ibaLogic

**Programmiersoftware für Signalmanagement,
Simulation und Soft-SPS**

Handbuch

Ausgabe 4.2 de / ibaLogic V 3.90c

Messtechnik- und Automatisierungssysteme

iba

ibaLogic – Handbuch

Verfasser

iba AG

Königswarterstr. 44

D-90762 Fürth

Tel.: + 49 (0)911 9 72 82-0

Vertrieb -10

Support -14

Technik -13

FAX -33

Email: iba@iba-ag.com

Web: www.iba-ag.com

Ausgabe: Handbuch V 4.2 de / ibaLogic V 3.90c

Der Inhalt dieser Druckschrift wurde auf Übereinstimmung mit der beschriebenen Hard- und Software überprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass für die vollständige Übereinstimmung keine Garantie übernommen werden kann. Die Angaben in dieser Druckschrift werden jedoch regelmäßig aktualisiert. Notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten oder können über das Internet heruntergeladen werden.

Die aktuelle Version liegt stets auf unserer Website http://www.iba-ag.com zum Download bereit.

Für Anregungen und Verbesserungsvorschläge sind wir natürlich immer dankbar.

Version	Datum	Änderung	Kapitel / Seiten	Autor	Version ibaLogic
V 4.2	13.02.09	Neuer OPC-Server ibaLogic-V3 Runtime Parameter Commandline		kol	3.90c

Inhalt

Vorwort

Dieses kompakte Handbuch liefert Ihnen die erforderlichen Informationen für den Umgang mit der grafischen Programmiersoftware ibaLogic.

Die Bedienung der Software wird z.T. anhand typischer Beispiele exemplarisch erläutert. Im konkreten Anwendungsfall, insbesondere im Zusammenhang mit den Hardwarekomponenten für Ein- und Ausgaben, sollte die zugehörige Hardwaredokumentation hinzugezogen werden.

Die aktuellste Version dieses Handbuchs finden Sie stets auf unserer Website http://www.iba-ag.com , wo sie im Downloadbereich bereit steht.

Dieses Handbuch umfasst sieben Kapitel, die Ihnen den Umgang mit ibaLogic und dessen Eigenschaften erläutern.

Kapitel 1 Im ersten Kapitel finden Sie eine Einleitung mit Hinweisen zu den wichtigsten Leistungen von ibaLogic und zur Norm IEC 61131-3.

Kapitel 2 In diesem Kapitel wird die Bedienoberfläche mit allen Menüs und Dialogfenstern beschrieben. Die wichtigsten Einstellungen für Programm und System sind hier erläutert.

Kapitel 3 In Kapitel 3 finden Sie die praktischen Anleitungen für das Arbeiten mit ibaLogic. Von der Programmgestaltung über Verwendung von Funktionsbausteinen, Erstellen von Makros, Testen und Debuggen bis hin zum Drucken werden die Arbeitsgänge ausführlich beschrieben.

Kapitel 4 In diesem Kapitel sind alle Funktionsbausteine und Funktionen, die ibaLogic standardmäßig bietet aufgeführt und beschrieben.

Kapitel 5 In diesem Kapitel finden Sie die Beschreibung der Ein- und Ausgangsressourcen und der OPC-Kommunikation.

Kapitel 6 Dieses Kapitel behandelt die Systemvoraussetzungen und den Installationsvorgang sowie Besonderheiten, die beim Einsatz von alten ISA-Karten zu berücksichtigen sind.

Kapitel 7 Im letzten Kapitel finden Sie zusätzliche Informationen für besondere Themen, wie z.B. Programmlistings, Beispiele für spezielle Anwendungen usw.

Abschließend finden Sie noch ein Glossar, ein Literatur- und Quellenverzeichnis sowie ein Stichwortverzeichnis, die Ihnen bei der Suche nach bestimmten Themen behilflich sein sollen.

Bei der Lektüre dieses Handbuchs werden Ihnen immer wieder einige Symbole und Hervorhebungen begegnen, die im Wesentlichen folgende Aussage haben:

Warnungen und Gefahrenhinweise

Tipp oder Beispiel als hilfreicher Hinweis oder Griff in die Trickkiste, um sich die Arbeit ein wenig zu erleichern.

Hinweis, wenn es etwas Besonderes zu beachten gibt, wie z.B. Ausnahmen von der Regel usw.

Verweis auf ergänzende Dokumentation oder weiterführende Literatur

Software auf der Auslieferungs-CD

Hier finden Sie Hinweise zu Beispielprogrammen, die zusammen mit ibaLogic auf der Auslieferungs_CD vorhanden sind oder weiterführender Software.

Wenn im Text von einem "Mausklick" die Rede ist, dann ist damit stets die linke Maustaste gemeint. Sollte einmal die rechte Maustaste benutzt werden, dann weisen wir explizit darauf hin.

Das Programm ibaLogic kann nur mit den Betriebssystemen MS Windows® NT 4.0, MS Windows® 2000 oder MS Windows® XP verwendet werden. Im Handbuch wird stellvertretend diese Systeme der Begriff Windows verwendet, wenn sich eine Aussage auf alle drei Systeme bezieht.

Windows NT, Windows 2000 und Windows XP sind eingetragene Warenzeichen der Microsoft Corporation.

1 Willkommen zu ibaLogic

1.1 Einleitung

ibaLogic kombiniert die Eigenschaften eines komfortablen Signalmanagers mit denen einer leistungsstarken Soft-SPS. ibaLogic wird vor allem in der Mess- und Regelungstechnik für schnelle und dynamische Prozesse, wie z.B. für Antriebsregelungen eingesetzt. Daher sind kurze Programmzykluszeiten (ab 1ms) und ein deterministisches Zeitverhalten wesentliche Systemeigenschaften.

Die ganz besonderen Highlights von ibaLogic sind aber neben der sehr einfachen Bedienbarkeit die ausschließliche Verwendung internationaler Standards bei Betriebssystem, Kommunikation und Programmiersprache, womit die Offenheit, Portierbarkeit und Wiederverwendbarkeit der mit ibaLogic erstellten Anwendersoftware sichergestellt wird.

Standard-PCs mit Windows® als Betriebssystem sind die Hardwareplattform für ibaLogic. Dies bedeutet, alle aktuellen und zukünftigen Weiterentwicklungen auf dem PC-Sektor, wie z.B. Internetanbindung, Remote Access usw. sowie die laufenden Performancesteigerungen der in den PCs eingesetzten Prozessoren kommen ibaLogic automatisch zugute.

Die Projektierung von ibaLogic ist sehr anwenderfreundlich und erfolgt mittels grafischer Werkzeuge. Die Anforderungen der IEC 61131-3 Standard-Programmiersprache für Soft-SPS werden von ibaLogic selbstverständlich voll erfüllt. Dies geschieht nicht nur aus Gründen der Portierbarkeit, der einfachen Einarbeitung oder aus Marketinggesichtspunkten heraus. ibaLogic nutzt konsequent alle Möglichkeiten und Datenformate der IEC 61131-3, wie z.B. “Structured Text (ST)“ als Meta-Sprache oder “String“ als konvertierbares Datenformat und eröffnet so dem Anwender Möglichkeiten bei der Programmgestaltung und bei Problemlösungen, welche weit über das Maß bisheriger Steuerungen hinausreichen.

Die variable Prozessanbindung und die Offenheit bei der Kommunikation zur Außenwelt sind zwei der Stärken von ibaLogic. Sensoren und Aktoren werden entweder über international genormte Feldbussysteme, wie z.B. Profibus angebunden oder bei Steuer- und Regelungsaufgaben mittels schneller PADUs (Parallel Analog Digital Umwandler) bzw. ibaNet750 IO-Komponenten von WAGO/Beckhoff direkt ein- oder ausgekoppelt. Die offene Kommunikation zwischen ibaLogic Funktionseinheiten zur Visualisierung oder zu anderen übergeordneten Rechnersystemen erfolgt mittels standardisierter OPC-Schnittstelle über TCP/IP oder „Named Pipes“.

Mit ibaLogic-V3-Runtime steht eine preisgünstigere Variante von ibaLogic-V3 zur Verfügung, bei der nur das Laufzeitsystem, aber kein Editor benötigt wird.

ibaLogic wird für folgende Anwendungen genutzt:

Schnelle Signalvorverarbeitung und -verteilung

- Signalmanagement und –vorverarbeitung für ibaQDA, ibaPLR oder ibaVision
- Signalaufbereitung und komplexe Trigger-Generierung für ibaPDA, ibaQDR, ibaPLR und ibaQDA
- Schnelle Signalrangierung von ibaLogic zu verschiedensten Anwendungen (z.B. ibaQDA oder vom Anwender selbst erstellte Visual C++ oder Visual Basic Programme)

1

Soft SPS entsprechend IEC 61131-3

- PC-basiertes Automatisierungssystem für Windows mit ibaLogic als Soft-SPS der oberen Leistungsklasse.
- Aufgrund der einfachen Handhabung und intuitiven Bedienung, der flexiblen Signaleinkopplung und der integrierten, schnellen Online-Beobachtung eignet sich ibaLogic auch hervorragend zur Modernisierung bestehender Anlagen. Vorhandene Programmquellen sind, sofern sie in Structured Text geschrieben wurden, unter ibaLogic wiederverwendbar.

Signalverarbeitung

- Condition Monitoring System für Maschinen
- Schwingungsanalyse für Maschinen, mit Signalabtastraten von bis zu 25 kHz / Kanal
- Lagerüberwachung und Alarmgenerierung
- Neue Möglichkeiten bei Qualitätsdatenaufzeichnung und –überwachung
- Direktes Abspeichern von ibaLogic-Variablen in Messdaten-Files (*.dat) und das Rücklesen dieser Daten (Playback)

Simulatoren

- Simulation von Walzgerüsten, z.B. zu Trainingszwecken
- Komplette Anlagensimulation, z.B. zum Test von Steuerungen oder Regelungen in anderen Automatisierungsgeräten

IEC 61131-3 Software-Entwicklungs-Paket

- Zielsystemunabhängige Programmiersprache auf Basis IEC 61131-3-Standards (ST)

1.2 Kurzübersicht der ibaLogic Systemeigenschaften

- ❑ Schnellste Programmzykluszeiten, ab 1ms.
- ❑ Deterministisches Zeitverhalten (Echtzeitverhalten) unter Windows.
- ❑ Benutzerfreundlich durch Windows-"Look and feel", sehr einfach erlern- und bedienbar. Grafische Projektierung mit Autorouting-Unterstützung.
- ❑ Schnelle Reaktionszeit bei Bedienereingaben oder Programmänderungen (turn around Zeit), diese Aktionen werden ohne Compilierungsläufe sofort ausgeführt. Erfolgen diese Änderungen im Online-Layer, dann wirken sie sich direkt auf den Prozess aus (als wenn Sie einen Schaltschrank unter Spannung verdrahten).
- ❑ HOT SWAP Umschaltmöglichkeit, d.h. während Sie Änderungen an einem Funktionsplan vornehmen, kann die bestehende Version noch gleichzeitig aktiv den Prozess steuern. Nach erfolgreicher Änderung wird stoßfrei vom alten auf den neuen Funktionsplan umgeschaltet. Diese Eigenschaft ist besonders bei kontinuierlich laufenden Prozessen (z.B. Papierindustrie) vorteilhaft.
- ❑ Programmiersprache, Datenformate und Bausteinbibliothek entsprechend internationalem Standard IEC 61131-3.
- ❑ Die folgenden Datentypen werden unterstützt:
  Bool, Integer (16 Bit, 32 Bit, Unsigned 32 Bit), DWord (32 Bit), Float (32 Bit und 64 Bit), String, Time und Array(4-dimensional, von vorgenannten Datentypen außer String, homogen)
- ❑ Umfangreiche Bausteinbibliothek mit zahlreichen Standard- und Sonderfunktionen (FBs) ist vorhanden. Einfache Erweiterung der Bibliothek durch den Anwender ist möglich.

- ❑ Zwei Wege zur interaktiven Erstellung neuer Funktionsbausteine (FBs)
  - Einfach ohne Programmierkenntnisse durch Verwendung mathematischer Formeln
  - Erweiterung der o.g. Methode unter Verwendung von “Structured Text” Metasprache. Damit sind “if-then-else” Abfragen, “for-next” Schleifen usw. möglich.
- ❑ Programmstrukturierung mittels Makrobausteinen (MBs) mit beliebiger Verschachtelungstiefe; einfache Erstellung von MBs durch markieren und zusammenfassen mehrerer Programmbausteine
- ❑ Offene DLL-Schnittstelle zur Einbindung von bestehendem Know-how, z.B. in Form von “C” oder “C++” Programmcode.
- ❑ Volle Unterstützung eines hierarchischen Designs durch die uneingeschränkte Möglichkeit zur Erzeugung und Einbindung von Makros.
- ❑ Unterstützung von “Multitasking” und Task-Task-Kommunikation.
- ❑ Voll integriertes Produkt, d.h. alle notwendigen Werkzeuge und Compiler (ST, C++, Assembler) sind in ibaLogic integriert; einfache Installation und Handhabung.
- ❑ I/O Prozessanbindung für folgende Systeme:
  - Eingang (typisch: 1ms) von Analog- / Digitaleingängen über Lichtwellenleiter mittels FOB I/O und FOB 4i-PCI (unidirektional) an PADU8/16/32.
  - Eingabe schneller Analog- und Digitaleingänge bis 25 kHz/Kanal mit FOB I/O-PCI oder FOB 4i-PCI im "FOB-M – Modus" (unidirektional) an PADU8 ICP / PADU M.
  - Ein-/Ausgabe (typisch: 1ms) von Analog- und Digitaleingängen sowie Analog- undDigitalausgängen über Lichtwellenleiter mittels FOB-IO Anschaltung (bidirektional) an PADU8/16/32 und Padu8-O, SLM.....
  - Ein-/Ausgabe von Analog- und Digitaleingängen sowie Analog- und Digitalausgängen über Lichtwellenleiter mittels FOB-IO und ibanet750-Anschaltung (bidirektional)an WAGO- Klemmen (I/O Verzögerung Modulspezifisch siehe Datenblatt des I/O Modules, Image-Kopie vom WAGO-Kopf: typ. 1 ms).
  - Diverse Ankopplungen an eingeführte Feldbus- oder Rückwandbussysteme (Profibus-DP, VME-Bus, MMC/S5 u.a.).
  - Reflective Memory (VMIC)
  - Diverse Ankopplungen an marktgängige Steuerungssysteme der Firmen ABB, ALSTOM, SIEMENS, SMS-Demag, KVÆRNER, PROSOFT, ALLEN-BRADLEY u.v.a.
- ❑ Ein- Ausgaberessourcen, offene Kommunikation
  - TCP/IP über „named pipes” (Ein- und Ausgänge) zu PCs und SPS-Systemen und CSV (Comma Separated Value)-Dateien (z.B. MS Excel o.a. Programme)
  - OPC Schnittstelle zu Standard Visualisierungs- (HMI) Systemen.
  - TCP/IP Kommunikation zu verteilten PDA/QDA und / oder ibaLogic Soft SPS-Anwendungen
  - SIMATIC S7 über L2B (Profibus DP Slave Anschaltung, uni- und bidirektional).
  - SIMATIC S5 oder MMC216 über SM64-IO Anschaltung, uni- und bidirektional.
  - Serielle Schnittstelle mit 3964R Protokoll (z.B. Siemens-Prozessrechner der R- und M-Serien).
  - Simatic TDC Schnittstelle FOB TDC zum GDM (bidirektional)
  - Simadyn-D Schnittstellenbaugruppe FOB SD zur Rahmenkopplung CS12/13/14
  - ALSTOM HPC (Logidyn D1, D2) über VME-interface SM128V und FOB...
  - Systemkopplungen zu CAN, DeviceNet, ControlNEt und Profibus Master in Vorbereitung

1

1.3 Die SPS-Sprachen nach IEC 61131-3

Die SPS-Sprachen vor Einführung der IEC 61131-3 waren nicht einheitlich und oft nur auf dem Gerätetyp eines Herstellers ablauffähig. Damals typische SPS-Sprachmittel, wie z.B. AWL, waren zu ineffektiv; viele Probleme hätten mit einer Hochsprache viel schneller gelöst werden können. Darüber hinaus war die Einarbeitung, z.B. des Instandhaltungspersonals in bestehende SPS-Programme außerordentlich mühsam. Das Fehlen von lokalen Speicherbereichen und einer symbolischen Adressierung führte zu Fehlern, nach denen man lange suchen musste.

Diese Defizite waren einer der Gründe zur Definition des Teils 3 der IEC 61131-Norm. Er vereinheitlicht die alten Sprachen, rundet sie ab und ergänzt sie um die neue Sprache (Structured Text "ST"). Der Standard legt aber nicht nur die Anweisungen fest, mit denen Programme erstellt werden können, er erlaubt darüber hinaus die vollständige Beschreibung der Architektur eines SPS-Systems aus Softwaresicht.

Mit den neuen Sprachen kann eine komplette SPS einschließlich SW-HW Zuordnung beschrieben werden. Die neuen Sprachelemente heißen *Konfiguration* (*configuration*), *Ressource* (*resource*) und *Task* (*task*). Auf Programmebene gibt es die Elemente *Programm* (*program*), *Funktionsbaustein* (*function block*) und *Funktion* (*function*).

1.3.1. IEC 61131-Software Modell

Aussage der Norm mit Beispielen:

- Ein Automatisierungssystem besteht aus einer oder mehreren *Konfigurationen* (*configuration*), die miteinander kommunizieren können. Eine Konfiguration ist z.B. eine SPS mit Rahmen und Baugruppen oder ein ibaLogic-PC.
- Eine *Konfiguration* besteht aus einer oder mehreren *Ressourcen* (*resource*). Eine Ressource ist einer CPU zugeordnet. Eine CPU kann mehrere Ressourcen haben. ibaLogic kennt eine Ressource pro PC und nennt diese "*Projekt*" (*layout*). Projekte werden entweder in einer *.lyt oder einer *.txt-Datei (als Structured Text) abgespeichert.
- Einer *Ressource* (Projekt) können eine oder mehrere *Tasks* zugeordnet werden. Wesentlich an einer Task ist ihr Zeitverhalten. Dieses Zeitverhalten kann explizit beschrieben werden. In einer Task werden zeitlich zusammengehörige Aufgaben zusammengefasst, z.B. alle Aufgaben, die im 20ms Takt aktiviert werden müssen.

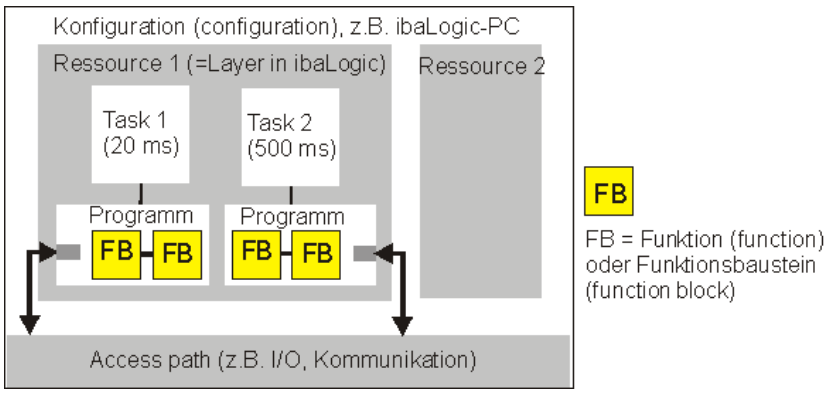


Bild 1 IEC 61131-3 Software-Modell

1.3.2. IEC 61131-Programm Organisations-Einheiten (POUs)

Programm Organisations-Einheiten (Program Organization Units POUs) sind laut IEC 61131 *Funktionen* (functions), *Funktionsbausteine*, (function blocks) und *Programme* (programs). Allgemein gilt für alle POUs, dass sie nicht rekursiv sein dürfen, d.h. sie dürfen sich nicht selbst aufrufen.

- *Funktionen* sind Unterprogramme, die beliebig viele Eingangsparameter haben und genau ein Ergebnis zurückliefern (Beispiel ist die Funktion sin(x)). Funktionen liefern bei gleicher Eingangsbeschaltung stets das gleiche Ergebnis (sie besitzen kein Gedächtnis).
- *Funktionsbausteine* haben beliebig viele, klar definierte Ein-, und Ausgangsparameter und können interne Variablen verwenden, d.h. sie besitzen ein Gedächtnis. Ein PID-Regler ist ein gutes Beispiel für einen Funktionsbaustein. Er kann in der gleichen Task oder von verschiedenen Tasks mehrfach und mit jeweils anderen Datensätzen genutzt werden.
- *Programme* beinhalten die Verschaltung von Funktionen und Funktionsbausteinen. Ein Programm kann in jeder der von IEC 61131 definierten Programmiersprache geschrieben werden. Die Programme sind explizit einer Task mit einem bestimmten Zeitverhalten zugeordnet.

