*)
IF g_boSend OR Sende_Byte.Busy THEN (*einmalig Byte senden*)

  Sende_Byte(TXbuffer:= g_TxBuffer,  (* see global variables *)
                        Busy=> SendBusy,       (*Status*)
                        Error=> SendErrorID);  (*Fehlermeldung*)

  g_boSend := FALSE; (*ruecksetzen*)
END_IF
o_bErrorTx := fErrorTypToByte(SendErrorID); (*Fehler als Nummer*)

(*=============================================================
 Receive_Byte
 Der Funktionsblock empfängt die Daten im g_RxBuffer.
*)
Receive_Byte(
  RXbuffer:= g_RxBuffer, (* see global variables *)
  ByteReceived=> ByteReceived,(*meldet Byteemfang*)
  ReceivedByte=> ReceivedByte,(*meldet Anzahl Byte*)
  Error=> ReceiveErrorID      (*Fehlermeldung*)
  );
IF ByteReceived THEN (*Empaenger meldet Byte empfangen*)
  ReceiveCounter := ReceiveCounter + 1; (*zaehlen*)
  LastReceivedByte := ReceivedByte;     (*speichern*)
END_IF
o_bErrorRx := fErrorTypToByte(ReceiveErrorID);(*Fehler als Nummer*)
END_PROGRAM


(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '\/I2C_Library' *)
(* @SYMFILEFLAGS := '2048' *)
PROGRAM READ_WRITE_IO_CTRL (*FastTask, Prioritaet 0, Zyklisch 1ms*)
(*************************************************************

 Funktion: Dieses Programm koordiniert den Datenaustausch
           zwischen der SoftPLC von Beckhoff TwinCat und den
           I2C-Komponenten von www.Horter.de .
           Der ARRAY g_aCommunication wird vollständig ab-
           gearbeitet. Also alle 16 Positionen werden zyklisch
```

```
        verarbeitet. Nicht abgefüllte ARRAY-Plaetze sind mit 99
        initialisiert und werden uebersprungen.

 Autor:    R. Lichtensteiger

 ****************************************************************)

VAR
  boInit:       BOOL;  (*fuer einmalige Initialisierung*)
  boW:          BOOL;  (*Autostart fuer WRITE*)
  boId:         BOOL;  (*Autostart fuer IDENT*)
  boSt:         BOOL;  (*Autostart fuer STATUS*)
  boV:          BOOL;  (*Autostart fuer VERSION*)
  boR:          BOOL;  (*Autostart fuer READ*)
  boSP:         BOOL;  (*Autostart fuer SPEED*)
  bCardNumber:  BYTE := 0;  (*Karten- oder Anfrageplatznummer*)
  bModemCmd:    BYTE;  (*Modembefehlsnummer*)
  fbGetVersion: FB_GetVersion;  (*Instanz ModemVersions-Anfrage*)
  fbGetIdent:   FB_GetIdent;    (*Instanz fuer ModemIdent-Anfrage*)
  fbSetSpeed:   FB_SetSpeed;    (*Instanz fuer BusSpeed-Setzen*)
  fbGetStatus:  FB_GetStatus;(*Instanz LeitungsStatus-Anfrage*)
  fbWriteData:  FB_WriteData;(*Instanz fuer I2CKomponenten-Setzen*)
  fbReadData:   FB_ReadData;(*Instanz fuer I2CKomponenten-Anfrage*)
  _ti:          INT;  (*Speicherplatz Index für Antwortzeiten*)
  _tTimeOut:    TON;  (*Antwortzeit Timer*)
  _tReceivedTime:   ARRAY [0..9] OF TIME;(*Speicher Antwortzeiten*)
  _wTAverage:   WORD; (*Durchschnitt der Antwortzeiten*)
END_VAR
VAR_INPUT
END_VAR
(* @END_DECLARATION := '0' *)
IF NOT boInit THEN
  (*Initialisierung*)
  boInit := TRUE;
  (*Alle ARRAY-Plaetze koennen von der PLC-Applikation
    ueberschrieben werden, somit wuerden zusätzlich vier Plaetze
    für das Anwenderprogramm frei!*)
  (*Speed: default = 43kHz*)
  g_aCommunication[13].i_bModemCmd := g_SPEED_43KHz;
  g_aCommunication[14].i_bModemCmd := g_IDENT;
  g_aCommunication[15].i_bModemCmd := g_STATUS;
  g_aCommunication[16].i_bModemCmd := g_VERSION;
  bCardNumber := 1; (*erster abzuarbeitender ARRAY-Platz*)
  _tTimeOut(IN := TRUE, PT := t#2000ms); (*Antwortzeit Timer*)
  WhatOrder(); (*Initialisierung*)
END_IF
_tTimeOut();(*Aufruf von Antwortzeit Timer*)