1.3.3. Unterstützte Datentypen

Die folgenden elementaren Datentypen werden von ibaLogic unterstützt:

Typ	Bereich (min)	Bereich (max)	Bemerkung
BOOL	*0 (FALSE)*	1 (TRUE)	
INT	-32_768	32_767	16-bit Integer (mit Vorzeichen)
DINT	-2_147_483_648	2_147_483_647	32-bit Integer (mit Vorzeichen)
UDINT	0	*4_294_967_295*	32-bit Integer (ohne Vorzeichen)
DWORD	16#0000_0000	16#FFFF_FFFF	32-bit Wort (ohne Vorzeichen)
REAL	1.175_494_351 e-38	3.402_823_466 e+38	Gleitpunkt, einfache Genauigkeit, 32 bit
LREAL	2.225_073_858_507_201_4 e-308	1.797_693_134_862_315_8 e+308	Gleitpunkt, doppelte Genauigkeit, 64 bit
TIME	-922_337_203_685_477_580.8 ms	922_337_203_685_477_580.7 ms	Zeit, intern abgebildet als 64-bit Integer (mit Vorzeichen) mit 0.1ms Auflösung per increment
STRING	0	1024 Zeichen	Zeichenfolge mit Anzahl der Zeichen incl. Ende-Kennung (NULL).
ARRAY	Struktur, bestehend aus einer beliebigen Folge eines der o.g. Datentypen (mit Ausnahme des String, der bereits ein Array darstellt); maximal vierdimensional maximale Anzahl Elemente: 1048576		

Tabelle 1 Unterstützte Datentypen

2 Bedienung und Einstellungen

2.1 ibaLogic starten

2.1.1. ibaLogic-V3

falls Sie ibaLogic noch nicht auf Ihrem PC installiert haben, schlagen Sie bitte in Kapitel 6.1 nach und verfahren Sie entsprechend. Dort werden Ihnen detailliert die ersten Schritte zur Installation von ibaLogic beschrieben.

ibaLogic starten Sie ganz einfach im Windows-Explorer über einen Doppelklick auf die Datei ibaLogic*version*.exe. Je nach individueller Installation kann ibaLogic auch über eine Verknüpfung auf dem Desktop oder über die Windows-"Start-leiste" gestartet werden.

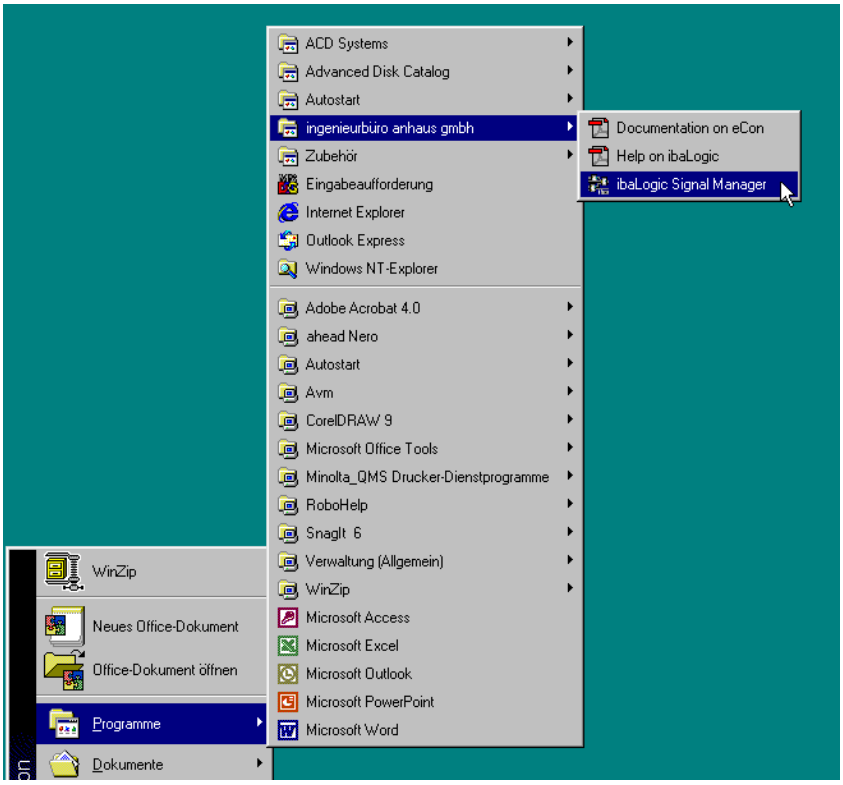


Bild 2 Start von ibaLogic

Wenn ibaLogic ohne Kopierschutz (Dongle) gestartet wird, dann erscheint zunächst eine Meldung, die einige alternative Möglichkeiten zum Start von ibaLogic, auch ohne Dongle, anbietet:

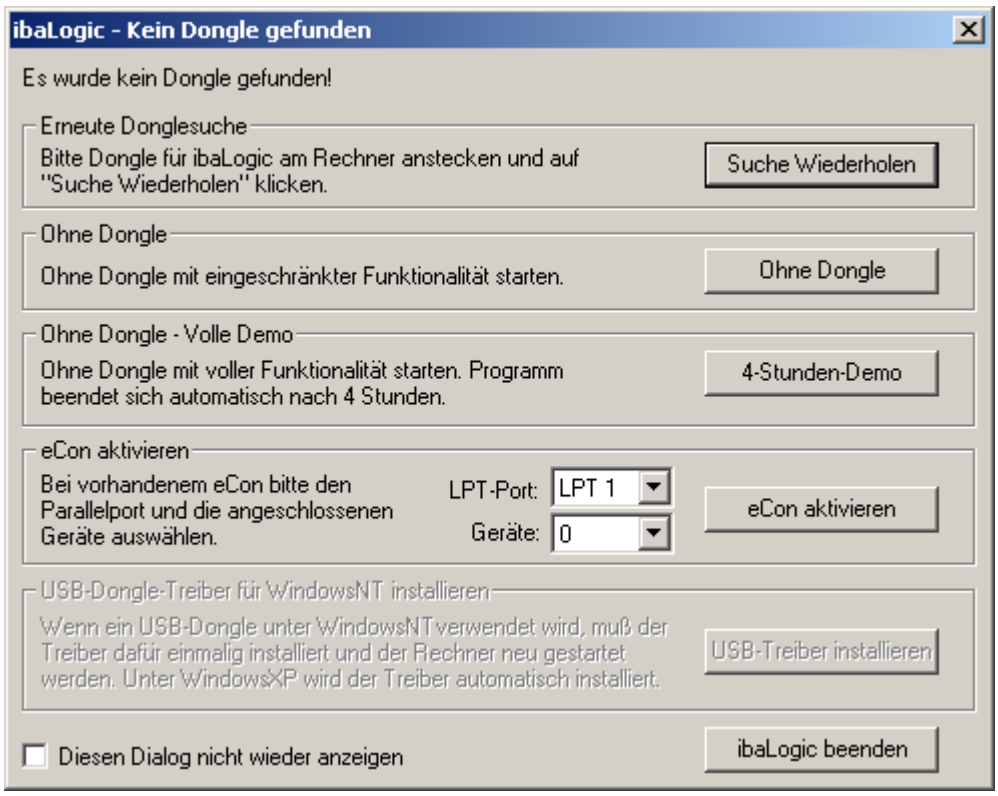


Bild 3 Start von ibaLogic ohne Dongle

Sollte der Dongle nur vergessen worden sein, stecken Sie ihn einfach auf und klicken Sie auf "Suche Wiederholen".

Wenn keine Online-Funktion und kein Playback erforderlich ist, kann ibaLogic auch ohne Dongle gestartet werden.

ibaLogic wird mit voller Funktionalität gestartet. Das Programm beendet sich automatisch nach 4 Stunden.

Wenn ibaLogic für den Betrieb eines **eCons** verwendet werden soll, ist auch kein Dongle erforderlich.

Wenn Sie unter Windows NT arbeiten und einen USB-Dongle verwenden wollen, jedoch noch nicht die USB-Treiber installiert haben, dann können Sie dies hier nachholen. Anschließend einfach noch einmal starten. Unter Windows XP ist die Funktion deaktiviert.

Nach dem Start erscheint die ibaLogic Bedienoberfläche mit folgenden Bereichen:

- ❑ Menüleiste
- ❑ Symbolleiste
- ❑ Ressourcenbereich mit Ressourcenauswahl
- ❑ Tasks mit Taskauswahl
  Jede Task verfügt über eine Eingangs- und Ausgangsrandleiste sowie den Arbeitsbereich.

2.1.2. ibaLogic-V3-Runtime

Bei ibaLogic-V3-Runtime handelt es sich um ein abgespecktes ibaLogic-V3, bei dem es keine Editionsmöglichkeit des Programms gibt. Die Runtime muss mit ibaLogic-V3 erstellt werden und auf den Rechner kopiert werden, auf dem sie laufen soll.

Die Datei „autostart_runtime.lyt" muss mit einem ibaLogic-V3-System erstellt werden. Dabei müssen die Versionsnummer von ibaLogic-V3-Runtime und ibaLogic-V3 übereinstimmen.

Damit die Runtime laufen kann, muss ibaLogic-V3-Runtime auf dem betreffenden Rechner installiert werden.

Die Vorgehensweise für die Installation von ibaLogic-V3-Runtime entspricht der Installation von ibaLogic-V3.

Bei der Erstellung der Runtime-Datei ist darauf zu achten, dass der Name „autostart-runtime.lyt" verwendet wird. Die Datei „autostart_runtime.lyt" ist in das Verzeichnis „…\schematics\", auf dem Rechner auf der sich ibaLogic-V3-Runtime befindet, zu kopieren.

Beim Start von ibaLogic-V3-Runtime wird automatisch die Lyt-Datei „autostart_runtime.lyt" gesucht und gestartet. Sofern diese Datei von ibaLogic-V3-Runtime nicht gefunden wird, öffnet sich folgende Fehlermeldung.

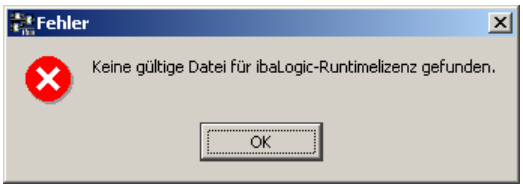


Bild 4 Fehlermeldung ibaLogic-V3-Runtime

Die ibaLogic-V3-Runtime wird mit einem Doppelklick auf das Icon im Desktopbereich gestartet.

Im Statusfenster werden der Verlauf und der Status der Runtime aufgelistet.

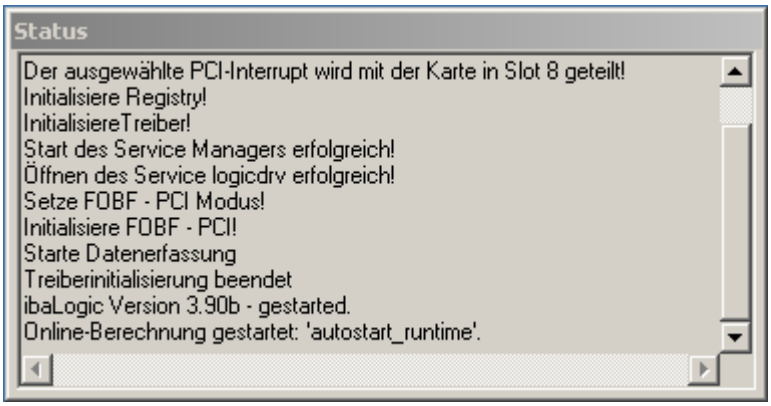


Bild 5 Statusfenster ibaLogic-V3-Runtime

Mit einem Klick der rechten Maustaste auf die Runtime in der Taskleiste wird das folgende Kontextmenü aufgerufen.

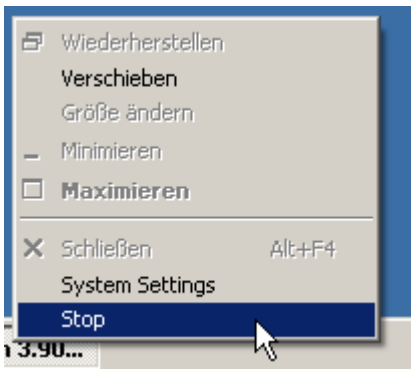
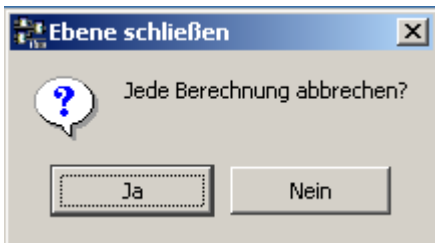


Bild 6 Kontext-Menü ibaLogic-V3-Runtime

Aus diesem Kontextmenü werden die Systemeinstellungen geöffnet oder die Runtime gestoppt und beendet.

Beim Aufruf der Systemeinstellung erfolgt eine Abfrage, ob die laufende Berechnung der Runtime angehalten werden soll oder nicht.



Sofern die Berechnungen weiterlaufen, ist nur eine Ansicht der Systemeinstellungen möglich. Werden die Berechnungen angehalten, können die Systemeinstellungen konfiguriert werden.

Nach dem Schließen der Systemeinstellungen werden die Berechnungen erneut gestartet, bzw. laufen normal weiter.

Eine detaillierte Beschreibung der Systemeinstellungen finden Sie Kapitel 2.5 Systemeinstellungen.

2.1.3. ibaLogic mit der Kommandozeile starten

ibaLogic-V3 kann auch mit einer (DOS-)Kommandozeile gestartet werden. Damit lässt sich der Programmaufruf auch mit Batchdateien oder aus anderen Programmen heraus, z.B. einer Visual Studio-Applikation ausführen.

Als Besonderheit beim Starten über die Kommandozeile können verschiedene Parameter übergeben werden, die ibaLogic-V3 dazu veranlassen unterschiedlich zu starten.

Syntax der Kommandozeile

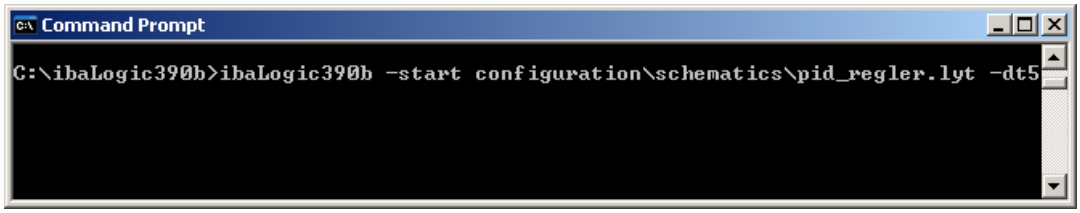


Bild 7 Kommandozeileninterpreter

C:\ibaLogicXXX>ibaLogicXXX –start

- **ibaLogic-V3-Runtime:** startet die Runtime-Datei automatisch mit der Datei „autostart_runtime.lyt“.
- **ibaLogic-V3:** startet ibaLogic-V3 mit einem leeren Arbeitsblatt.

C:\ibaLogicXXX>ibaLogicXXX –start configuration\schematics\Datei.lyt

- **ibaLogic-V3: s**tartet ibaLogic-V3 mit der Datei.lyt und sperrt das Arbeitsblatt

C:\ibaLogicXXX>ibaLogicXXX –start -dt

Zusätzlich kann mit dem Parameter –dt ein Defaultwert für die Basis-Zykluszeit voreingestellt werden. Dies ist dann notwendig, wenn zum Beispiel ibaLogic-V3 zum ersten Mal aufgerufen wird.

XXX = Versionsnummer

2.2 ibaLogic Bedienoberfläche

Nach dem Start erscheint die ibaLogic-Bedienoberfläche:

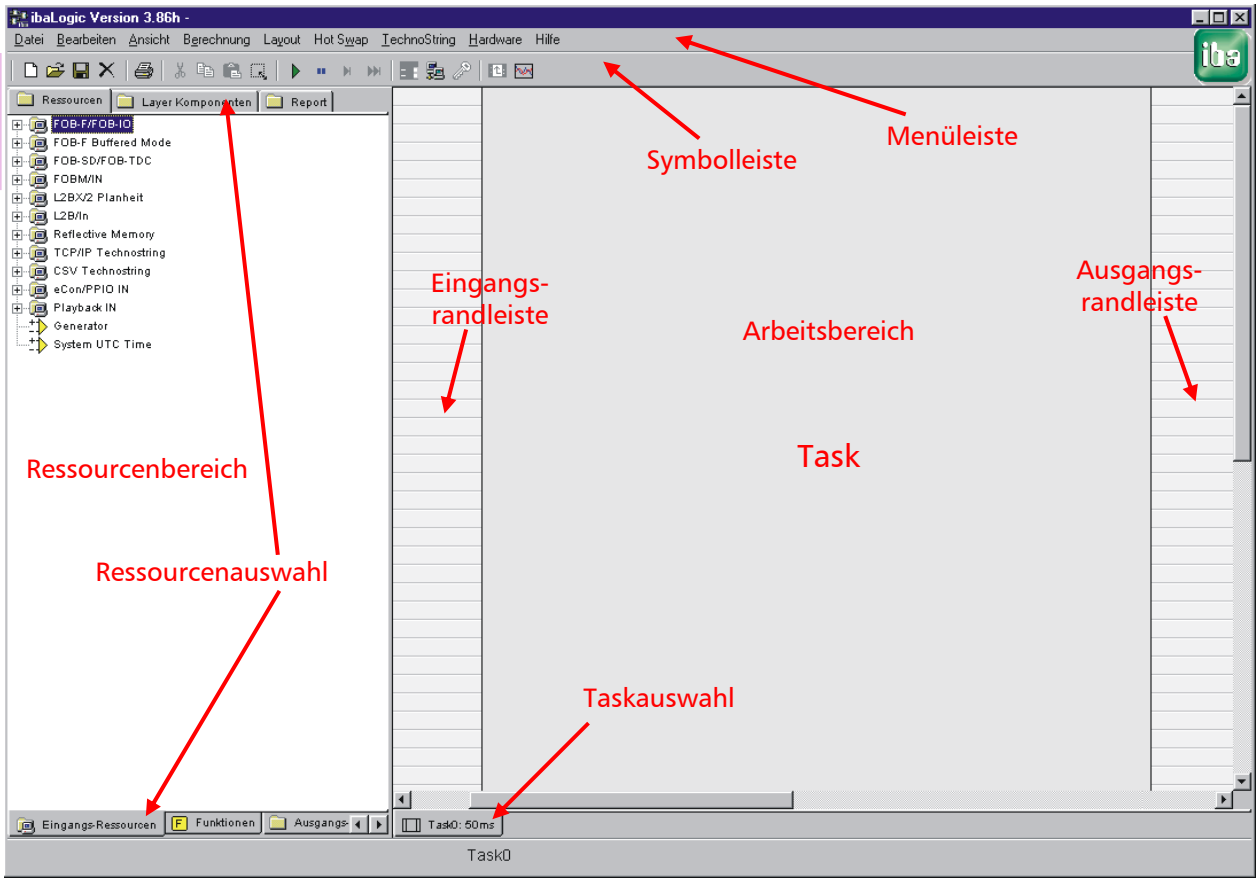


Bild 8 ibaLogic Bedienoberfläche (Standardbildschirm)

Im oberen Bildbereich finden Sie die von Windows-Anwendungen bekannte Menüleiste (mit Drop-down Menüs) sowie die Symbolleiste mit der symbolischen Kurzanwahl häufig benutzter Befehle. Die Kommandos der Menü- und Symbolleiste werden im folgenden Kapitel erläutert.

ibaLogic verwendet ein zweigeteiltes Anwendungsfenster. Im linken Bereich befindet sich der *Ressourcenbereich*. In diesem Bereich gibt es drei unterschiedliche Ansichten, die mit den entsprechenden Schaltflächen (Reitern) am oberen Rand des Ressorcenbereiches angewählt werden können: Ressourcen, Hierarchie-Sicht und Report. Ist eine Ansicht gewählt, erscheinen am unteren Rand des Ressourcenbereiches ggf. weitere, dazu passende Optionen.

Die Ressourcen sind in drei Klassen eingeteilt, den Eingangsressourcen "Eingangs-Ressourcen", den Funktionsbausteinen "Funktionen" und den Ausgangsressourcen "Ausgangs-Ressourcen". Über die *Ressourcenauswahl* kann die gewünschte Klasse ausgewählt werden.