(*Auslesen von Modembefehl*)
bModemCmd := g_aCommunication[bCardNumber].i_bModemCmd;
CASE  bModemCmd OF
  (*Erkennen des Modembefehls*)
  g_SPEED_43KHz, g_SPEED_28KHz, g_SPEED_17KHz, g_SPEED_9KHz,
  g_SPEED_5KHz, g_SPEED_2500Hz, g_SPEED_1300Hz:
   IF fbGetIdent.o_boDone AND
      fbGetStatus.o_boDone AND
      fbGetVersion.o_boDone AND
      fbWriteData.o_boDone AND
      fbReadData.o_boDone
   THEN (*ueberpruefen ob Absetzen von Befehl erlaubt ist*)
    (*Befehl senden mit entsprechenden Daten*)
    fbSetSpeed(i_pboSend:= ADR(boSP), (*Input Startvariable*)
```

```
            i_bSpeed := bModemCmd);(*Input Modembefehl*)
   (*Daten von I2C-BusSpeed
     in globale ARRAY-Position 13 schreiben*)
   IF NOT fbSetSpeed._boWaitForAnswer_Speed THEN
     (*Antwort erfolgt*)
     (*Daten von Antwort in entsprechendem Platz des ARRAY
       g_aCommunication ablegen*)
     g_aCommunication[bCardNumber].o_boOK := fbSetSpeed.o_boOK;
     IF g_aCommunication[bCardNumber].o_boOK THEN
       g_aCommunication[bCardNumber].o_strErrorState := 'OK';
     ELSE
       g_aCommunication[bCardNumber].o_strErrorState :=
                                        'Speed not set';
     END_IF
     (*nächster abzuarbeitender ARRAY-Platz setzen*)
     bCardNumber := bCardNumber + 1;
     WhatOrder(); (*neue Startvariable setzen*)
   END_IF
  END_IF

 g_IDENT:
  IF fbSetSpeed.o_boDone AND
     fbGetStatus.o_boDone AND
     fbGetVersion.o_boDone AND
     fbWriteData.o_boDone AND
     fbReadData.o_boDone
  THEN  (*ueberpruefen ob Absetzen von Befehl erlaubt ist*)
   (*Befehl senden mit entsprechenden Daten*)
   fbGetIdent(i_pboSend:= ADR(boId));  (*Input Startvariable*)
   (*Daten von I2C-Modem in globale ARRAY-Position 14 schreiben*)
   IF NOT fbGetIdent._boWaitForAnswer_Ident THEN
     (*Antwort erfolgt*)
     (*Daten von Antwort in entsprechendem Platz des ARRAY
       g_aCommunication ablegen*)
     g_aCommunication[bCardNumber].o_boOK :=
                                    fbGetIdent.o_boModemOn;
     IF g_aCommunication[bCardNumber].o_boOK THEN
       g_aCommunication[bCardNumber].o_strErrorState := 'OK';
     ELSE
       g_aCommunication[bCardNumber].o_strErrorState :=
                                      'not connected';
     END_IF
     (*nächster abzuarbeitender ARRAY-Platz setzen*)
     bCardNumber := bCardNumber + 1;
     WhatOrder(); (*neue Startvariable setzen*)
   END_IF
  END_IF

 g_STATUS:
  IF fbGetIdent.o_boDone AND
     fbSetSpeed.o_boDone AND
     fbGetVersion.o_boDone AND
     fbWriteData.o_boDone AND
     fbReadData.o_boDone
  THEN  (*ueberpruefen ob Absetzen von Befehl erlaubt ist*)
   (*Befehl senden mit entsprechenden Daten*)
   fbGetStatus(i_pboSend:= ADR(boSt)); (*Input Startvariable*)
   (*Daten von I2C-Busleitungen in
     globale ARRAY-Position 15 schreiben*)
   IF NOT fbGetStatus._boWaitForAnswer_Status THEN
     (*Antwort erfolgt*)
     (*Daten von Antwort in entsprechendem Platz des ARRAY
```

```
      g_aCommunication ablegen*)
    g_aCommunication[bCardNumber].o_boOK :=
              ((fbGetStatus._bByte1ofStatus AND g_OK) = g_OK);
    IF g_aCommunication[bCardNumber].o_boOK THEN
      g_aCommunication[bCardNumber].o_strErrorState := 'OK';
    ELSE
      g_aCommunication[bCardNumber].o_strErrorState :=
                                        'Stat unknown';
    END_IF
    g_aCommunication[bCardNumber].o_boINT := fbGetStatus.o_boINT;
    g_aCommunication[bCardNumber].o_boSCL := fbGetStatus.o_boSCL;
    g_aCommunication[bCardNumber].o_boSDA := fbGetStatus.o_boSDA;
    (*nächster abzuarbeitender ARRAY-Platz setzen*)
    bCardNumber := bCardNumber + 1;
    WhatOrder(); (*neue Startvariable setzen*)
  END_IF
 END_IF

g_VERSION:
 IF fbGetIdent.o_boDone AND
    fbGetStatus.o_boDone AND
    fbSetSpeed.o_boDone AND
    fbWriteData.o_boDone AND
    fbReadData.o_boDone
 THEN  (*ueberpruefen ob Absetzen von Befehl erlaubt ist*)
  (*Befehl senden mit entsprechenden Daten*)
  fbGetVersion(i_pboSend:= ADR(boV)); (*Input Startvariable*)
  (*Daten von I2C-Modem in
    globale ARRAY-Position 16 schreiben*)
  IF NOT fbGetVersion._boWaitForAnswer_Version THEN
    (*Antwort erfolgt*)
    (*Daten von Antwort in entsprechendem Platz des ARRAY
      g_aCommunication ablegen*)
    IF fbGetVersion._bByte1ofVersion >= 1 AND
       fbGetVersion._bByte2ofVersion >= 4
    THEN
      g_aCommunication[bCardNumber].o_boOK := TRUE;
      g_aCommunication[bCardNumber].o_strErrorState := 'OK';
    ELSE
      g_aCommunication[bCardNumber].o_boOK := FALSE;
      g_aCommunication[bCardNumber].o_strErrorState := 'to low';
    END_IF
    g_aCommunication[bCardNumber].o_strVersion
                              := fbGetVersion.o_strVersion;
    (*nächster abzuarbeitender ARRAY-Platz setzen*)
    bCardNumber := bCardNumber + 1;
    WhatOrder(); (*neue Startvariable setzen*)
  END_IF
 END_IF

g_WRITE:
 IF fbGetIdent.o_boDone AND
    fbGetStatus.o_boDone AND
    fbGetVersion.o_boDone AND
    fbSetSpeed.o_boDone AND
    fbReadData.o_boDone
 THEN  (*ueberpruefen ob Absetzen von Befehl erlaubt ist*)
  (*Daten vom globalen ARRAY zu I2C-Komponente senden*)
  fbWriteData.i_abDaten :=
                 g_aCommunication[bCardNumber].i_abDaten;
  fbWriteData.i_bAnzByte :=
                g_aCommunication[bCardNumber].i_bAnzByte;
```

```
   fbWriteData.i_bSlaveAddress :=
            g_aCommunication[bCardNumber].i_bSlaveAddress;
   (*Befehl senden mit entsprechenden Daten*)
   fbWriteData(i_pboSend:= ADR(boW)); (*Input Startvariable*)
   IF NOT fbWriteData._boWaitForAnswer_Write THEN
    (*Antwort erfolgt*)
    (*Daten von Antwort in entsprechendem Platz des ARRAY
      g_aCommunication ablegen*)
    g_aCommunication[bCardNumber].o_boOK :=fbWriteData.o_boOK;
    g_aCommunication[bCardNumber].o_strErrorState
                          := fbWriteData.o_strErrorState;
    (*nächster abzuarbeitender ARRAY-Platz setzen*)
    bCardNumber := bCardNumber + 1;
    WhatOrder(); (*neue Startvariable setzen*)
   END_IF
  END_IF