Die Ressourcen (z.B. Analogeingang 1 von Modul 1 der FOB/FOB-F Schnittstellenkarte) werden mittels Mausklick angewählt und per "drag and drop" zur *Eingangsrandleiste*, in den *Arbeitsbereich* oder zur *Ausgangsrandleiste* der angewählten *Task* verschoben. Zur größeren Darstellung des *Arbeitsbereichs* kann der *Ressourcenbereich* durch die Menübefehle ↪*Ansicht* ↪*Nichts* aus- bzw. eingeblendet werden.

Unter „*Layer Komponenten*“ finden Sie in drei verschiedenen Formen eine Übersicht der im aktuellen Projekt verwendeten Ressourcen. Die Übersicht "Hierarchie" zeigt für jede Task in Form einer Baumstruktur die unterschiedlichen Ressourcen getrennt nach ihrer Art an:

FB und Macros, Inputs, Outputs, Off Task Inputs, Off Task Outputs und Intra Page (Konnektoren). Zum leichten Auffinden der Signale oder Funktionen im Plan muss nur die gewünschte Ressource im Baum angeklickt werden und schon wird die entsprechende Stelle im Funktionsplan angezeigt und die Ressource markiert. In der Übersicht "Objekte" sind alle im Projekt verwendeten Objekte nach Klassen sortiert aufgelistet. Klickt man auf das kleine Kreuz vor jedem Objekt, dann werden die Tasks angezeigt, in denen das betreffende Objekt verwendet wird, wobei die Objekte nach Typen alphabetisch sortiert sind. Die dritte Übersicht "Instanzen" entspricht der vorhergehend beschriebenen, nur, dass die Objekte nach den Instanznamen alphabetisch sortiert sind.

Die „*Report*“-Ansicht bietet zwei weitere Möglichkeiten zur Auswahl: *Evaluierungsreihenfolge* und *Rückkopplungen*.

Auch hier sind die einzelnen Tasks in einer Baumstruktur dargestellt. Unterhalb der Tasks sind jeweils alle darin verwendeten Funktionen untereinander aufgeführt und zwar in der Reihenfolge, wie Sie beim Programmablauf bearbeitet werden. Der oberste Funktionsbaustein wird zuerst, der unterste zuletzt bearbeitet. Das Wissen um die Abarbeitungsreihenfolge ist bei komplexen und verschachtelten Programmen mitunter sehr wichtig, um etwaige Fehlfunktionen erklären zu können. Auch hier springt die Funktionsplandarstellung automatisch zu der betreffenden Funktion, wenn diese in der Baumstruktur angeklickt wird.

In der Ansicht Rückkopplungen werden Rekursionen oder Endlosschleifen angezeigt, wenn sie vorhanden sind. Auch hier werden die Rückkopplungen in einer Baumstruktur dargestellt, mit all den Funktionen, die im Rückkopplungskreis liegen. Wie bei den vorigen Punkten bereits beschrieben, werden die entsprechenden Funktionen bei Mausklick markiert und angesprungen.

Die vom Anwender erstellten Programme werden durch “Tasks“ bearbeitet. Jede Task hat eine festgelegte Zykluszeit (z.B. 50ms). Die Taskzykluszeit wird in den Taskanwahl-Buttons angezeigt. Über die *Taskauswahl* kann zwischen den verschiedenen Tasks umgeschaltet werden. Die Summe aller Tasks wird in einem Arbeitspaket (Projekt) zusammengefasst und entweder in einer *.lyt-Datei oder einer Structured Text (ST)-Datei (.txt) gespeichert.

Sobald ibaLogic in den “Berechnungs“ (Offline-Berechnung) oder “Online“-Modus geschaltet wird, erscheint in der unteren linken Ecke der Bedienoberfläche das *Berechnung [%]*-Fenster (im o.g. Beispiel nicht dargestellt). In diesem Fenster wird die aktuelle verbrauchte Task-Rechenzeit in Relation zur jeweils zugehörigen Taskzykluszeit angezeigt.

2

2.2.1. Symbolleiste

In der Symbolleiste von ibaLogic stehen folgende Kurzbefehle zur Auswahl:

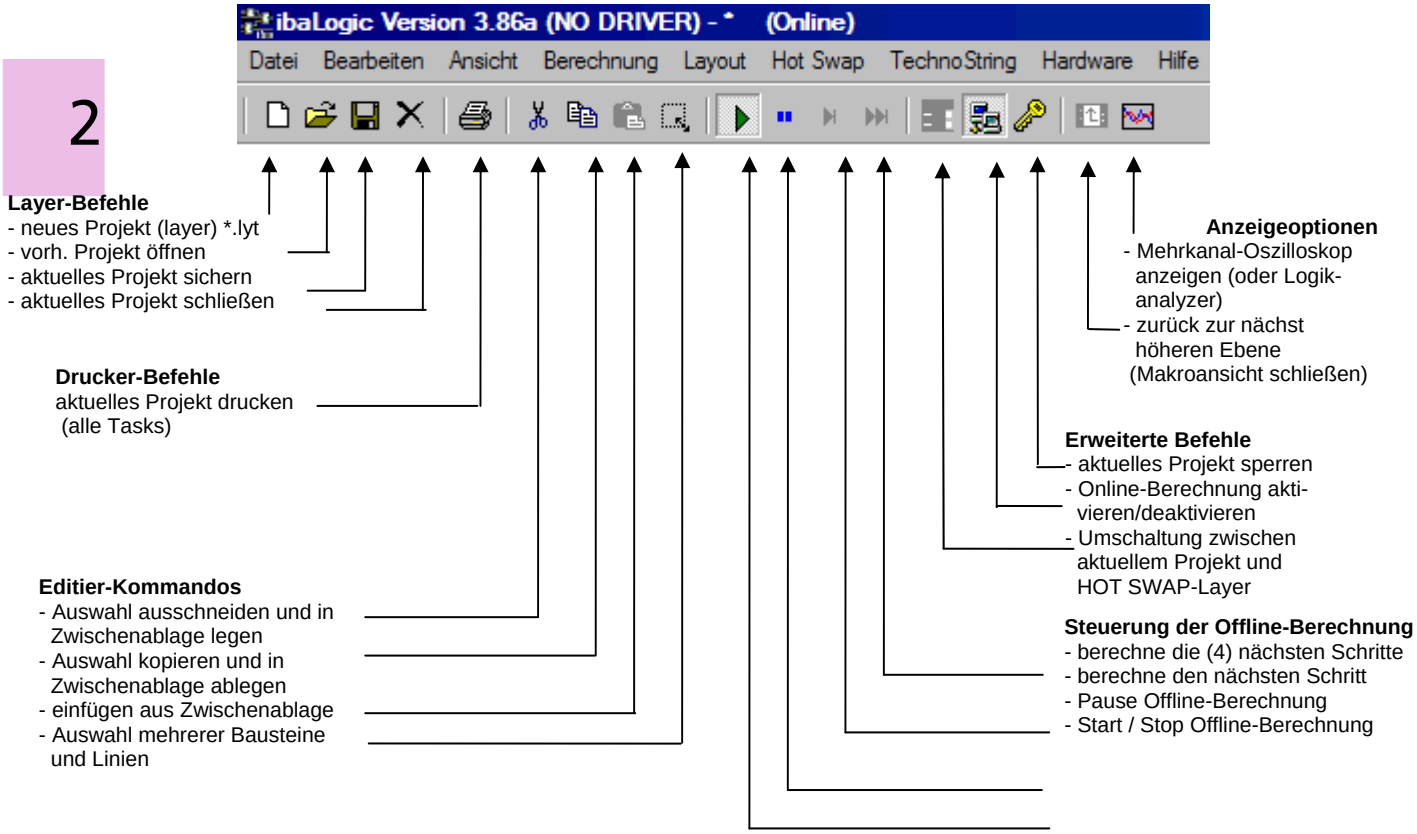


Bild 9 Symbolleiste

2.2.2. Hot Keys

Tastenkombination	Funktion
<Strg>+<A>	Vorhandenes Layout öffnen (*.txt)
<Strg>+<Backspace>	Eine Ebene zurück (im Makro, nach oben)
<Strg>+<C>	Markiertes Objekt in die Zwischenablage kopieren.
<Strg>+<M>	Mehrfachauswahl aktivieren (anschließend mit Maus Objekte umreißen
<Strg>+<N>	Neues Layout erzeugen
<Strg>+<O>	Vorhandenes Layout öffnen (*.lyt)
<Strg>+<P>	Aktuelles Layout drucken
<Strg>+<Q>	Berechnung abbrechen
<Strg>+<S>	Aktuelles Layout speichern
<Strg>+<V>	Inhalt der Zwischenablage einfügen
<Strg>+<X>	Markiertes Objekt ausschneiden und in die Zwischenablage kopieren.
<Alt>+<Eingabe>	Markiertes Objekt ändern
<Alt>+<I>	Einzelschritt in Berechnung
<Alt>+<L>	Online-Ebene sperren / freigeben

Tastenkombination	Funktion
<Alt>+<M>	Mehrfachschritt in Berechnung
<Alt>+<O>	Online / Offline-Umschaltung
<Alt>+<P>	Pause der Berechnung
<Alt>+<R>	Berechnung rücksetzen und neu starten
<Alt>+<S>	Start / Stopp der Berechnung
<Entf>	Markiertes Objekt löschen

Tabelle 2 Tastenbedienung (Abkürzungen)

2.2.3. Kombinationen aus Maus- und Tastenbedienung

LM = Linke Maustaste. RM= Rechte Maustaste

Taste	Maus	Funktion
	LM (Klick)	Markieren eines Objektes im Arbeits- oder Ressourcen-bereich
<Strg>+	LM (Klick)	Markieren eines weiteren Objektes im Arbeits- oder Ressourcenbereich (sukzessive); bei Markierung miteinander verbundener Objekte werden die Verbindungslinien auch markiert.
<Shift>+	LM (Klick)	Markieren eines weiteren Objektes im Arbeits- oder Ressourcenbereich (sukzessive); bei Markierung miteinander verbundener Objekte werden die Verbindungslinien auch markiert.
<Alt>+	LM (Klick)	Verbindungslinien aufbrechen und durch IntraPage-konnektoren ersetzen; Mauszeiger muss auf die betreffende Linie zeigen.
	LM (Doppelklick)	Auf Funktionsbaustein: öffnen des Funktionsbausteins Auf Symbolname: ändern des Namens
	LM (halten)	verschieben des sichtbaren Ausschnitts des Arbeitsbereiches, wenn Mauszeiger im Arbeitsbereich (Mauszeiger ändert sich zum Kreuzpfeil)
	LM (halten)	Auswahl eines oder mehrerer Objekte im Arbeitsbereich durch Umreißen und Verschieben eines markierten Objektes / einer Gruppe
	LM (halten)	Verändern von Verbindungslinien, wenn Mauszeiger Kreuzpfeil zeigt (an Eckpunkten)
	LM (halten)	Erweitern des Arbeitsbereiches um eine weitere Seite nach rechts oder nach unten; Mauszeiger muss auf den ganz rechten oder untersten Seitenrand positioniert werden (ändert sich in Doppelpfeil-Symbol), dann über die Grenze hinausziehen.
	RM	Aufruf des Kontextmenüs, wo vorhanden, z.B. im Arbeitsbereich oder auf den Reitern der Taskleiste

Tabelle 3 Kombinierte Maus- und Tastenbedienung

2.3 ibaLogic Menüleiste

2.3.1. "Datei"-Menü

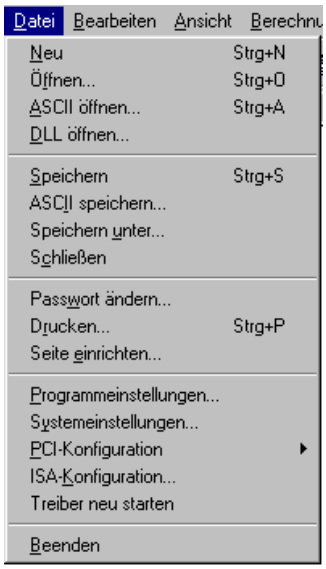


Bild 10 Menü Datei

❑ ***Dateifunktionen***

- *Neu*: Erstellt einen neues Layout oder Arbeitspaket "Projekt"
- *Öffnen*: Öffnet vorhandene Projekt-Datei (*.lyt)
- *ASCII öffnen*: Öffnet ASCII Datei (*.txt) (**S**tructured **T**ext)
- *DLL öffnen*: Öffnet (importiert) eine DLL-funktion
- *Speichern*: Speichert aktuelles Projekt in *.lyt-Datei
- *ASCII speichern*: Speichert aktuelles Projekt als **S**tructured **T**ext (ST) in einer ASCII-Datei (*.txt)
- *Speichern unter*: Speichert aktuelles Projekt in *.lyt- und *.txt Datei unter neuem Namen
- *Schließen*: Schließt aktuelles Projekt.

❑ ***Passwort- und Druckfunktionen***

- *Passwort ändern*: Eingabe/Änderung des Online-Passworts.
  Nach Freischaltung des Passworts sind Änderungen, Speichern und Schließen des aktuellen Projektes nur nach korrekter Passwort-Eingabe möglich. Hot-Swap-Ebenen können so vor Umschaltung geschützt werden.
- *Drucken*: Öffnet Druckauswahlfenster zum Ausdruck von Teilen oder des gesamten Projektes.
- *Seite einrichten*: Seite einrichten für Druckerausgabe mit Angabe der Seitengröße, Ränder, usw.

❑ ***Einstellungen***

- *Programmeinstellungen*: Dialogaufruf für Programmeinstellungen,
  siehe Seite 2-24 ff.
- *Systemeinstellungen*: Dialogaufruf für Systemeinstellungen,
  siehe Seite 2-32 ff.
- *PCI-Konfiguration*: Dialogaufruf für PCI-Konfiguration,
  siehe Seite 2-41 ff.
- *ISA-Konfiguration*: Dialogaufruf für ISA-Konfiguration,
  (nicht aktiviert unter Windows XP)
- *Treiber neu starten*: Neustart der Kommunikationstreiber
- *Beenden*: Beenden von ibaLogic

2.3.2. "Bearbeiten"-Menü

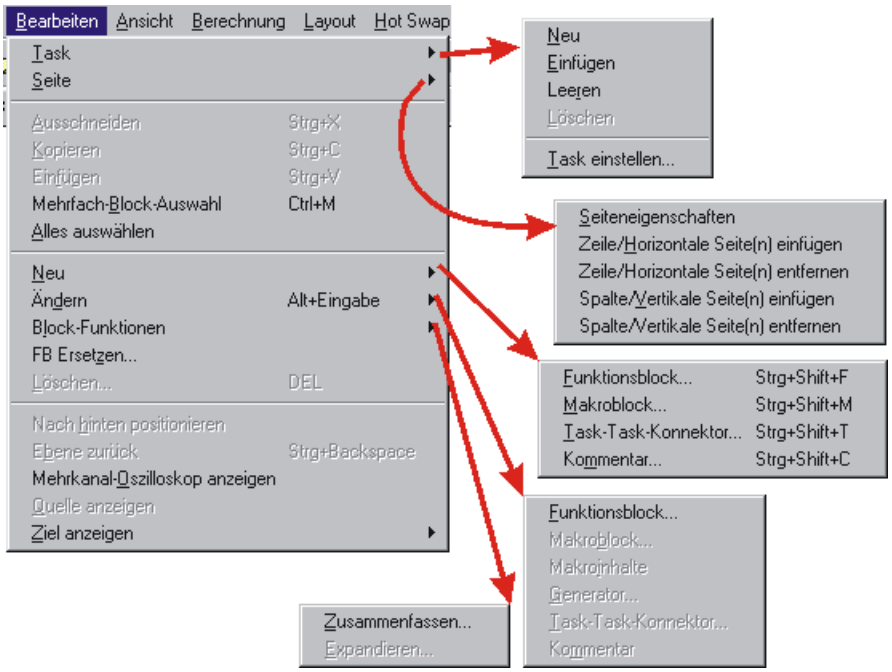


Bild 11 Menü Bearbeiten

- ***Task-Befehle:***
  - *Neu*: Erstellt eine neue Task
  - *Einfügen*: Fügt eine neue Task ein (vor die aktuell angewählte)
  - *Leeren*: Löscht Inhalt der Task
  - *Löschen*: Löscht die angewählte Task
  - *Task einstellen*...: Task-Konfiguration, mit Festlegung des Tasknamens, Zykluszeit, Größe des Task-Arbeitsbereichs und der Taskreihenfolge
- ***Seiten-Befehle:***
  - *Seiteneigenschaften*: Öffnet ein Fenster für die Einstellungen des Ausdrucks
  - *Zeile/Horizontale Seite(n) einfügen oder entfernen*: Fügt eine neue Seite bzw. Zeile oberhalb der aktuellen Seite ein oder entfernt diese.
  - *Spalte/Vertikale Seite(n) einfügen oder entfernen*: Fügt eine neue Seite bzw. Spalte links von der aktuellen Seite ein oder entfernt diese.
- ***Baustein-Kommandos 1:***
  - *Ausschneiden:* Funktionsbaustein oder Auswahl ausschneiden
  - *Kopieren:* Auswahl kopieren
  - *Einfügen:* Auswahl einfügen
  - *Mehrfach-Block-Auswahl:* Umschaltung der Cursorfunktion (Gummiband) zur Auswahl einer Gruppe von Bausteinen, Linien, Kommentaren
- ***Baustein-Kommandos 2:***
  - *Neu:* Öffnet ein Dialogfenster zur Erstellung eines neuen Funktionsbausteins, Makrobausteins, Off-Task-Konnektors oder Kommentars.
  - *Ändern:* Öffnet ein Dialogfenster zur Änderung der Eigenschaften eines Objektes, das vorhanden und markiert sein muss .
  - *Block-Funktionen*:
    Zusammenfassen: Fügt die ausgewählten Bausteine, Linien und Kommentare zu einem Makro zusammen
    Expandieren: Löst einen Makrobaustein in Einzelbausteine auf und fügt diese ein

- *FB Ersetzen*: Hiermit läßt sich ein FB durch einen anderen ersetzen, wahlweise in einer (markierten) Instanz oder in allen Instanzen
- *Löschen*: Löscht Auswahl

2

Das Menü "Bearbeiten" erhalten Sie auch, wenn Sie im Arbeitsbereich von ibaLogic auf die rechte Maustaste drücken (Kontextmenü).

❑ *Navigation*

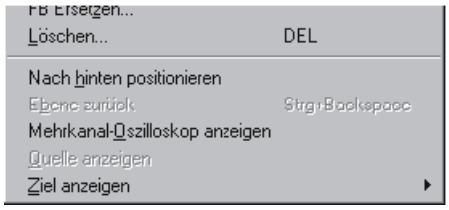


Bild 12 Menü Bearbeiten, Navigationskommandos

- *Nach hinten positionieren:* Setzt ein markiertes Objekt im Funktionsplan grafisch in den Hintergrund
- *Ebene zurück*
  Wechselt eine Hierarchie nach
  oben, d.h. verläßt den Makro-Level.
- *4Kanal-Oszilloskop anzeigen:* Zeigt das angewählte 4-Kanal-Oszilloskop in einem eigenen Fenster an.
- *Quelle anzeigen:* Zeigt die Verbindung zur Quelle (Task) bei Anwahl eines Task-Task-Konnektor-Eingangs.
- *Ziel anzeigen:* Zeigt die Verbindung zum auszuwählenden Ziel (Task(s)) bei Auswahl eines Task-Task-Konnektor-Ausgangs

2.3.3. "Ansicht"-Menü

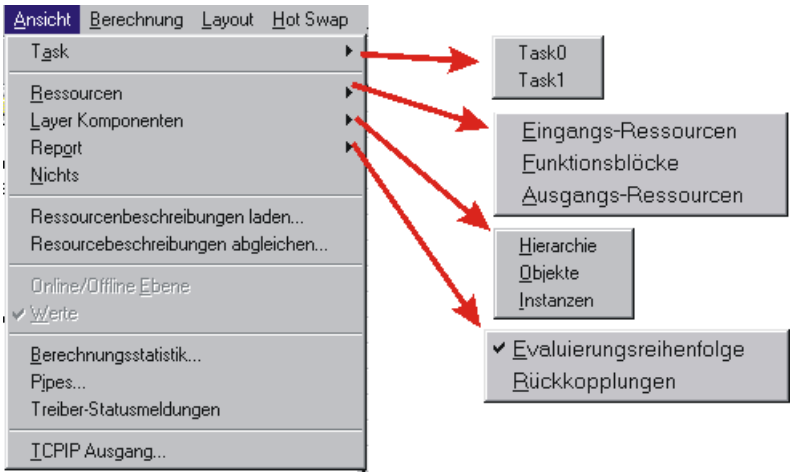


Bild 13 Menü Ansicht

❑ ***Task-Kommandos***

- Task: Wählt aus den bestehenden Tasks (im Beispiel 0 und 1) eine Task zur Ansicht aus.

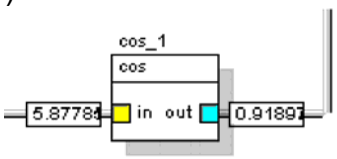
❑ ***Ressourcen-Auswahl***

- *Ressourcen*
  Eingangs-Ressourcen: Öffnet das Verzeichnis der Eingangsressourcen
  Funktionsblöcke: Öffnet den Katalog der Funktionen und Funktionsbausteine
  Ausgangs-Ressourcen: Öffnet das Verzeichnis der Ausgangsressourcen
- *Layer Komponenten*
  Hierarchie: Öffnet einen Strukturbaum welcher die verwendeten Objekte in ihrer Hierarchie darstellt, geordnet nach Tasks. Durch Mausklick auf ein Objekt kann zu diesem im Funktionsplan gesprungen werden.
  Objekte: Öffnet einen Strukturbaum, welcher alle Objekte und Instanzen, die in dem Layout verwendet wurden, nach *Objekttypen* geordnet anzeigt. Klickt man auf eine Instanz, wird angezeigt, wo diese Instanz verwendet wird (Task).
  Instanzen: Ansicht wie vorhergehende, aber nach *Instanznamen* sortiert.
- Report:
  Evaluierungsreihenfolge: Öffnet einen Strukturbaum der die Tasks und deren Objekte aufgelistet entsprechend ihrer Abarbeitungsreihenfolge von oben nach unten zeigt.
  Rückkopplungen: Öffnet einen Strukturbaum, der auf Endlosschleifen im Anwenderprogramm hinweist und die beteiligten Objekte aufführt.
- *Nichts*: Blendet den Ressourcenbereich komplett aus, d.h. der gesamte Bildschirm wird für den Arbeitsbereich genutzt.
- *Ressourcen-beschreibung laden*: Modifizierte Beschreibungen der I/O Ressourcen können geladen werden (z.B. I/O Ressourcen, die mit einem externen Editor umbenannt wurden und als CSV Dateien vorliegen. (Zur Erzeugung dieser CSV Datei einfach gewünschte Ressource mit rechter Maustaste anwählen und Export bestätigen.)
- *Ressourcen-beschreibung abgleichen*: Signalnamen vom Plan können als Ressourcenbezeichnung übernommen werden. Auch umgekehrt können Ressourcennamen in den Plan übernommen werden.

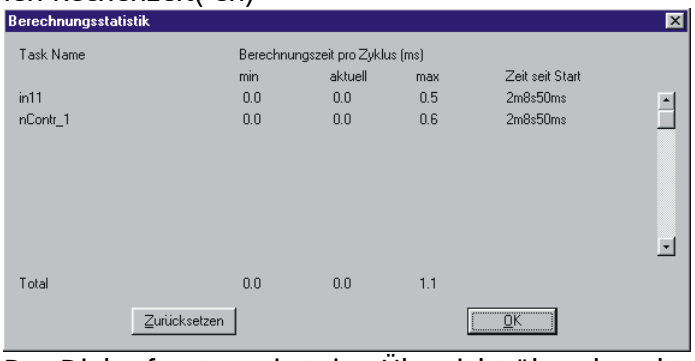
2

❑ ***Ebenen-Steuerung***

- *Online/Offline Ebene*: Umschaltung der Anzeige zwischen Online- und Offline Ebene im "Hot-Swap"-Modus.
- *Werte*: Anzeige der aktuellen Signalwerte von Funktionsbausteinen und Taskein- bzw. –ausgängen im Berechnungs- oder Online-Modus (siehe Beispiel "Werte ein")

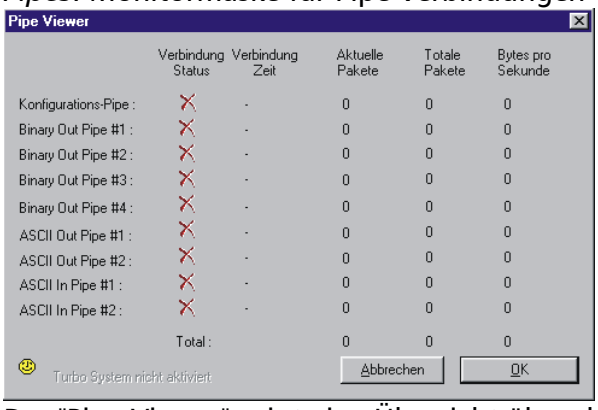


- *Berechnungsstatistik:* Abfrage und Anzeige der pro Task benötigten aktuellen Rechenzeit(-en)



Das Dialogfenster zeigt eine Übersicht über den aktuellen Task-Status sowie Informationen zu Task-Name, Rechenzeit (in ms) mit Min.-, Max- und Momentanwert, Laufzeit und Gesamtrechenzeit.

- *Pipes*: Monitormaske für Pipe-Verbindungen



Der "Pipe Viewer" zeigt eine Übersicht über den aktuellen Status der konfigurierten Pipes.

↗ *Siehe dazu auch 5.1.8*

- *Treiber Statusmeldungen*: Öffnet ein Fenster mit den Statusmeldungen von ibaLogic, wie z.B. Neustart der Treiber, Initialisierungen der Registry usw.
- TCPIP-Ausgang: Öffnet ein Anzeigefenster mit einer Übersicht über den aktuellen Status der konfigurierten TCP/IP-Verbindungen.

2.3.4. "Berechnung"-Menü

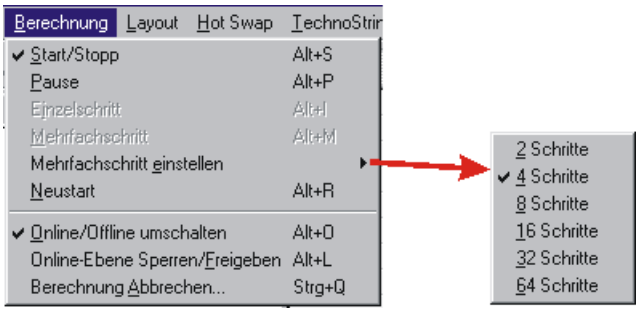


Bild 14 Menü Berechnung

- ***Steuerung des Berechnungs-Modus***
  - *Start/Stop*: Start/Stop der Offline-Berechnung aller Tasks (Berechnungs-Modus)
  - *Pause*: Unterbrechung bzw. Fortsetzung des Berechnungs-Modus
  - *Einzelschritt*: Berechnung eines Zyklus (aller Tasks)
  - *Mehrfachschritt*: Berechnung mehrerer Zyklen
  - *Mehrfachschritt einstellen*: Festlegung der Anzahl der Schritte (2..64) für "*Mehrfachschritt*"
  - *Neustart*: Rücksetzen aller Tasks
- ***Steuerung des Online / Offline-Modus***
  - *Online/Offline umschalten*: Umschaltung zwischen Online- und Offline-Modus. Im Online-Modus wird der Arbeitsbereich von ibaLogic rosa dargestellt.
  - *Online-Ebene Sperren / Freigeben*: Sperren der aktuellen Online-Ebene durch Eingabe des Passworts (falls Passwort vergeben); es sind kein Offline-Modus und keine Änderungen an dieser Ebene mehr möglich. Eine Online-Ebene muss gesperrt werden, ehe eine "Hot-Swap"-Ebene erzeugt werden kann.
  - *Berechnung abbrechen*: Unterbricht nach Bestätigung sofort den Online- bzw. Berechnungs-Modus.

2.3.5. "Layout"-Menü

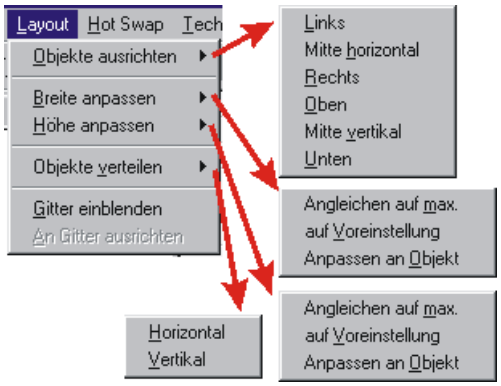


Bild 15 Menü Layout

❑ *Layout-Funktionen*

Die Layout-Funktionen beziehen sich auf die Darstellung der Objekte im Arbeitsbereich von ibaLogic, wie z.B. Funktionsblöcke, Off-Task-Konnektoren oder Kommentarfelder. Die betreffenden Objekte müssen zuvor markiert worden sein.

- *Objekte ausrichten*: Die markierten Objekte werden entsprechend dem Untermenü an einer gemeinsamen Linie ausgerichtet. Rechts, links, oben und unten beziehen sich jeweils auf die Außenkanten, Mitte horizontal/vertikal auf die Mittellinien der Objekte.
- *Breite anpassen, Höhe anpassen*: In den Untermenüs werden jeweils verschiedene Möglichkeiten der Anpassung geboten:
  Angleichen auf max.: Es müssen mehr als ein Objekt markiert sein. Die Funktion passt die Breite / Höhe aller markierten Objekte an das breiteste / höchste Objekt der Gruppe an.
  auf Voreinstellung: Es können ein einzelnes oder mehrere Objekte markiert sein. Die Funktion passt die Breite / Höhe der Objekte je nach Typ entsprechend der Einstellungen im Menü *↪Datei ↪Programmeinstellungen ↪Bearbeitung* an.
  Dabei kann die Anpassung im Sinne einer Verkleinerung nur soweit erfolgen, dass der Inhalt des Objektes, z.B. die Ein- und Ausgangskonnektoren eines Funktionsblocks, noch vollständig dargestellt werden können.
  Anpassen an Objekt: Die Funktion passt die Breite des markierten Objekts an den Inhalt an, so dass z.B. die Namen der Ein- und Ausgangskonnektoren vollständig zu lesen sind. Die Höhenanpassung berücksichtigt die Einstellung für den Abstand der Bausteinanschlüsse im Menü *↪Datei ↪Programmeinstellungen ↪Bearbeitung*.
- *Objekte verteilen*: Es müssen mindestens drei Objekte markiert sein. Entsprechend der Vorwahl im Menü *↪Datei ↪Programmeinstellungen ↪Bearbeitung*, werden die markierten Objekte in horizontaler bzw. vertikaler Richtung mit einem gleichen Abstand ihrer linken bzw. oberen Kante zueinander oder mit einem gleichen Zwischenraum verteilt.

2.3.6. "Hot Swap"-Menü

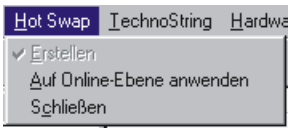


Bild 16 Menü Hot Swap

❑ ***Hot Swap-Steuerung***

- *Erstellen*: Eine "Hot Swap" - Ebene ermöglicht es, eine aktuell im Online-Modus laufende Task zu kopieren, zu ändern und anschließend wieder im laufenden Betrieb "heiß" umzuschalten
  Zur Erzeugung einer "Hot Swap" - Ebene müssen folgende Arbeitsschritte ausgeführt werden:
  1.) In Online Modus wechseln "*Online-Berechnung aktivieren*"
  2.) Sperren der aktuellen Online-Ebene "*Online-Ebene sperren*" (Schlüsselsymbol)
  3.) Hot Swap Ebene erzeugen durch ↪*Hot Swap* ↪*Erstellen*. Durch dieses Kommando wird der Inhalt der aktuellen Online-Ebene kopiert, ohne den Online-Modus zu verlassen. Die kopierte "Hot Swap" - Ebene kann nun modifiziert werden.
- *Auf Online-Ebene anwenden*: Die modifizierte Hot Swap Ebene wird mit diesem Befehl im laufenden Betrieb online geschaltet.
  Anmerkung: Es gilt hier Folgendes zu beachten. Wird die Ebene erzeugt, so übernimmt die "Hot Swap" - Ebene zunächst das Gedächtnis der Online-Ebene (Speicherelemente). Wird die "Hot Swap" - Ebene online geschaltet, so übernimmt sie erneut das Gedächtnis der Online-Ebene, sofern diese Elemente bereits vorhanden sind. Für neue Funktionsbausteine (mit Gedächtnis) werden die Werte aus der "Hot Swap" - Ebene übernommen. Damit ist sichergestellt, dass zwischenzeitlich durchgeführte Zustandsänderungen in der Online Ebene (z.B. durch OPC Steuerbefehle) nicht verloren gehen.
- *Schließen*: Schließen der Hot Swap-Ebene; Änderungen gehen wieder verloren, sofern sie nicht zuvor online geschaltet wurden oder gespeichert werden.

2.3.7. "TechnoString"-Menü

Bild 17 Menu TechnoString

2

❑ ***TechnoString***

- *TCP/IP*: Öffnet das unten dargestellte Dialogfenster "TCP/IP TechnoString" mit der Möglichkeit der Zuordnung von TechnoString-Inhalten zu Eingangs-Variablen.

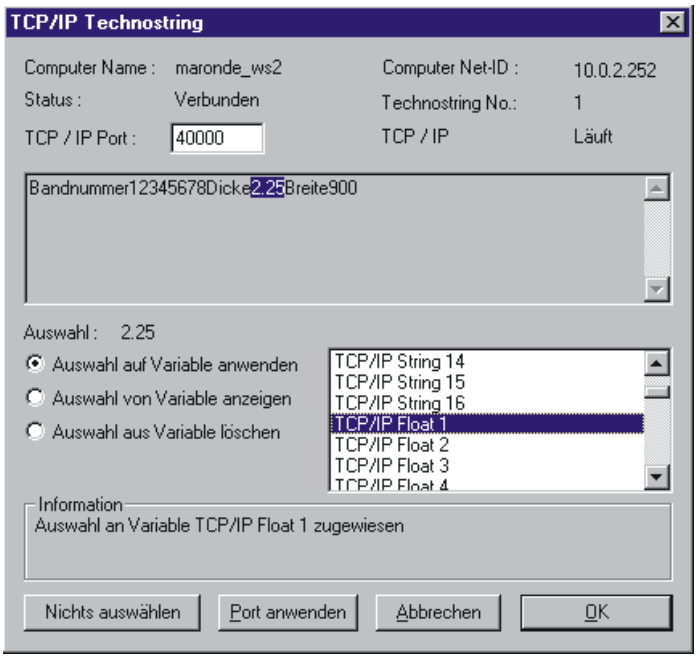


Bild 18 TCP/IP Technostring, Dialogfenster

Dieses Fenster dient zur Dekodierung (Zerlegung) eines ankommenden TCP/IP - Strings von beliebiger Struktur. Sein Inhalt kann indexorientiert sowohl Strings (TCP/IP String 1..16) als auch Floatvariablen (TCP/IP Float 1..96) zugeordnet werden. Dazu wie folgt verfahren:

1 Als Voraussetzung muss der TCP/IP-Betrieb für ibaLogic aktiviert sein. (Menü ↪Datei ↪Systemeinstellungen ↪Sonstige)

2 Im Feld *„TCP/IP Port"* ggf. die Portnummer korrigieren. Sie muss mit der Portnummer des sendenden Systems übereinstimmen. Siehe Kasten unten.

3 Von einem anderen System aus oder mit Hilfe des Testprogramms *TCPIP Test.exe* nun einen Technostring an den ibaLogic-Rechner schicken. Der Technostring erscheint in dem oben gezeigten Dialogfenster.

Es handelt sich um eine „hart" indizierte Dekodierung, d.h. der ankommende String muss immer gleiches Format haben (Gefahr bei Unterdrückung führender Nullen!)

---

4 Die Option *„Auswahl auf Variable anwenden"* anwählen.

5 Dann mit der Maus den gewünschten Bereich innerhalb des Stringfensters markieren, der einer Variable zugeordnet werden soll. Der markierte Bereich wird unter *„Auswahl:"* dargestellt (hier 2.25).

6 Nun innerhalb des Variablenfensters die gewünschte Variable anklicken (hier TCP/IP Float 1) und schon ist der markierte String der Variable zugeordnet.

7 Wenn weitere Teile des Technostrings weiteren Variablen zugeordnet werden sollen, die Schritte 4 und 5 entsprechend wiederholen.

8 Zur Kontrolle der Zuordnungen die Option „*Auswahl von Variable anzeigen*“ wählen und dann auf eine der verwendeten Variablen klicken. Der verknüpfte String wird angezeigt.

9 Sind alle Zuordnungen getroffen, dann auf OK klicken, um das Dialogfenster zu schließen.

Mit dem Schließen des Dialogfensters wird eine ASCII-Datei *iba_tcp.cfg* im Ordner *configuration* erzeugt, in der die Einstellungen gespeichert sind.

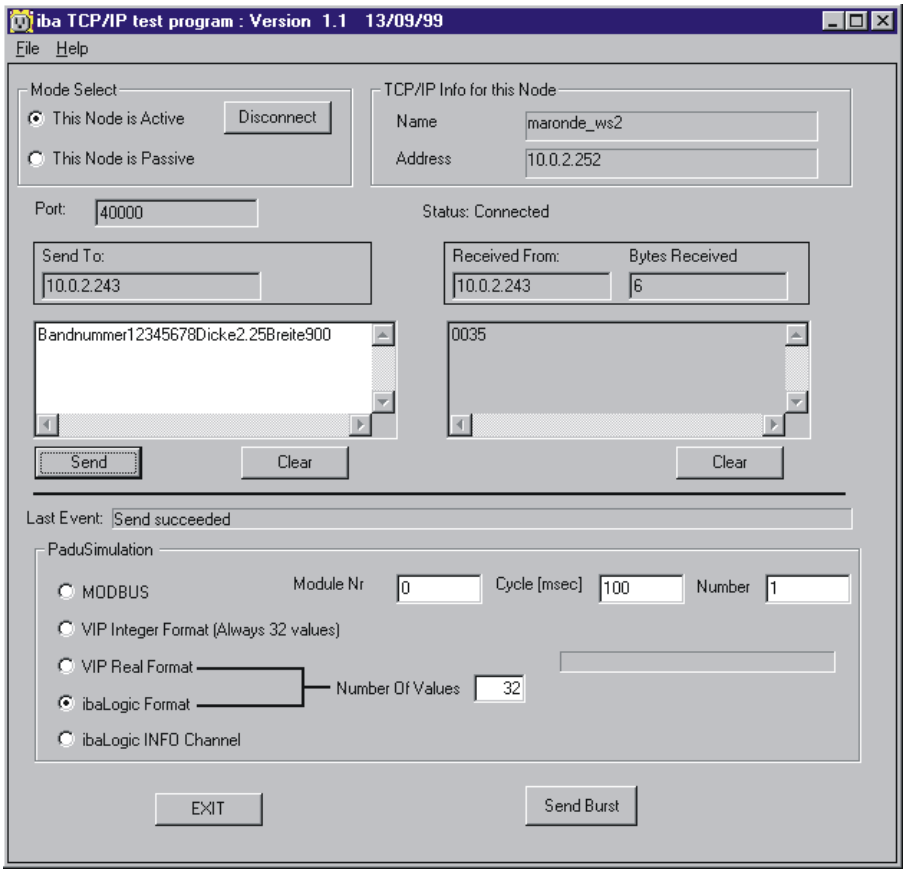
Für die Portnummer wird von ibaLogic stets der Defaultwert 1500 für den Technostring-Empfang eingetragen, sofern es keine Datei iba_tcp.cfg gibt, in der eine andere Portnummer abgespeichert ist.

Wenn die Portnummer für den Empfang anderer Daten als dem Technostring verwendet werden soll, dann muss in das oben gezeigte Dialogfenster eine andere Portnummer eingetragen und mit „OK“ die Datei iba_tcp.cfg erzeugt werden.

Wenn diese Datei beim Start von ibaLogic vorhanden ist, dann wird die darin gespeicherte Portnummer verwendet.