 g_READ:
  IF fbGetIdent.o_boDone AND
     fbGetStatus.o_boDone AND
     fbGetVersion.o_boDone AND
     fbWriteData.o_boDone AND
     fbSetSpeed.o_boDone
   THEN  (*ueberpruefen ob Absetzen von Befehl erlaubt ist*)
    fbReadData.i_bAnzByte :=
                g_aCommunication[bCardNumber].i_bAnzByte;
    fbReadData.i_bSlaveAddress :=
            g_aCommunication[bCardNumber].i_bSlaveAddress;
    (*Befehl senden mit entsprechenden Daten*)
    fbReadData(i_pboSend:= ADR(boR)); (*Input Startvariable*)
    IF NOT fbReadData._boWaitForAnswer_Read THEN
     (*Antwort erfolgt*)
     (*Daten von Antwort in entsprechendem Platz des ARRAY
       g_aCommunication ablegen*)
     (*Daten von I2C-Komponente in globalen ARRAY schreiben*)
     g_aCommunication[bCardNumber].o_boOK :=fbReadData.o_boOK;
     g_aCommunication[bCardNumber].o_strErrorState
                          := fbReadData.o_strErrorState;
     g_aCommunication[bCardNumber].o_abDaten
                          := fbReadData.o_abDaten;
     IF g_aCommunication[bCardNumber].o_abDaten[1] = 0 THEN
       (*fuer Diagnose erstellt -> Breakpoint*)
       bCardNumber := bCardNumber;
     END_IF
     (*nächster abzuarbeitender ARRAY-Platz setzen*)
     bCardNumber := bCardNumber + 1;
     WhatOrder(); (*neue Startvariable setzen*)
    END_IF
   END_IF

  ELSE
   (*Fruehzeitiges weiterschalten,
     da keine gültigen Befehle eingetragen sind*)
   (*nächster abzuarbeitender ARRAY-Platz setzen*)
   bCardNumber := bCardNumber + 1;
   WhatOrder(); (*neue Startvariable setzen*)
END_CASE

IF NOT (bCardNumber >= 1 AND
       bCardNumber <= (g_MaxI2CCard + g_ModemCmd))
THEN
 bCardNumber := 1;  (*Zuruecksetzen fuer naechsten Abfragezyklus*)
```

```
  SaveTime();       (*Aufruf fuer Antwortzeiterfassung*)
  _tTimeOut(IN := TRUE, PT := t#2000ms);  (*Timer starten*)
END_IF

I2C_COM_Port();  (*SendByte- und ReceiveByte-Bausteine aufrufen.*)
FAST_PC_ComContol();(*Hardwaretreiber*)
END_PROGRAM

ACTION SaveTime:
(*****************************************************************

 Funktion: Diese Aktion SaveTime speichert 10 Antwortzeiten und
           berechnet die durchschnittliche Antwortzeit für je 16
           mögliche Modembefehle.

 Autor:    R. Lichtensteiger

 *****************************************************************)
IF _ti >= 0 AND _ti <= 9 THEN  (*ARRAY Zugriffspruefung*)
  _tReceivedTime[_ti] := _tTimeOut.ET; (*Zeit erfassen*)
END_IF
_ti := _ti + 1; (*ARRAY Index erhöhen*)

IF _ti = 10 THEN
(*nach 10 Eintraegen Durchschnitt rechnen*)
_wTAverage := ( TIME_TO_WORD(_tReceivedTime[0]) +
                TIME_TO_WORD(_tReceivedTime[1]) +
                TIME_TO_WORD(_tReceivedTime[2]) +
                TIME_TO_WORD(_tReceivedTime[3]) +
                TIME_TO_WORD(_tReceivedTime[4]) +
                TIME_TO_WORD(_tReceivedTime[5]) +
                TIME_TO_WORD(_tReceivedTime[6]) +
                TIME_TO_WORD(_tReceivedTime[7]) +
                TIME_TO_WORD(_tReceivedTime[8]) +
                TIME_TO_WORD(_tReceivedTime[9])   ) / 10;
END_IF

_ti := _ti MOD 10; (*ARRAY Index, bei 10 nullen*)

_tTimeOut(IN := FALSE); (*Timer wird gestoppt*)
END_ACTION

ACTION WhatOrder:
(*****************************************************************

 Funktion: Diese Aktion WhatOrder steuert die zyklische
           Abarbeitung des globalen ARRAY "g_aCommunication".
           Weiter liesst sie die Modembefehle aus dem obigen
           ARRAY und gibt entsprechend die Befehlsanfragen frei.

 Autor:    R. Lichtensteiger

 *****************************************************************)
IF NOT (bCardNumber >= 1 AND
       bCardNumber <= (g_MaxI2CCard + g_ModemCmd))
THEN
  bCardNumber := 1;  (*Zuruecksetzen fuer naechsten Abfragezyklus*)
  SaveTime();       (*Aufruf fuer Antwortzeiterfassung*)
  _tTimeOut(IN := TRUE, PT := t#2000ms); (*Timer starten*)
END_IF
CASE g_aCommunication[bCardNumber].i_bModemCmd OF
  g_SPEED_43KHz, g_SPEED_28KHz, g_SPEED_17KHz, g_SPEED_9KHz,
```