2

Anmerkung: Zum Test der TCP/IP-Funktion über das Netzwerk kann das von iba mitgelieferte Testprogramm Tcpip.exe genutzt werden. Hier einfach Netzwerkadresse und Portnummer des Zielrechners eintragen, den gewünschten Text in das Sendefenster eintragen, „Connect" und dann „Send" drücken. Der String erscheint im Empfangsfenster (siehe oben)



2.3.8. "Hardware"-Menü

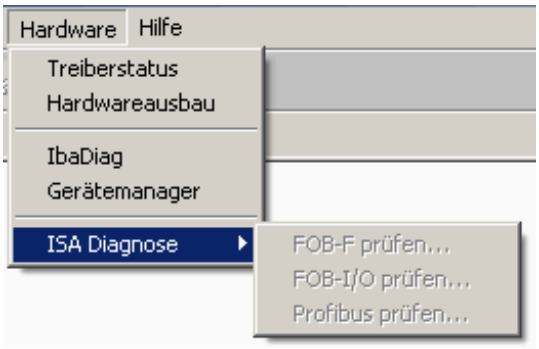
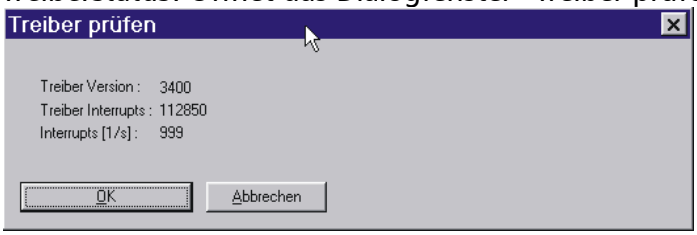


Bild 19 Menü Hardware

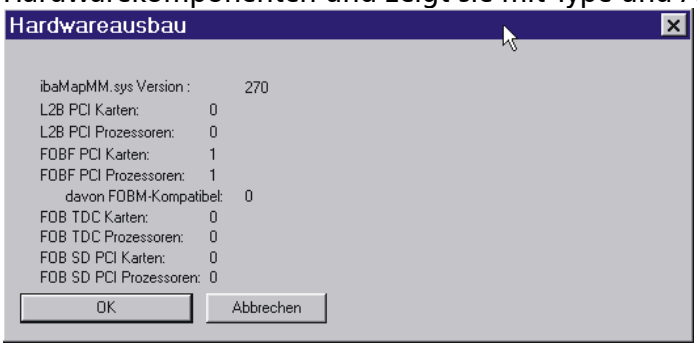
❑ ***Hardware***

- *Treiberstatus*: Öffnet das Dialogfenster "Treiber prüfen" (siehe unten).



Wenn die Installation ordnungsgemäß erfolgt ist, dann sollte der Wert für Interrupts [1/s] bei ca. 1000 liegen. (kann schwanken)
(Ausnahme: FOB 4i-PCI im Asynchron-Modus)

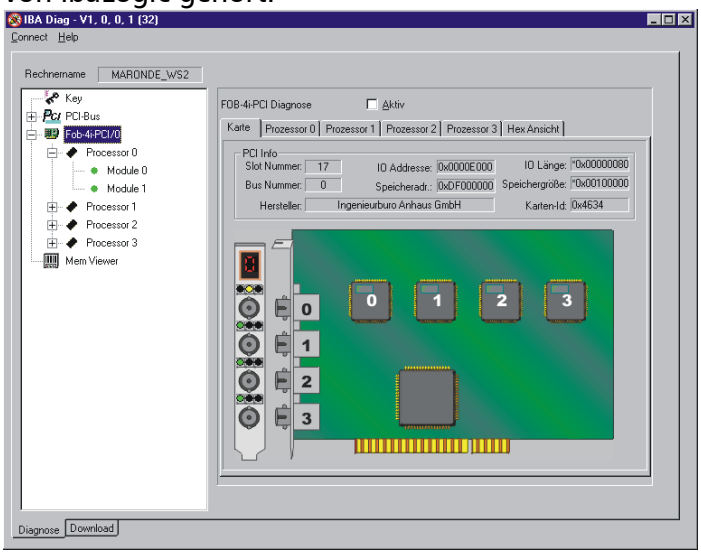
- *Hardwareausbau*: Das System erkennt automatisch die installierten (iba-) Hardwarekomponenten und zeigt sie mit Type und Anzahl an.



In diesem Beispiel ist eine FOB I/O-Karte installiert.
Bei Installation einer FOB 4i PCI-Karte würden eine FOBF PCI-Karte und vier FOBF PCI Prozessoren angezeigt werden.

2

- *IbaDiag*: Startet das Diagnoseprogramm ibadiag.exe, das zum Lieferumfang von ibaLogic gehört.

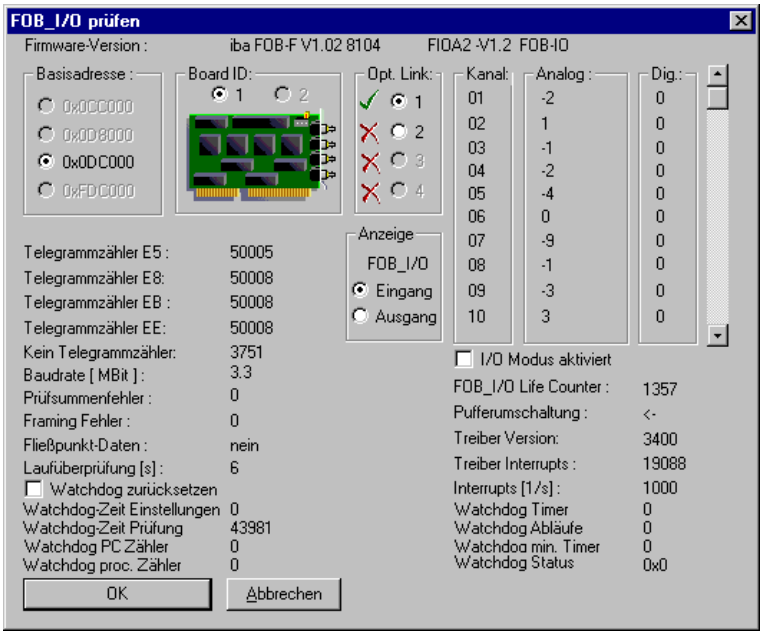


Beispiel für die Anzeige des ibaDiag-Programms.
ibaDiag kann auch unabhängig von ibaLogic auf einem PC gestartet werden.
Neben der detaillierten Karten-Diagnose liefert das Programm auch viele Informationen über den PCI-Bus und die angeschlossenen Komponenten.

Ausführliche Informationen zum Programm ibaDiag finden Sie im zugehörigen Handbuch ***sw_man_ibaDiag_de_a4.pdf****.*

- *Gerätemanager*: Dieser Menüpunkt funktioniert nur in Zusammenhang mit Windows XP. Es wird damit der Windows-Gerätemanager geöffnet, wo die installierten Treiber und Hardwarekomponenten angezeigt werden. Wenn in dem betreffenden PC I/O-Karten von iba installiert sind, so findet sich in der Baumstruktur im Dialogfenster des Gerätemanagers ein Zweig *iba Devices*. Öffnet man diesen Zweig, so findet man die Liste der installierten iba-Karten. Mit einem Doppelklick auf ein Kartensymbol erhält man weitere Informationen zu dieser Karte.

- *ISA-Diagnose mit Untermenü*: Öffnet das FOB-F, FOB-I/O (siehe Beispiel) oder Profibus-Diagnosefenster, sofern die entsprechende ISA-Hardware installiert ist.



Beispiel für eine ISA-Anzeige.
Neuere Systeme werden nur noch mit PCI-Karten ausgestattet, da die ISA-Technik bei den PCs nicht mehr weiter entwickelt wird.
Bei Verwendung von ISA-Karten verweisen wir an dieser Stelle auf die Version 2 der ibaLogic-Dokumentation.

2.3.9. "Hilfe"-Menü

Bild 20 Menü Hilfe

- *Inhalt*: Öffnet die Online-Hilfe-Funktion (in Vorbereitung)
- *Über...*: Anzeige der aktuellen ibaLogic Version

2.4 Programmeinstellungen

2.4.1. Menü ↪Datei ↪Programmeinstellungen ↪Allgemein

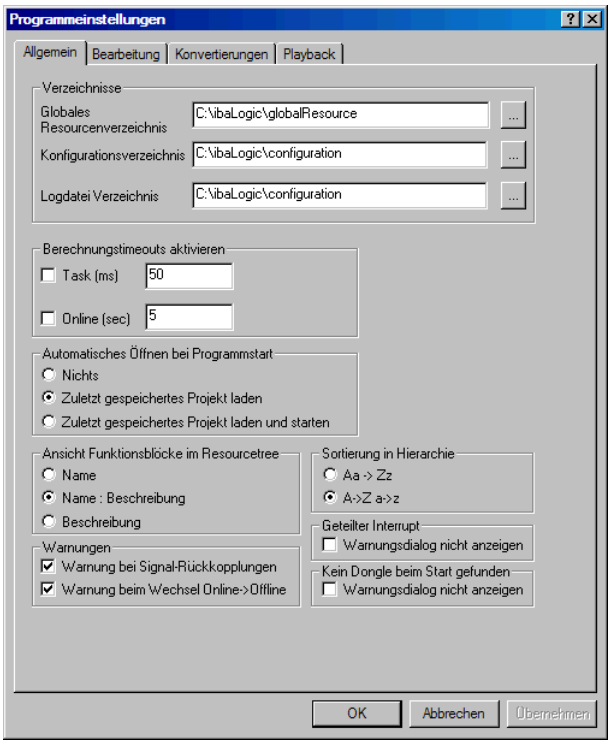


Bild 21 Allgemeine Programmeinstellungen

❑ ***Verzeichnisse***

- *Globales Ressourcenverzeichnis*: Pfadangabe für globale Ressourcen (vom Anwender erstellte DLLs, FBs, Makros und Libraries).
- *Konfigurationsverzeichnis*: Pfadangabe für Konfiguration mit DLLs, FBs, Makros, Libraries und Funktionsplänen (Projekte *.lyt / Structured Text *.txt).
- *Logdatei Verzeichnis:* Pfadangabe für die Protokolldatei von ibaLogic.

❑ ***Berechnungstimeouts aktivieren:***

Der Berechnungs- bzw. Online-Modus wird unterbrochen, sobald der eingestellte Berechnungstimeout einer Task (z.B. 50 ms) oder von ibaLogic (z.B. 5 s) abgelaufen ist. Diese Funktion unterbricht Dauerschleifen, die durch den Anwender projektiert wurden, bzw. reaktiviert ibaLogic in einem eventuellen Fehlerfall.

Die Berechnung wird u. U. auch angehalten, wenn die Werte für Berechnungstimeouts zu niedrig eingestellt sind.

❑ ***Automatisches Öffnen bei Programmstart:***

Festlegung des Startverhaltens von ibaLogic; hier kann der automatische Wiederanlauf aktiviert oder das Fortsetzen der Bearbeitung abgekürzt werden.

❑ ***Ansicht Funktionsblöcke im Ressourcetree:***

Auswahl der Anzeigeart der Funktionen und FBs im Ressourcen-Baum: nur der Name, nur die Funktionsbeschreibung oder beides

❑ ***Sortierung in Hierarchie:***

Regel für die alphabetische Sortierung der Objekte in der Ansicht "Layer Komponenten" / "Hierarchie" (Ressourcenbereich); ohne oder mit Unterscheidung der Groß-und Kleinschreibung.

❑ ***Warnungen:***

- *Warnung bei Signal-Rückkopplungen:* bei Aktivierung dieser Option wird bereits während der Projektierung eine Meldung ausgegeben, wenn eine Rückkopplung projektiert wurde.
- *Warnung bei Wechsel Online -> Offline:* dient zur Vermeidung einer versehentlichen Abschaltung des Online-Modus bei laufendem Prozess.

❑ ***Geteilter Interrupt***

Wenn ein Interrupt auf PCI-Ebene von zwei Karten geteilt wird, z.B. weil bei der Einrichtung kein exklusiver Interrupt für die PCI-Karten fest gelegt wurde, dann wird beim Start von ibaLogic (bzw. der Treiber) eine Warnmeldung ausgegeben. Der Start von ibaLogic wird dann unterbrochen. Man kann den Start fortsetzen und mit ibaLogic trotzdem arbeiten, wenn diese Meldung manuell quittiert wird.

Wenn die Option "Warnungsdialog nicht anzeigen" ein Häkchen bekommt, dann wird der Dialog nicht mehr angezeigt und ibaLogic startet durch.

Wenn ein ibaLogic-System für automatischen Neustart konfiguriert werden soll, damit es z.B.nach Spannungsausfall und –wiederkehr wieder selbständig startet, dann muss diese Option ein Häkchen bekommen, um den Hochlauf nicht zu blockieren.

❑ ***Kein Dongle beim Start gefunden:***

Ist dieses Kästchen nicht abgehakt, dann erfolgt beim Start von ibaLogic eine Meldung, wenn kein Dongle gefunden wurde. In dem Meldungsdialog werden verschiedene Möglichkeiten für den Start von ibaLogic ohne Dongle angeboten (Demo-Modus oder eCon-Modus).

2.4.2. Menü ↪Datei ↪Programmeinstellungen ↪Bearbeitung

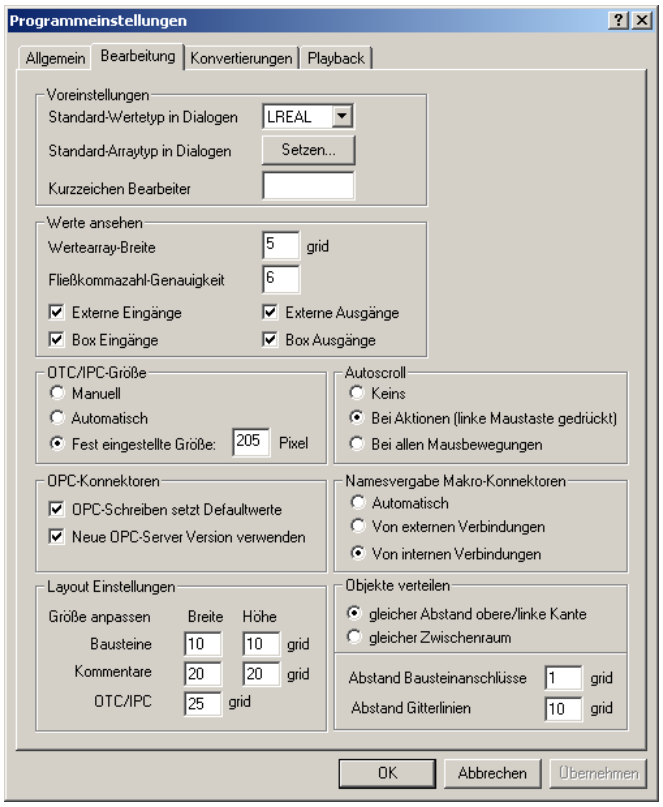
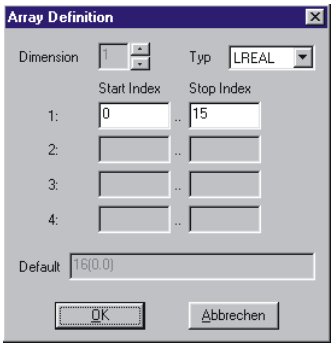


Bild 22 Programmeinstellungen, Bearbeitung

❑ ***Voreinstellungen***

- *Standard-Wertetyp in Dialogen*: Vorbesetzter Datentyp (z.B. LREAL)
- *Standard-Arraytyp in Dialogen*: Vorbesetzter Arraytyp (z.B. eindimensional LREAL)
  Schaltfläche "*Setzen*": Setup-Dialog für Standard-Arraytyp



  Mit Veränderung der Dimension (Pfeil-Buttons), ein- bis vierdimensional, werden die entsprechenden Indexfelder aktiviert.
  Start- und Stop-Index beschreiben die Anzahl der Arrayzellen je Dimension. In der Zeile Default wird die Anzahl der Array-Zellen und deren Defaultwerte angezeigt. Im Beispiel oben: eindimensionales Array mit 16 Zellen in denen jeweils der Wert 0.0 steht.
- *Kurzzeichen Bearbeiter*: Der Anwender kann hier sein Kürzel eingeben, das dann z.B. im Ausdruck erscheint.

❑ ***Werte ansehen:***

Mit Anwahl der Menü-Funktion ↪*Ansicht* ↪*Werte* kann man im Berechnungs- bzw. Online-Modus die aktuellen Signalwerte von Funktionsbausteinen und Taskein- bzw. –ausgängen beobachten.

- *Wertearray-Breite*: Größe der Wertanzeigen an den FBs im Evaluierungsmodus in Anzahl Rasterpunkte; (einen Eindruck von der Größe eines Rasterpunktes erhält man, wenn die Gitteranzeige eingeschaltet ist).
- *Fließkommazahl-Genauigkeit*: Anzahl der Nachkommastellen bei Real und LReal-Werten;
- *Kästchen mit Art der Ein- und Ausgänge*: Bestimmen, welche Werte angezeigt werden sollen;

❑ ***OTC/IPC-Größe:***

Hier kann die Darstellungsgröße von Off-Task- oder Intra-Page- Konnektoren gewählt werden. In der Stellung „Automatisch" wird die Größe des Konnektors an seinen Namen angepasst.

❑ ***Autoscroll:***

- *Keins*: Autoscroll ist ausgeschaltet, d.h. die Navigation im Arbeitsbereich erfolgt durch Drücken der linken Maustaste und Bewegung des Cursors.
- *Bei Aktionen (linke Maustaste gedrückt):* Navigation entweder durch Drücken der linken Maustaste (s.o.) oder automatisch beim Verschieben von Bausteinen bzw. Ziehen von Verbindungslinien.
- *Bei allen Bewegungen*: Autoscroll ist eingeschaltet, d.h. die Navigation im Arbeitsbereich erfolgt automatisch, sobald der Cursor sich im Randbereich des Arbeitsfensters befindet.

❑ ***OPC-Konnektoren:***

- *OPC-Schreiben setzt Defaultwerte*: Ist diese Option gewählt, dann kann der Defaultwert eines OPC-Konnektors vom OPC-Client überschrieben werden. Mit dieser Einstellung wird jede Wertänderung, die z.B. über ein HMI vorgenommen wurde als neuer Defaultwert vom OPC-Konnektor übernommen. Damit steht dem OPC-Konnektor bei einem Programmneustart der letzte aktuelle Wert als Startwert zur Verfügung. Ist die Option nicht gewählt, dann bleiben die bei der Projektierung eingestellten Defaultwerte unverändert. Die Wahl dieser Option ist nur für OPC-Konnektoren mit der Kennung OPC→ibaLogic relevant.
- *Neue OPC-Server Version verwenden:* Die Verwendung des neuen OPC-Servers wird dringend empfohlen.

❑ ***Namensvergabe Makrokonnektoren:***

Jeder Anschluss eines Funktionsbausteins hat eine Bezeichnung. Beim Zusammenfassen mehrerer Bausteine zu einem Makroblock entstehen an den geschnittenen Verbindungslinien die neuen Ein- und Ausgänge des Makroblocks. Je nach Auswahl der entsprechenden Option werden die Bezeichnungen der Anschlüsse eines Makroblocks automatisch vergeben oder von denen der inneren Bausteine oder denen der äußeren Bausteine / Ressourcen abgeleitet.

❑ ***Layout Einstellungen***

Bei den Layout-Einstellungen handelt es sich um Voreinstellungen für die Verwendung der Funktionen "Breite anpassen" und "Höhe anpassen" im Menü ↪*Layout*. Die Wertangabe erfolgt in Rasterpunkten.

- Bausteine: Vorgaben für die Abmessungen von Funktionsbausteinen
- Kommentare: Vorgaben für die Abmessungen von Kommentarfeldern
- OTC/IPC: Vorgabe für die Breite von Off-Task- oder Intra-Page-Konnektoren

↗ *Siehe dazu auch Kapitel 2.3.5*

2

❑ ***Objekte verteilen:***

- *gleicher Abstand obere/linke Kante*: Die markierten Objekte (mind. drei) werden bei Verwendung der Funktion "Objekte verteilen" im Menü ↪*Layout* an ihrer oberen/linken Kante neu ausgerichtet. Überschneidungen sind möglich.
- *gleicher Zwischenraum*: Die Markierten Objekte (mind. drei) werden bei Verwendung der Funktion "Objekte verteilen" zueinander so ausgerichtet, dass zwischen Ihnen stets der gleiche Zwischenraum entsteht.

Sonstige Einstellungen:

- *Abstand Bausteinanschlüsse*: Hier kann ein Mindestabstand für die Anschlüsse bei Funktionsbausteinen festgelegt werden. Die Einstellung kommt an einem markierten Baustein erst mit Verwendung der Funktion "Anpassen an Objekt" im Menü ↪Layout ↪Höhe anpassen zum Tragen.
- *Abstand Gitterlinien*: Angabe des Gitterlinienabstands in Rasterpunkten. Um das Gitternetz zu sehen, einfach Menü ↪*Layout* ↪*Gitter einblenden* anwählen.

↗ *Siehe dazu auch Kapitel 2.3.5*

2.4.3. Menü ↪Datei ↪Programmeinstellungen ↪Konvertierungen

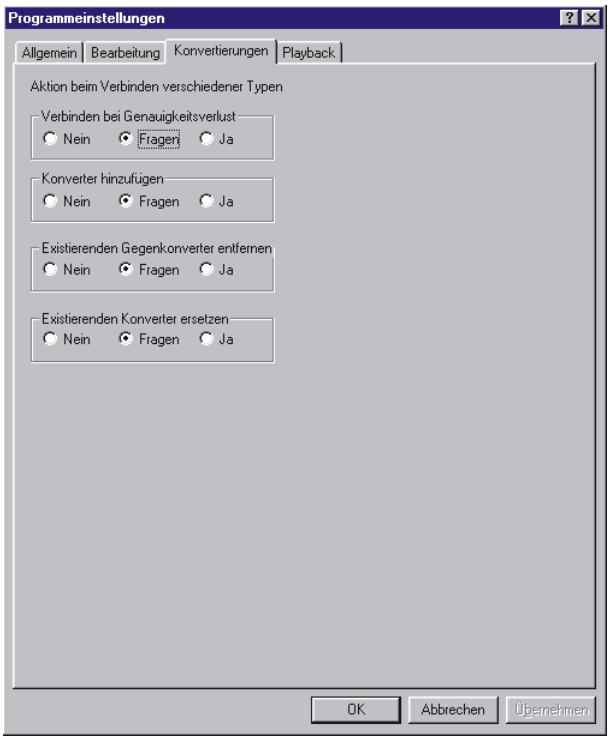


Bild 23 Programmeinstellungen, Konvertierungen

Über diese Schalter wird abgefragt, welche Aktion ibaLogic automatisch ausführen soll, sobald der Anwender versucht, Variablen unterschiedlichen Datenformats miteinander zu verbinden.

❑ ***Auswahl:***

- Verbinden, wenn möglich, aber Verlust von Genauigkeit
- Konvertierungsbaustein einfügen
- Vorhandenen Gegenkonvertierungsbaustein löschen
- Vorhandenen Konvertierungsbaustein ersetzen

Defaulteinstellung: "Fragen", d.h., im Bedarfsfall erscheint beim Projektieren ein Dialog, der vom Anwender bestätigt oder verneint werden kann.

2.4.4. Menü ↪Datei ↪Programmeinstellungen ↪Playback

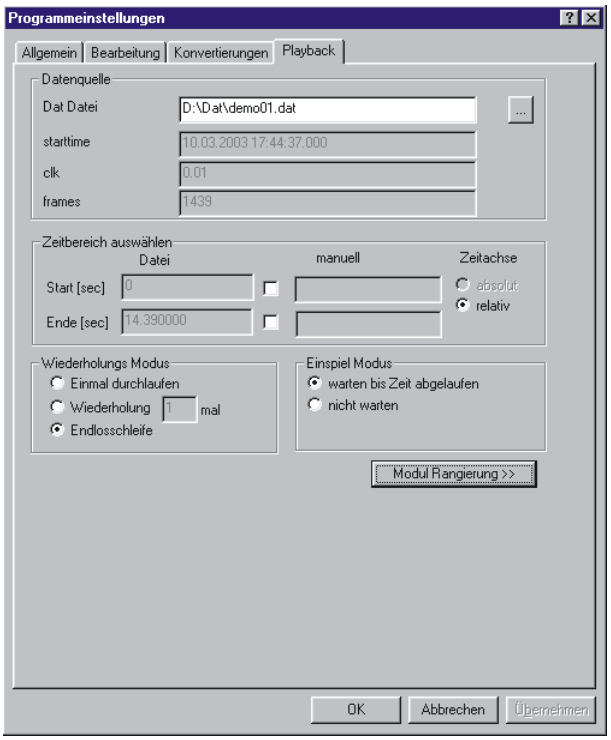


Bild 24 Programmeinstellungen, Playback

❑ ***Datenquelle:***

- *Dat Datei*: Pfad und Dateiname der Messdatei (*.dat), die als Signalquelle dienen soll. Ggf. Browser-Button rechts daneben benutzen. Ist eine gültige Datei gefunden worden, werden die wichtigsten Infos aus der Messdatei angezeigt (Startzeit, Sample time und Anzahl Messwerte).

❑ ***Zeitbereich:***

Hier kann der Zeitbereich aus der Messdatei, der mit dem Playback betrachtet werden soll, weiter eingeschränkt werden. Für die manuelle Eingabe von Anfangs- bzw. Endzeitpunkt zuvor in dem entsprechenden Kästchen ein Häkchen machen.

❑ ***Wiederholungs Modus:***

Auswahl, wie oft die Wiedergabe erfolgen soll.

❑ ***Einspiel Modus:***

Um eine natürliche Wiedergabe zu erhalten, muss die Task-Laufzeit von ibaLogic gleichgroß oder kleiner als die Erfassungszeit der Daten in der Messdatei sein. Im Playback-Modus liest ibaLogic mit jedem Taskzyklus ein neues Sample aus der Massdatei ein, wenn die Zykluszeit der ibaLogic-Task mit der Erfassungszeit in der Messdatei übereinstimmt. Wenn die Task-Laufzeit von ibaLogic kürzer als die Erfassungszeit der Daten ist, dann hat die Wahl des Einspielmodus folgende Wirkung:

- *warten bis Zeit abgelaufen*: ibaLogic wartet mit dem Einlesen eines neuen Samples solange, bis die Erfassungszeit abgelaufen ist. (Bsp.: ibaLogic Zykluszeit = 5 ms, Erfassungszeit lt. Messdatei = 20 ms → ibaLogic liest nur in jedem vierten Zyklus neue Daten ein. Drei Zyklen lang wird jeweils derselbe Wert verwendet.)

- *nicht warten*: ibaLogic liest in jedem Zyklus ein neues Sample ein. Damit läßt sich ein Zeitraffereffekt erzeugen.

Wenn die Zykluszeit der ibaLogic-Task länger ist als die Erfassungszeit, dann können Signale „verloren gehen“, da ibaLogic mit jedem Zyklus zeitrichtig die Daten aus der Messdatei ausliest. Die Wahl „warten..“ oder „nicht warten...“ ist in diesem Fall irrelevant.

❑ ***Schaltfläche*** **"Modul Rangierung"**

Öffnet das Dialogfenster für die Zuordnung der Eingangssignale.

↗ *Siehe dazu auch Abschnitt* 3.6.4 für weitere Informationen

2.5 Systemeinstellungen

2.5.1. Menü ↪Datei ↪Systemeinstellungen ↪Allgemein

Bild 25 Allgemeine Systemeinstellungen

❑ ***Interrupt-Einstellungen:***

Auswahl des Interrupts für ISA-Boards oder des PCI-Boards, das den Interrupt erzeugen soll.

❑ ***Betriebsart:***

Auswahl der Betriebsart von ibaLogic:

- *Signal Manager*: Im Signalmanager-Modus liegt die Priorität der Verarbeitung auf der Erfassung und Berechnung jedes Signalwechsels; siehe auch Kapitel 3.6.1
- *Soft PLC*: Im Soft-PLC-Modus liegt die Priorität auf der Verarbeitung der aktuellsten Signalzustände; siehe auch Kapitel 3.6.2
- *Turbo Modus*: Diese Betriebsart kann nur bei PCs mit Doppelprozessor genutzt werden. Einer der Prozessoren wird dann ausschließlich für die ibaLogic-Verarbeitung benutzt; siehe auch Kapitel 3.6.3.
- *Playback*: Als Signaleingangsquelle dient eine Messdatei vom Typ *.dat, die zuvor mit ibaPDA, ibaScope oder auch ibaLogic erzeugt wurde; siehe auch Kapitel 3.6.4.

❑ ***Allgemeine Einstellungen***

- *Samplingtime*: Einstellung der Basis-Zykluszeit für ibaLogic-Programme. Sie darf nicht größer sein als die kürzeste Task-Zykluszeit.
- *Watchdog Time*: Einstellung der Überwachungszeit der Watchdog-Funktion. Wenn "Watchdog enable" angewählt ist, dann sendet ibaLogic an die ihm zugeordneten Steckkarten zyklisch Watchdog-Telegramme. Das Senden (Triggern) dieser Telegramme muss zyklisch innerhalb der hier eingestellten Zeit erfolgen. Überwacht wird dieser Vorgang von der Firmware der Karten, die diese Zeit kennen. Wenn das Triggern des Watchdogs, nicht in der eingestellten Zeit erfolgt, dann werden die Ausgaben am Lichtleiterausgang der Karten auf „0“ (Null) gesetzt. (Funktion wird nur von FOB IO, FOB 4i/4o [FOB-F] unterstützt.)
- *Pipe Untersetzung*: In das Eingabefeld kann ein Faktor (ganze Zahl) eingegeben werden. Er bezieht sich nur auf den Sendetakt von QDA-Pipes (siehe auch Kapitel 5.2.6). Der Untersetzungsfaktor bestimmt, mit welcher Taktrate als ganzzahliges Vielfaches der ibaLogic-Samplingtime (steht darüber) die QDA-Pipes gesendet werden. Die Anwendung dieses Untersetzungsfaktors ist dann sinnvoll, wenn die QDA-Pipes nicht unbedingt in der Samplingtime bearbeitet werden müssen. Die Prozessorlast lässt sich damit verringern.

❑ ***Verhalten der Eingangssignale***

- *Nicht verfügbare Signale sind invalid*: Eingangsressourcen der iba-Baugruppen (FOB IO, FOB 4i, L2B x/8 usw.) werden im Layout als invalid, d.h. rot umrandet gekennzeichnet, wenn die entsprechende Karte nicht im PC eingebaut bzw. nicht verfügbar ist.
- *Nullen bei Verbindungsabbruch*: Im Falle einer Unterbrechung der (optischen) Verbindung zu den Eingabegeräten, wird ibaLogic bei Wahl dieser Option für die entsprechenden Eingangssignale den Wert „0“ (Null) verwenden und nicht den zuletzt gültigen Wert.
  ↗ Siehe dazu auch Kapitel 3.7

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

2.5.2. Menü ↪Datei ↪Systemeinstellungen ↪Sonstige

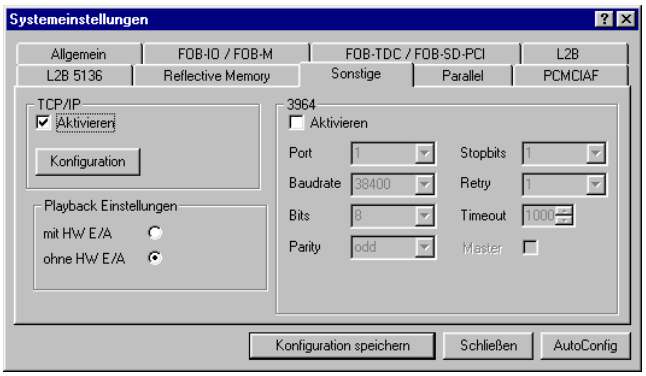


Bild 26 Systemeinstellungen, Sonstige

Dieser Dialog dient der Auswahl weiterer Verbindungen für die Ein- und Ausgabe von Signalen.

❑ ***TCP/IP:***

Aktivieren / deaktivieren der TCP/IP-Verbindung als Datenquelle. TCP/IP muss aktiviert werden, für Ein-/Ausgaben über ABB VIP, Modbus (TCP/IP), Verwendung von Technostrings sowie bei Verwendung von DLLs, die die TCP/IP-Kommunikation nutzen und beim Gebrauch des Funktionsblocks „TCPIP_SendRecv. Mit dem Button "*Konfiguration*" wird der Dialog "*TCPIP Einstellungen*" aufgerufen, mit dem die erforderlichen Einstellungen für die TCP/IP-Verbindung vorgenommen werden können; siehe auch Kapitel 2.6.6.

❑ ***3964***

Aktivieren / deaktivieren einer seriellen Verbindung, z.B. vom Typ 3964 R (DUST). Die Einstellung der Schnittstellenparameter erfolgt entsprechend dem Zielsystem. Für die Kommunikation über 3964 stehen in ibaLogic besondere Funktionsbausteine zur Verfügung.

❑ ***Playback Einstellungen***

Mit / ohne HW E/A, d.h. mit oder ohne Nutzung der Hardware- Ein-/Ausgaben im Playbackbetrieb. Damit lässt sich der Anwendungsbereich der Playback-Funktion noch mehr ausweiten. Bei Auswahl "mit" können Daten aus einer Messdatei dann in Kombination mit Hardwaresignalen verarbeitet werden. Damit sich die Playbacksignale und die Hardwaresignale nicht überlagern, stehen spezielle Playback-Eingaberessourcen zur Verfügung, siehe auch Kapitel 3.6.4.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

2.5.3. Menü ↪Datei ↪Systemeinstellungen ↪Parallel

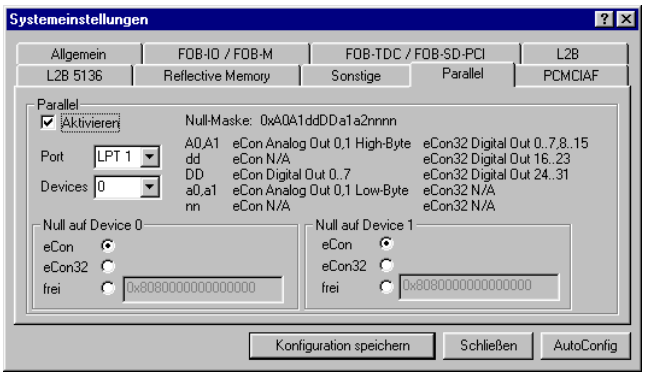


Bild 27 Systemeinstellungen, Parallel (Schnittstelle)

❑ ***Parallel***

- *Aktivieren:* Aktivieren / deaktivieren der Parallelschnittstelle des PCs (Druckerschnittstelle, LPT). Diese Schnittstelle ist dem Einlesen und Ausgeben von Signalen über die iba-Geräte **eCon** und **eCon32** vorbehalten. Diese Funktion steht auch ohne Dongle zur Verfügung.
- *Port:* Hier aus der Pick-Liste die gewünschte Schnittstelle auswählen, an der das eCon-Gerät angeschlossen ist. Im BIOS muss für diesen Port der bidirektionale Modus oder EPP-Modus eingestellt sein!
- *Devices:* Hier aus der Pick-Liste auswählen, ob ein eCon-Gerät oder zwei eCon-Geräte verwendet werden sollen.

0	...wenn nur ein **eCon** angeschlossen ist oder ...wenn zwei **eCons** angeschlossen sind, aber nur das erste genutzt wird.
1	...wenn zwei **eCons** angeschlossen sind, aber nur das zweite genutzt wird.
0&1	...wenn zwei **eCons** angeschlossen sind und beide Geräte genutzt werden.

❑ ***Null auf Device 0 / Device 1***

In diesen Auswahlfeldern sind die verwendeten Geräte zu markieren. Verbunden mit dieser Auswahl werden vordefinierte Nullmasken aktiviert. Die Nullmasken dienen dazu, alle Ausgänge der eCon-Geräte auf Null zusetzen, wenn das Layout offline geschaltet wurde. Die Maskierung der Ausgänge erfolgt mit Hilfe einer 16-stelligen Hex-Zahl. Abhängig vom Typ der eCon-Geräte unterscheiden sich die Nullmasken in der Interpretation der Bitbelegung. Mit einer dritten Option können auch individuelle Maskierungen eingestellt werden. Es lassen sich somit sogar gezielt Werte auf die Ausgänge legen, die ungleich Null sind. Dies ist aber in der Regel unüblich, denn wenn ein Layout offline geschaltet wird, dann erwartet man auch, dass die Ausgänge zurückgesetzt werden.

↗ *Siehe dazu auch Kapitel 5.2.9*

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

Eine detaillierte Beschreibung der Systemkonfiguration für den Betrieb mit eCon-Geräten finden Sie in der entsprechenden eCon-Dokumentation:

hw_man_eCon_de_A4.pdf

2.5.4. Menü ↪Datei ↪Systemeinstellungen ↪FOB-IO / FOB-M

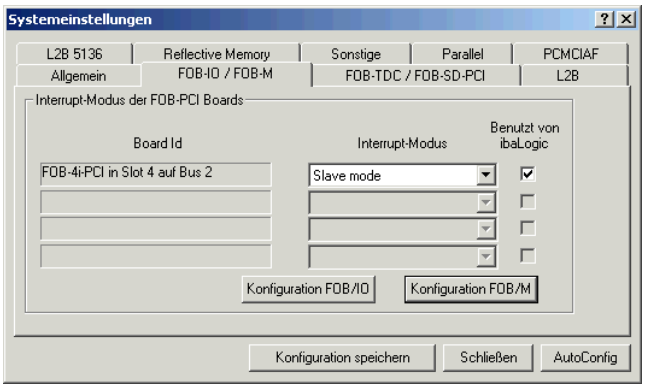


Bild 28 Systemeinstellungen, FOB-IO / FOB-M

❑ ***Interrupt-Modus der FOB-PCI Boards***

- *Board ID*: Anzeige der installierten iba PCI-Karten dieses Typs, automatische Erkennung
- *Interrupt-Modus:* Auswahl Master mode intern, extern oder Slave mode; nur eine iba PCI-Karte im PC darf im "Master mode" arbeiten!
- *Benutzt von ibaLogic:* ja / nein, Häkchen machen, wenn diese Karte ausschließlich von ibaLogic verwendet werden soll (und nicht von ibaPDA o.ä.)

Über die Buttons "*Konfiguration...*" werden die entsprechenden Einstellungsdialoge geöffnet, die auch über ↪*Datei* ↪*PCI-Konfiguration* ... aufgerufen werden können; siehe auch Kapitel 2.6.1 und 2.6.2.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

Anmerkung:

Die Auswahlkästchen "Verwendung als FOB-M" früherer Versionen (<3.88) sind entfallen. Für schnelle Messungen (bis 25 kHz Erfassungsrate) mit Padu8 M, Padu8 ICP oder Padu16 M, wenn die Karte also im FOB-M-Modus arbeiten muss, sind im Dialog Konfiguration FOB I/O die FOB M PCI-Link Einstellungen vorzunehmen. Die einzelnen Prozessoren der FOB 4i PCI-Karte können individuell auf FOB-M Modus umgeschaltet werden, so dass mit einer Karte gemischter Betrieb FOB-F / FOB-M möglich ist.

2.5.5. Menü ↪Datei ↪Systemeinstellungen ↪FOB-TDC / FOB-SD-PCI

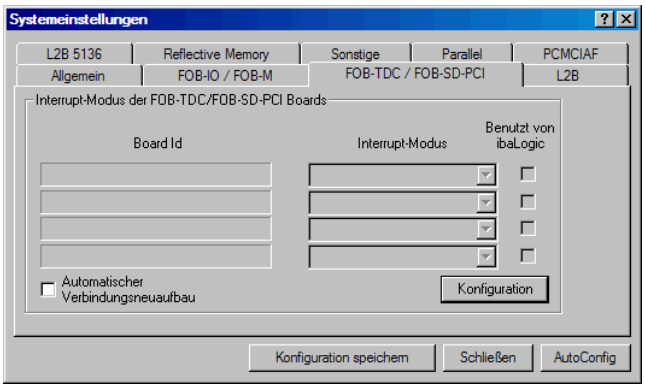


Bild 29 Systemeinstellungen, FOB-TDC / FOB-SD

❑ ***Interrupt-Modus der FOB TDC/FOB SD PCI Boards:***

- *Board ID:* installierte Karten dieses Typs, automatische Erkennung
- *Interrupt-Modus:* Auswahl Master mode intern, extern oder Slave mode; nur eine iba PCI-Karte im PC darf im "Master mode" arbeiten!
- *Benutzt von ibaLogic:* ja / nein, Häkchen machen, wenn diese Karte ausschließlich von ibaLogic verwendet werden soll (und nicht von ibaPDA o.ä.)