```
  g_SPEED_5KHz, g_SPEED_2500Hz, g_SPEED_1300Hz:
   boSP := TRUE; (*Startvariable fuer Speed*)
  g_WRITE:
   boW := TRUE;  (*Startvariable fuer WRITE*)
  g_IDENT:
   boId := TRUE; (*Startvariable fuer IDENT*)
  g_STATUS:
   boSt := TRUE; (*Startvariable fuer STATUS*)
  g_VERSION:
   boV := TRUE;  (*Startvariable fuer VERSION*)
  g_READ:
   boR := TRUE;  (*Startvariable fuer READ*)
  ELSE
   ;
END_CASE
END_ACTION


(* @NESTEDCOMMENTS := 'Yes' *)
(* @PATH := '' *)
TYPE T_Communication :
STRUCT
   i_bModemCmd:    BYTE := 99; (*Modembefehle, siehe Globale
                               Variablen GV_I2C_Lib*)
           i_bAnzByte:     BYTE; (*Anzahl zu schreibende, sendende Bytes*)
           i_bSlaveAddress:BYTE; (*Adresse der I2C-I/O-Karte*)
   i_abDaten:      ARRAY [1..16] OF BYTE;(*zu schreibende
                                    Byte 1-16*)

   o_boOK:         BOOL;   (*Erfolgsbestaetigung*)
           o_strErrorState:STRING; (*Zustand beim Schreiben od. Lesen*)
           o_abDaten:      ARRAY [1..16] OF BYTE;(*zu lesende Byte 1-16*)

   o_boSDA:     BOOL;   (*Zustand der Datenleitung*)
   o_boSCL:     BOOL;   (*Zustand der Taktleitung*)
   o_boINT:     BOOL;   (*Zustand der Interruptleitung*)
   o_strVersion:STRING; (*Modem Version*)
END_STRUCT
END_TYPE

(* @END_DECLARATION := '0' *)

(* @NESTEDCOMMENTS := 'Yes' *)
(* @GLOBAL_VARIABLE_LIST := 'GV_I2C_Lib' *)
(* @PATH := '' *)
(* @SYMFILEFLAGS := '2048' *)
VAR_GLOBAL CONSTANT
   (*Modembefehle*)
   g_VERSION:     BYTE := 80;
   g_IDENT:       BYTE := 16;
           g_SPEED_43KHz: BYTE := 32;
           g_SPEED_28KHz: BYTE := 33;
           g_SPEED_17KHz: BYTE := 34;
           g_SPEED_9KHz:  BYTE := 35;
           g_SPEED_5KHz:  BYTE := 36;
           g_SPEED_2500Hz:BYTE := 37;
           g_SPEED_1300Hz:BYTE := 38;
   g_STATUS:      BYTE := 48;
   g_READ:        BYTE := 128; (*liest 1 Byte /
            Offset 0-15 bestimmt Anzahl zu lesende Byte (1-16)*)
   g_WRITE:       BYTE := 64;  (*schreibt 1 Byte /
            Offset 0-15 bestimmt Anzahl zu lesende Byte (1-16)*)
```

```
   (*Modemantwort*)
   g_OK:         BYTE :=192;

   (*Modemantwort Leitungsstatus*)
   g_MaskINT:     BYTE := 196;
   g_MaskSCL:     BYTE := 194;
   g_MaskSDA:     BYTE := 193;

   g_MaxI2CCard:  BYTE := 12;
   g_ModemCmd:    BYTE := 4;
END_VAR

VAR_GLOBAL
   (*Sendeaufforderung*)
   g_boSend:      BOOL;

         (*   I/O Variabeln fuer den PC-COM Port *)
         g_COMin          AT %I*  : PcComInData; (* Verknuepft zum Port vom
                          TwinCAT System Manager *)
         g_COMout AT %Q*  : PcComOutData;(* Verknuepft zum Port vom
                          TwinCAT System Manager *)
         g_TxBuffer       : ComBuffer;  (* Emfangs Databuffer; benuetzt
                    von allen empfangs Funktionsbloecken*)
         g_RxBuffer       : ComBuffer;   (* Sende Databuffer; benuetzt
                      von allen sende Funktionsbloecken*)

   (*Hardwareabbild der I2C - Topologie*)
   g_aCommunication: ARRAY [1..36] OF T_Communication;
END_VAR

(* @OBJECT_END := 'GV_I2C_Lib' *)
(* @CONNECTIONS := GV_I2C_Lib
FILENAME : ''
FILETIME : 0
EXPORT : 0
NUMOFCONNECTIONS : 0
*)
```