❑ ***Automatischer Verbindungsneuaufbau***

Wenn das Zielsystem (Simadyn D / Simatic TDC) im Betrieb abgeschaltet wurde oder aus anderen Gründen nicht verfügbar ist, dann werden die zugehörigen Ein-/Ausgaben in ibaLogic gesperrt da die entsprechenden Treiber nicht mehr arbeiten. Sofern unter *Systemeinstellungen ↪Allgemein*, *Verhalten der Eingangssignale,* die Option "*Nicht verfügbare Signale sind invalid*" (siehe 2.5.1) aktiviert wurde, werden die Ein-/Ausgaben als invalid gekennzeichnet. Andere Ein-/Ausgangsressourcen, die von dem Ausfall nicht betroffen sind, z.B. FOB IO-Karten, werden weiter berechnet.

Mit der Wahl dieser Option bestimmt der Anwender, dass sich ibaLogic nach einer Wiederkehr des Zielsystems (ibaLogic kann dies selbständig erkennen) erneut dort anmeldet und die Treiber neu startet. Dieser Vorgang dauert jedoch zwischen 5 und 20 Sek. In dieser Zeit wird die Layout-Berechnung von ibaLogic komplett angehalten, d.h. es stehen keine Ein-und Ausgaben zur Verfügung. Daher sollte die Option nur mit entsprechender Vorsicht verwendet werden.

Über den Button "*Konfiguration*..." wird der entsprechende Einstellungsdialog geöffnet, der auch über *↪Datei ↪PCI-Konfiguration* ... aufgerufen werden kann; siehe auch Kapitel 2.6.4.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

2.5.6. Menü ↪Datei ↪Systemeinstellungen ↪L2B

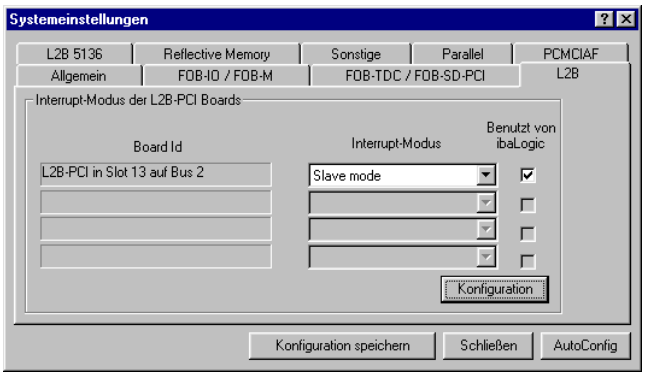


Bild 30 Systemeinstellungen, L2B

❑ ***Interrupt-Modus der L2B-PCI Boards:***

- *Board ID:* installierte Karten dieses Typs, automatische Erkennung
- *Interrupt-Modus:* Auswahl Master mode intern, extern oder Slave mode; nur eine iba PCI-Karte im PC darf im "Master mode" arbeiten!
- *Benutzt von ibaLogic:* ja / nein, Häkchen machen, wenn diese Karte ausschließlich von ibaLogic verwendet werden soll (und nicht von ibaPDA o.ä.)

Über den Button "*Konfiguration*..." wird der entsprechende Einstellungsdialog geöffnet, der auch über ↪*Datei* ↪*PCI-Konfiguration* ... aufgerufen werden kann; siehe auch Kapitel 2.6.3.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

2.5.7. Menü ↪Datei ↪Systemeinstellungen ↪L2B 5136

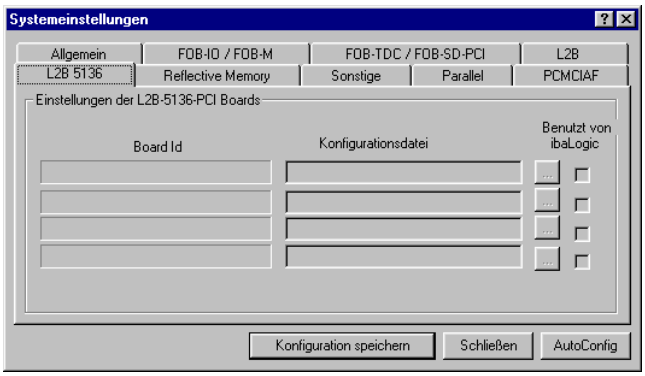


Bild 31 Systemeinstellungen, L2B 5136

❑ ***Einstellungen der L2B 5136 Boards:***

- *Board ID:* installierte Karten dieses Typs, automatische Erkennung
- *Konfigurationsdatei:* Pfad und Dateiname der Konfigurationsdatei eintragen oder über BrowserButton auswählen
- *Benutzt von ibaLogic:* ja / nein, Häkchen machen, wenn diese Karte ausschließlich von ibaLogic verwendet werden soll (und nicht von ibaPDA o.ä.)

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

2.5.8. Menü ↪Datei ↪Systemeinstellungen ↪Reflective Memory

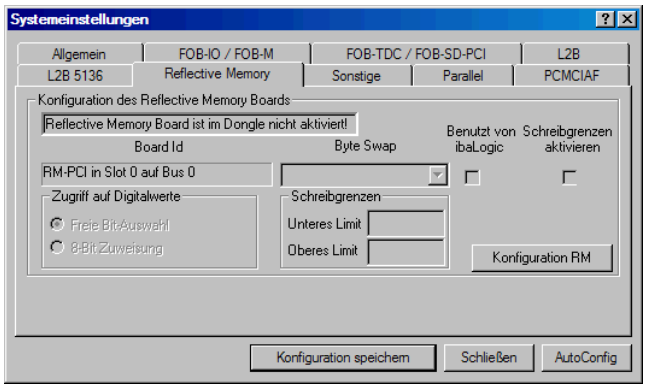


Bild 32 Systemeinstellungen, Reflective Memory

❑ ***Konfiguration des Reflective Memory Boards***

- *Board Id*: anzeige der installierten RM-Karte, automatische Erkennung
- *Byte Swap*: Aktivieren / deaktivieren eines Swap-Modus; hängt vom angeschlossenen System ab. Zu aktivieren, wenn das Server-System z.B. im Big Endian-Modus arbeitet. Zur Auswahl stehen *No Swap, Byte Swap, Word Swap, Byte and Word Swap* sowie *Swap on Size*.
  Hinweis: Die Swap-Funktionen werden nur noch von den RM-Karten VMI5576, -5579 und –5586 unterstützt. Die neuen Karten VMI5565 unterstützen den Swap-Modus nicht mehr.
- *Benutzt von ibaLogic:* ja / nein, Häkchen machen, wenn diese Karte ausschließlich von ibaLogic verwendet werden soll (und nicht von ibaPDA o.ä.)
- *Schreibgrenzen aktivieren*: Häkchen machen, wenn die Schreibgrenzen aktiviert werden sollen.
- *Zugriff auf Digitalwerte*: Hier auswählen, ob der Zugriff auf Digitalwerte bit- oder byteweise erfolgen soll.
- *Schreibgrenzen*: Vorgabe der unteren und oberen Schreibgrenze, nur möglich, wenn "*Schreibgrenzen aktivieren*" abgehakt ist.

Über den Button "*Konfiguration RM*" wird der entsprechende Einstellungsdialog geöffnet, der auch über ↪*Datei* ↪*PCI-Konfiguration* ... aufgerufen werden kann; siehe auch Kapitel 2.6.5.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

2.5.9. Menü ↪Datei ↪Systemeinstellungen ↪PCMCIAF

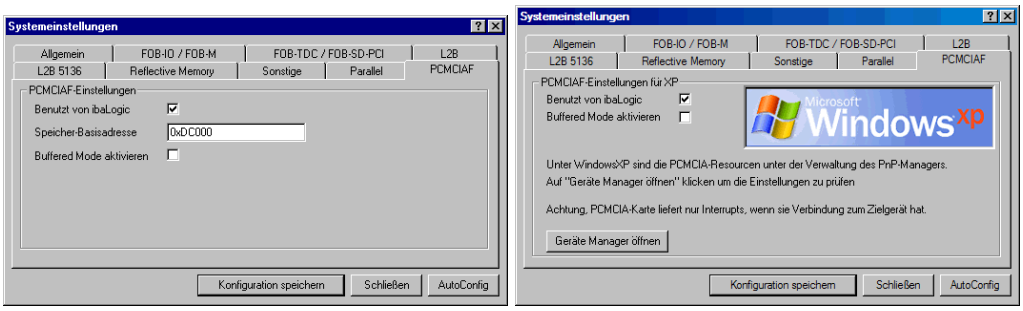


Bild 33 PCMCIAF-Unterstützung für Windows NT (li.) und XP (re.)

❑ *PCMCIAF-Einstellungen*

Mit Hilfe der PCMCIAF-Unterstützung kann ibaLogic auch auf einem Notebook mit Eingangssignalen versorgt werden. Es ist dazu die Karte PCMCIA-F von iba (Best.-nr. 1020) einzusetzen. Wenn die PCMCIA-F-Karte in ibaLogic verwendet werden soll, dann muss in dem entsprechenden Kästchen ein Häkchen gemacht werden. Die über die PCMCIA-F-Karte einlaufenden Daten werden dann auf die FOB-F-Eingangsressourcen gelegt, und zwar auf die ersten zwei Module bei den Analog- und Digitalwerten.

Die Speicher-Basisadresse ist automatisch voreingestellt. Ggf. muss sie im Rahmen der Karteninstallation angepasst werden.

Buffered Mode aktivieren: Mit Wahl dieser Option ist es möglich, ibaLogic auch auf einem Notebook-Computer im Buffered Mode zu betreiben. Es können dann die gepufferten Eingangsressourcen *FOB-F Buffered Mode* verwendet werden. (siehe 5.1.2)

Unter Windows XP ist die Kartenverwaltung etwas komfortabler als unter NT mit dem Gerätemanager realisiert.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" betätigt wurde.

2.6 PCI-Konfiguration

In dem Menü ↪*Datei* ↪*PCI-Konfiguration* gibt es die Möglichkeit, die Konfigurations-Dialoge für ausgewählte Karten zu öffnen, wie es auch in den entsprechenden Dialogen bei den Systemeinstellungen mit der Schaltfläche „Konfiguration“ möglich ist. (vergl. Abschnitte 2.5.2 bis 2.5.8)

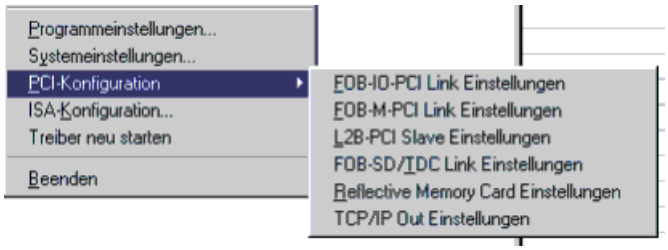


Bild 34 Menü PCI-Konfiguration

2.6.1. FOB-IO-PCI Link Einstellungen

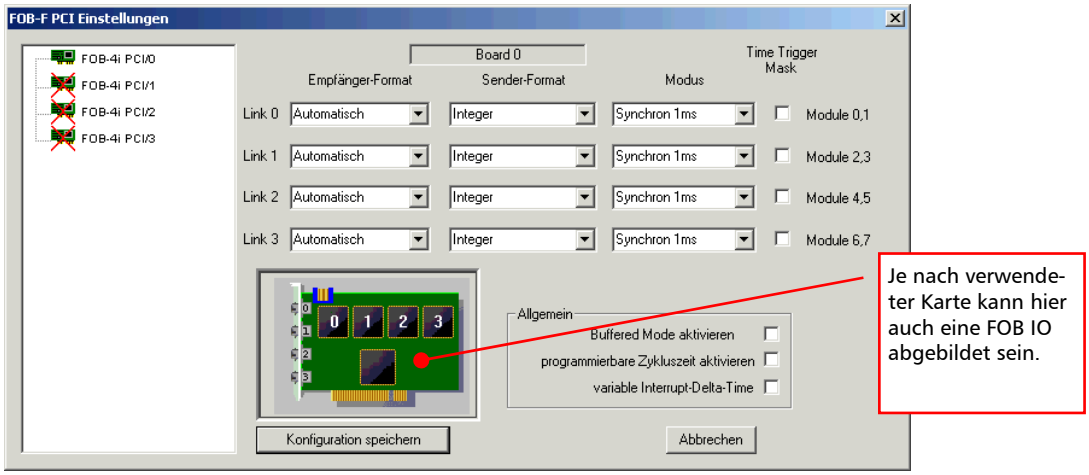


Bild 35 FOB-IO-PCI Link Einstellungen

In dieser Maske werden die Konfigurationen jedes Kanals (0...3) von bis zu vier FOB-F-Karten angezeigt (z.B. FOB IO oder FOB 4i PCI). Nach Anwahl einer Karte mittels Mausklick können deren Kanaleinstellungen verändert werden.

Die FOB-4i-X-Karten und die FOB-4o-X-Karten arbeiten nur im F-und M-Modus. Der X-Modus (32 Mbit Telegramm) kann unter ibaLogic-V3 nicht genutzt werden.

Die FOB-2i-X-Karten und die FOB-2io-X-Karten werden als FOB-4i-Karten angezeigt, es stehen aber nur zwei Links zur Verfügung.

❑ ***Empfänger-Format***

Datenformat für die Daten, die über den Lichtleiter einlaufen; empfohlene Einstellung: *Automatisch (default)*;

- *Integer*: für Daten von SM64, SM128V, ibaNet750 und Padus
- *Real*: für Daten von SM64, SM128V
- *S5 Real*: für Daten von SM64 im S5 Real-Format; die SM64-Karte muss auf den gleichen Modus eingestellt sein
- *FOB-M Mode*: Für schnelle Messungen (bis 25 kHz Erfassungsrate) mit Padu8 M, Padu8 ICP oder Padu16 M, wenn die FOB-Karte also im FOB-M-Modus arbeiten muss. Bei FOB-M Mode wird gleiches Empfänger- und Senderformat erzwungen.
  Da die einzelnen Prozessoren der FOB 4i PCI-Karte individuell auf FOB-M Modus umgeschaltet werden können, ist mit einer Karte gemischter Betrieb FOB-F / FOB-M möglich.

❑ ***Sender-Format***

wie oben, nur in Senderichtung

❑ ***Modus***

- *Synchron 1 ms*: Die Daten werden synchron zu den angeschlossenen Peripheriekomponenten mit der internen Basiserfassungszeit (1 ms) gelesen. Dies ist der übliche Modus für die Verwendung von FOB-F, FOB IO und FOB 4i PCI Eingangssignale.
- *Asynchron 1...10 ms*: Die Daten werden mit einer anderen Rate als der Basiserfassungszeit gelesen.
  ↗ *Siehe dazu auch* Besonderheiten des Asynchronmodus*, S. 2-44*

❑ ***Time Trigger Mask***

Freigabe für die Ausgabe programmierbarer Abtastraten an dem entsprechenden Lichtleiteranschluss. Als eine der Voraussetzungen für den Betrieb dieses Anschlusses im Asynchronmodus muss hier ein Häkchen gemacht werden.

❑ ***Allgemein***

- *Buffered Mode aktivieren*: Die eingelesenen Werte werden gepuffert und dem Layout als Array-Ressourcen (Puffertiefe max. 256) zur Verfügung gestellt. Diese Funktionalität ist optional und nur für die ersten 8 Fob-F Module implementiert. (Vergl. Kapitel 5.1.2 und 5.2.2)
- *programmierbare Zykluszeit aktivieren*: Aus dem Layout heraus können Abtastzeiten am Lichtleiteranschluss der Karte eingestellt werden.
- *variable Interrupt-Delta-Time*: Die Zeit zwischen zwei Interrupts kann variabel sein.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" bzw. "Werte übernehmen"+"Konfiguration speichern" betätigt wurden.

Besonderheiten des Asynchronmodus

Sinn und Zweck des Asynchronmodus ist es, die Abtastzeit der Padus an den zu beobachtenden Sachverhalt anzupassen (z.B. für eine FFT mit möglichst wenig Abtastpunkten).

Folgende Voraussetzungen sind zu erfüllen:

1. Der entsprechende Lichtleiteranschluss ist auf Asynchronmodus eingestellt.
2. Option Programmierbare Zykluszeit ist aktiviert.
3. Option Variable Interrupt-Delta Time ist eingeschaltet.
4. Die FOB-F-Karte, deren erster Anschluss im Asynchronmodus betrieben wird, generiert den Interrupt für ibaLogic.
5. Die FOB-F-Karte wird als Master/External betrieben.
6. Der Lichtleiter wird im Kreis verbunden.
7. ibaLogic läuft in der Betriebsart Signalmanager-Modus (↪Datei ↪System-einstellungen).

Sind diese Voraussetzungen erfüllt, dann gilt:

Die Einheit [ms] bei Samplingtime und Task-Berechnungsintervall wird ersetzt durch Interrupts (Anzahl). Das bedeutet, dass die Zeit zwischen zwei Interrupts variabel ist und somit auch die Zeit zwischen zwei Berechnungsintervallen.

Die in ibaLogic an die Task übergebene EvalDeltaTime wird unter diesen Voraussetzungen korrigiert und entspricht der echt abgelaufenen Zeit.

2.6.2. FOB-M-PCI Link Einstellungen

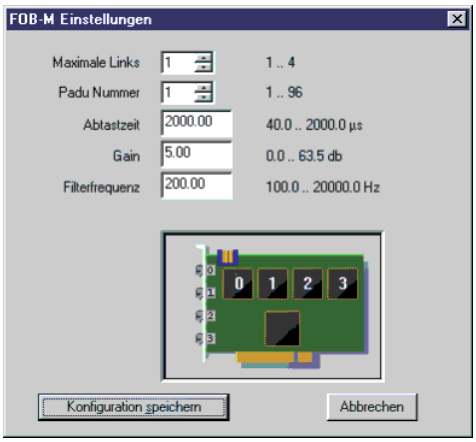


Bild 36 FOB-M-PCI Link Einstellungen

Mit diesem Dialog können die Defaultwerte für den Betrieb im FOB-M-Modus der Karte vorgegeben werden.

Diese Einstellungen gelten für alle Prozessoren (Links), die im FOB-M-Modus arbeiten.

Üblicherweise werden die Einstellungen jedoch im Layout individuell für jeden Prozessor angepasst. Die Layout festgelegten Einstellungen überschreiben dann diese Defaultwerte.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" bzw. "Werte übernehmen"+"Konfiguration speichern" betätigt wurden.

2

2.6.3. L2B-PCI Slave Einstellungen

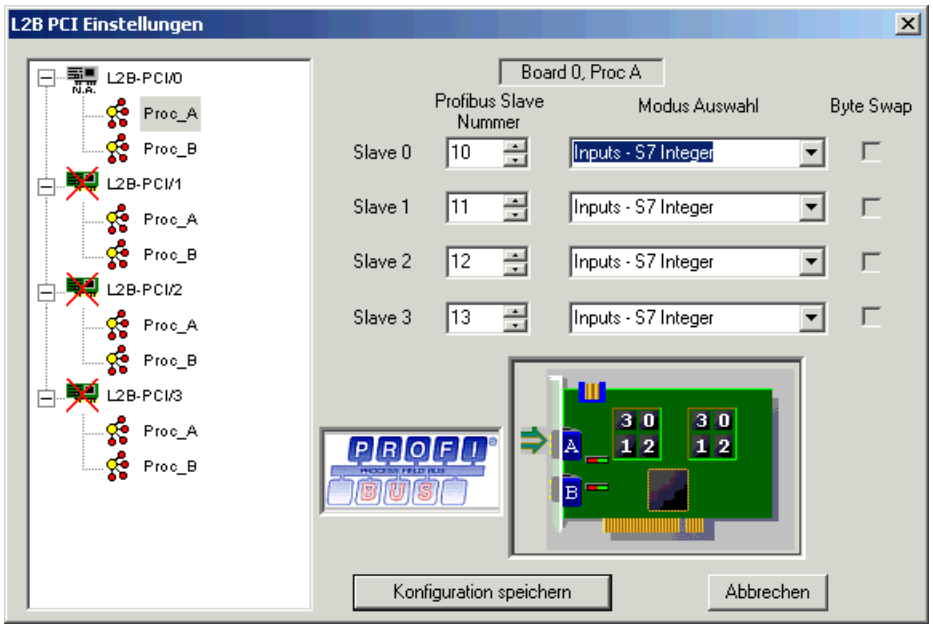


Bild 37 L2B-PCI Slave Einstellungen

In dieser Maske werden die Konfigurationen von bis zu vier L2B-PCI-Karten angezeigt. Nach Anwahl einer Karte links im Funktionsbaum mittels Mausklick können deren Einstellungen verändert werden.

Nicht für alle Firmwareversionen sind alle Modi verfügbar. Auch bei älteren iba-Logic-Versionen stehen nicht alle Funktionen zur Verfügung.

- ❑ Die Vorbelegung der Profibus Slave-Nummern ist gem. der DP-Programmierung anzupassen.
- ❑ Für jeden Slave kann der Datenverarbeitungsmodus, bzw. das Datenformat ausgewählt werden. Die Modi „Planheit..“ sind speziell für Daten aus dem Siemens Planheitsmesssystem vorgesehen (vergl. Kapitel 5.1.5).
- ❑ Ob für einen Slave die Option Byte Swap gewählt werden muss, hängt vom Zielsystem ab.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" bzw. "Werte übernehmen"+"Konfiguration speichern" betätigt wurden.

2.6.4. FOB-SD/TDC Link Einstellungen

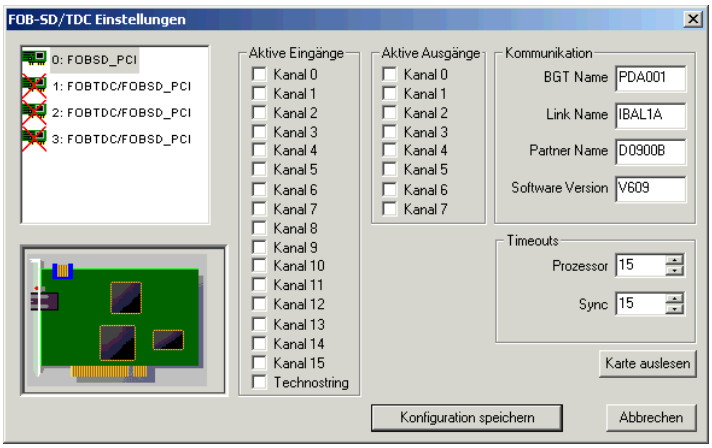


Bild 38 FOB-SD/TDC Link Einstellungen

In dieser Maske wird die Konfiguration einer FOB-SD/TDC-Karte angezeigt. Die Karteneinstellungen können verändert werden.

❑ ***Aktive Eingänge***

Hier können gezielt einer oder mehrere aus insgesamt 16 Kanälen selektiert werden (Häkchen für 0...15). Ein Kanal x entspricht in der Ressourcenleiste von ibaLogic einem FOB-SD/FOB-TDC - Simadyn Lite – Modul x (analog + digital, x = 0...15).

Bitte beachten, dass für jeden gewählten Eingangskanal ein Sendetelegramm MxPDADAT (x = 0...9, A...F) in Simadyn D, bzw. TDC vorzusehen ist.

❑ ***Aktive Ausgänge***

Hier können gezielt einer oder mehrere aus insgesamt acht Kanälen selektiert werden (Häkchen für 0...7). Ein Kanal x entspricht in der Ressourcenleiste von ibaLogic einem FOB-SD/FOB-TDC - Simadyn Lite – Modul x (analog + digital, x = 0...7).

Bitte beachten, dass für jeden gewählten Eingangskanal ein Empfangstelegramm PDAMxDAT (x = 0...7) in Simadyn D, bzw. TDC vorzusehen ist.

Da ibaLogic hier dem Prinzip der Kommunikation zwischen Simadyn D und ibaPDA folgt, finden Sie weitere Hinweise in den Projektierungsanleitungen zum PDA-System, ***sw_man_ibaPDA-SD_Project...*** *bzw.* ***sw_man_ibaPDA-TDC_Project...,*** *die auf unserer Website zum Download bereit stehen.*

❑ ***BGT Name***

Hier steht der eigene, lokale PC-Systemname. Er entspricht in SD oder TDC dem Namen des Baugruppenträgers. Die Standard Einstellung "PDA001" muss nicht geändert werden.

❑ ***Own Name***

Dieser Eintrag bezeichnet den eindeutigen Verbindungsnamen zwischen der FOB-SD/TDC-Karte und dem Koppelpartner (CS14 Baugruppe bzw. GDM). Die Standard Einstellung ist evtl. dann zu ändern, wenn dieser Partner eine Verbindung zu einem weiteren ibaLogic- oder ibaPDA-System hat. Ist der Name nicht eindeutig, dann kommt die Fehlermeldung 0x6AA0.

2

❑ ***Partner Name***

In diesem Feld steht der Name der Koppelpartnerbaugruppe (CS14 oder GDM-Prozessor). Dieser muss zwingend hier eingetragen werden. Man erfährt ihn aus der SD- bzw. TDC-Projektierung oder aus der BGT-Diagnose mit ibaDiag. Ist der Name nicht richtig, dann kommt die Fehlermeldung 0x6AA6.

*Weitere Informationen hierzu finden Sie auch in unserem Handbuch zum Programm ibaDiag **sw_man_ibaDiag_de_a4.pdf**, Kap.2.3.11.*

❑ ***Software Version***

Dies ist die Versionsbezeichnung der Simadyn D bzw. Simatic TDC Grundsoftware. Diese muss zwingend hier eingetragen werden. Man erfährt sie aus der SD- bzw. TDC-Projektierung oder aus der BGT-Diagnose mit ibaDiag. Ist die Version nicht richtig, dann kommt die Fehlermeldung 0x6AB3.

*Weitere Informationen hierzu finden Sie auch in unserem Handbuch zum Programm ibaDiag **sw_man_ibaDiag_de_a4.pdf**, Kap.2.3.11.*

❑ ***Timeouts***

Hier sind die Überwachungszeiten für die Rückmeldung von Kommandos an die FOB-SD/TDC-Karte eingetragen. Die Standard Einstellung "15" muss normalerweise nicht geändert werden

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" bzw. "Werte übernehmen"+"Konfiguration speichern" betätigt wurden.

2.6.5. Reflective Memory Card Einstellungen

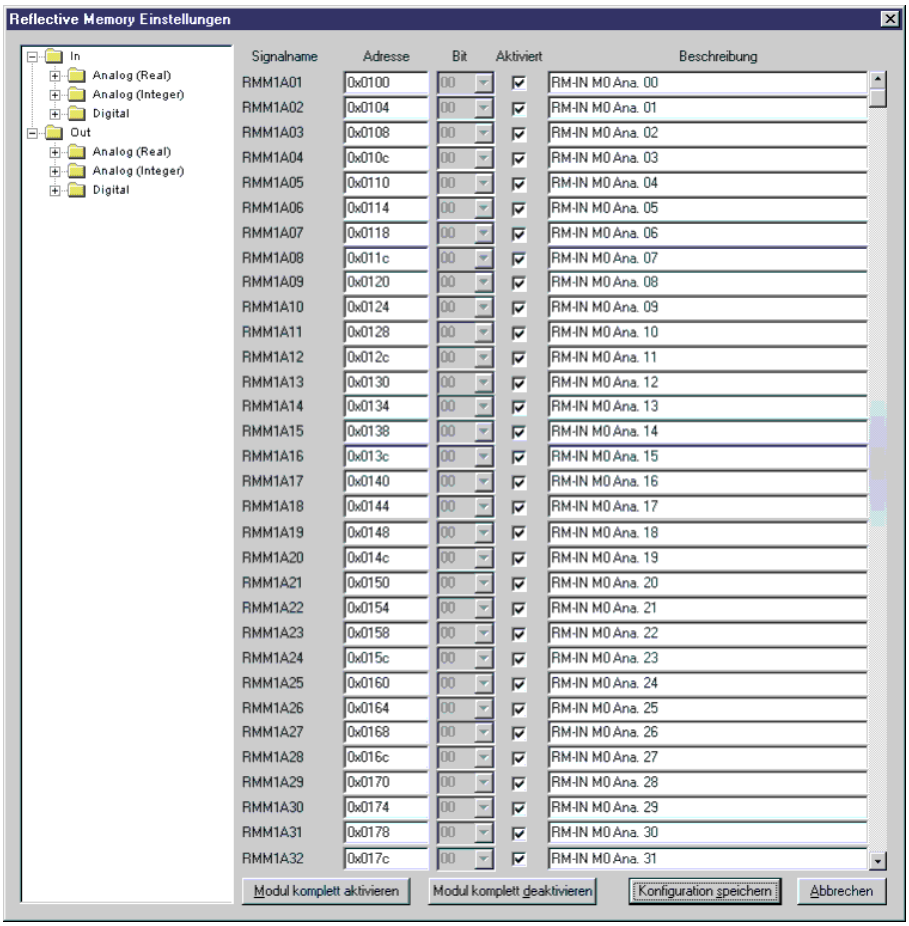


Bild 39 Reflective Memory Card Einstellungen

In dieser Maske können die Adressen und symbolischen Namen für Ein- und Ausgangsvariablen definiert werden, die über Reflective Memory mit anderen Systemen ausgetauscht werden. Dazu stehen entsprechenden Ein- und Ausgangsressourcen in ibaLogic zur Verfügung (je 32 Module zu je 32 Analogwerten, REAL oder Integer und 32 Module mit je 32 Digitalsignalen). Die entsprechenden Module sind in dem Signalbaum im linken Teil des Dialogfensters zu sehen.

Welche Einstellungen hier zu treffen sind, hängt wesentlich von dem angeschlossenen System ab. Die Adresswerte und Signalnamen, die in dem Dialogfenster angezeigt werden, sind Vorbelegungen, die nur beispielhaft zu verstehen sind und ggf. geändert werden müssen.

❑ ***Signalname***

In dieser Spalte werden die Namen der Signale angezeigt, die von ibaLogic intern verwendet werden und nicht verändert werden können. Diese Namen erscheinen im Tooltip, wenn der Mauszeiger auf dem Anschlusspunkt des Ein- oder Ausgangs im Funktionsplan platziert wird.

❑ ***Adresse***

Hier muss die Speicheradresse eingetragen werden, bei der das gewünschte Signal im Reflective Memory zu finden ist. Die Voreinstellung zeigt eine Adressierung, wie sie aussehen könnte:

Real-Signale ab 0x0100 in 4-Byte-Schritten mit Modulabstand 0x0100, Integer-Signale ab 0x0180 in 2-Byte-Schritten mit Modulabstand 0x0200 und Digitalsignale ab 0x0080 in einem Doppelwort (= 32 bit) mit Modulabstand 0x0002.

2

Wenn es sich um eine Punkt zu Punkt-Verbindung zwischen ibaLogic und einem angeschlossenen System handelt, bzw. die Daten so optimal strukturiert und zusammenhängend im Speicher vorliegen ist eine vergleichbare Adressierung denkbar. Reflective Memory (RM) erlaubt aber auch den Zusammenschluss von mehreren Systemen in einer Ringtopologie, wo mitunter auch Daten ausgetauscht werden, die nicht für ibaLogic bestimmt sind. In einem solchen Fall kann die Adressierung völlig frei erfolgen und an die RM-Projektierung angepasst werden.

Es besteht keine starre Verbindung zwischen RM-Adresse und ibaLogic-Variable. Die Modulstruktur muss hier nicht eingehalten werden!

❑ ***Bit***

Die Auswahlkästchen der Bitadressen werden erst bei Anwahl der Digitalsignale aktiviert. ibaLogic erwartet, dass die Digitalsignale je Modul in einem Doppelword (DWORD, 32 Bit) verpackt sind. Die Einzelsignale werden über ihre Bitnummer innerhalb des Doppelwortes adressiert. Auch die Bitadressierung kann hier den Erfordernissen angepasst werden.

❑ ***Aktiviert***

Mit diesem „Schalter“ können einzelne Signale deaktiviert werden, wenn sie für die Verarbeitung in ibaLogic nicht benötigt werden, bzw. von ibaLogic nicht beschrieben werden dürfen.

❑ ***Beschreibung***

Bei der Beschreibung handelt es sich um einen einfachen Klartext, der im Funktionsplan als Signalname verwendet wird. Die Beschreibung taucht als Signalname sowohl in der Randleiste des Funktionsplans als auch im Signalbaum der Ein-/Ausgangsressourcen auf.

❑ ***Schaltflächen „Modul komplett aktivieren / deaktivieren“***

Setzen die Aktivierungsbits aller Signale in dem angezeigten Modul, bzw. setzen sie zurück.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" bzw. "Werte übernehmen"+"Konfiguration speichern" betätigt wurden.

Wenn sehr viele Signale zu projektieren sind, ist die Eingabe der Adressen und Beschreibungen über das Dialogfenster sehr mühsam. Es gibt eine Möglichkeit, sich die Arbeit zu erleichtern:

In dem Verzeichnis ...\configuration im Programmverzeichnis von ibaLogic gibt es die Datei dynconf.cfg, in der die RM-Einstellungen in ASCII-Form abgespeichert sind.

Es handelt sich um eine csv-Datei, die direkt mit Excel geöffnet werden kann (z.B. vorher die Dateiendung in .csv umbenennen). In Excel können die Einstellungsinformationen dann mit tabellarischen Mittel viel effizienter bearbeitet werden. Anschließend speichert man die Datei wieder unter dem Originalnamen ab.

↗ *Siehe dazu auch 5.1.6*

2.6.6. TCP/IP Out Einstellungen

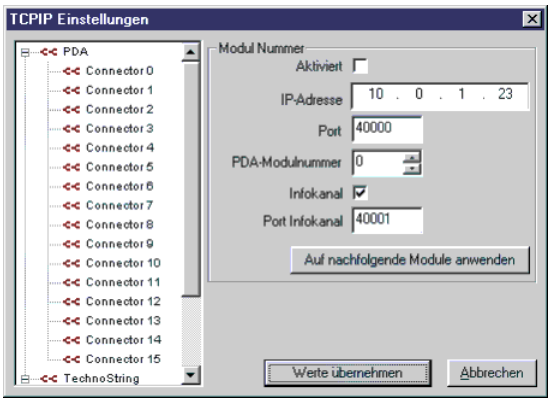


Bild 40 TCP/IP Out Einstellungen

In dieser Maske sind die Einstellungen für die Nutzung von Ausgabesignalen über eine TCP/IP-Verbindung vorzunehmen.

Entsprechend der Ausgaberessourcen TCP/IP OUT (siehe Abschnitt 5.2.5) besteht die Möglichkeit bis zu 16 Module mit je 32 analogen und 32 digitalen Signalen an ein ibaPDA-System zu senden. Außerdem stehen zusätzlich noch vier String-Variablen für Technostring-Ausgaben zur Verfügung. Damit diese Ausgaberessourcen genutzt werden können, müssen die TCP/IP-Kanäle konfiguriert und aktiviert werden. Für die 16 Module mit Ausgangssignalen an ein ibaPDA-System stehen 16 Kanäle zur Verfügung (Connector 0...15), die unabhängig voneinander aktiviert und konfiguriert werden können. So können die Verbindungen für ungenutzte Module deaktiviert werden, oder es können verschiedene ibaPDA-Zielsysteme (IP-Adressen) eingestellt werden.

Die an das ibaPDA-System gerichteten Daten können modulweise auf ein beliebiges Modul in ibaPDA rangiert werden.

❑ ***Modul Nummer***

- *Aktiviert*: Nur wenn das Aktivierungsbit markiert ist (Häkchen) dann wird die dazugehörige, angewählte Verbindung geöffnet.
- *IP-Adresse*: IP-Adresse des Empfängers; dies kann auch die lokale IP-Adresse sein, wenn z.B. ibaLogic und ibaPDA auf dem selben PC laufen.
- *Port*: Hier muss die gleiche Portnummer eingetragen werden, die im ibaPDA-System in den Systemeinstellungen eingetragen ist.
- *PDA-Modulnummer*: Hier die Nummer des Moduls im ibaPDA eintragen, wo die Daten ankommen sollen.
- *Infokanal*: noch ohne Funktion, in Vorbereitung für die Übertragung der Signalnamen
- *Port Infokanal*: noch ohne Funktion
- *Schaltfläche „Auf nachfolgende Module anwenden“:* Mit Betätigung dieser Schaltfläche erhalten alle Module unterhalb des aktuell angewählten dessen Einstellungen.

Veränderte Einstellungen werden nur wirksam, wenn die Schaltfläche "Konfiguration speichern" bzw. "Werte übernehmen"+"Konfiguration speichern" betätigt wur-

Um die Ausgaben von ibaLogic über TCP/IP in ibaPDA nutzen zu können, muss a) im PDA-Dongle die Option „TCPIP ibaLogic to PDA“ freigeschaltet sein, b) in den Systemeinstellungen von PDA die gleiche Portnummer eingestellt sein, c) in der Modulauswahl der Modultyp „IbaLogic“ gewählt werden

3 Arbeiten mit ibaLogic

ibaLogic stellt eine Vielzahl von Funktionen zur Verfügung. Darüber hinaus existieren fast immer mehrere Wege zum Ziel. Um erfolgreich die Automatisierungsaufgaben erfüllen zu können ist es erforderlich, die Struktur und Arbeitsweise von ibaLogic verstanden zu haben, bevor mit der Projektierung begonnen wird. Deswegen an dieser Stelle zuvor einige wichtige Hinweise:

3.1 Systemgrenzen und Randbedingungen

Bewusst wurden in ibaLogic keine Limitierungen im Sinne von Merkeranzahl, Anzahl der I/Os usw. eingebaut, wie dies sonst bei SPS-Steuerungen üblich ist.

Diese bewusste Unterlassung ist aus unserer Sicht prinzipiell von Vorteil für den Anwender, sie birgt aber auch Risiken hinsichtlich einer Überlastung.

Prinzipiell gilt, dass jedes System in seiner Verarbeitungskapazität limitiert ist, d.h. eine endliche Anzahl von Operationen in einem definierten Zeitintervall abarbeiten kann. Bei einem offenen System wie ibaLogic ergeben sich die Systemgrenzen z.B. aus dem CPU Takt, dem Hauptspeicherausbau und einer Reihe weiterer Eigenschaften der Hardware auf der ibaLogic laufen soll. Bei der Programmerstellung ist es daher wichtig, sich über das Zusammenwirken dieser Faktoren im Klaren zu sein, um nicht in allen Richtungen die Leistungsfähigkeit des Systems voll auszunutzen und es damit an die Grenzen seiner Kapazität zu bringen.

Prinzipiell gelten folgende Beschränkungen / Einschränkungen:

Die Anzeige "Berechnung%" sollte die Marke von 100% (d.h. 1.0) nicht überschreiten!
Je größer das Programm, desto eher treten Verzögerungen beim Compilieren im Rahmen von Online- Änderungen auf (nicht bei Hot Swap!).

Dabei kommt es darauf an, welche Änderung vorgenommen wird. Betrifft diese nur den einen Task so kann die Änderung durchaus schnell, d.h. ohne merkliche Verzögerung erfolgen (2-stelliger ms-Bereich). Sind jedoch z.B. mehrere Tasks betroffen, z.B. ein OPC-Konnektor wird geändert, so muss u.U. das ganze Projekt compiliert, gelinkt und located werden. Bei einem Projekt, welches ca. 350 Druckseiten umfasst, dauert dieser Vorgang ca. 1-2s (auf einem Doppel Pentium 1GHz mit ausreichend RAM und mit Eval% annähernd 100%) – dies wird voraussichtlich zu unerwünschtem Maschinenverhalten führen.

Durch Compilierungszeiten kann je nach Auslastung und Art der Änderung bei Online-Änderungen unter Umständen der Online-Prozess für einige Sekunden angehalten werden, d.h. die Ausgänge werden dann nicht aktualisiert!

Das kann Gefahr für Mensch und Maschine bedeuten!

Verwenden Sie immer den Hot Swap Mechanismus, um Änderungen des Funktionsplans bei laufender Maschine durchzuführen!

Sichern Sie das Projekt in jedem Fall gegen unbefugte Änderungen durch die Passwortschutz- und Lock- Funktion in ibaLogic ab!

Je nach Größe des Projektes kann die Erzeugung des Hot Swap Layers einige (zehn) Sekunden benötigen (obiges Beispiel ca. 20-30s in jeder Umschaltrichtung). Der Sicherheitsgewinn rechtfertigt jedoch diese Zeiten.

3.2 Wichtige Begriffe und Funktionen

ibaLogic beschreibt mittels grafischer Funktionen, Funktionsbausteinen, Makros, Linienverbindungen und Kommentaren die gewünschte Anwenderfunktionalität. Das Arbeitspaket in dem alle Tasks ablaufen, wird "Projekt" genannt.

Die Erstellung eines neuen Projektes beginnt mit der Erzeugung eines neuen Projektes. Dieses Projekt beinhaltet eine von der Applikation abhängige Anzahl von Tasks, jeweils gestartet mit einem bestimmten Zeitverhalten (Zykluszeit als Vielfaches vom ibaLogic Grundzyklus bzw. der FOB-Karte). Der Inhalt eines Projektes wird als eine Datei im Format *.lyt gespeichert. Das Projekt wird auch immer im "Structured Text (ST)"-ASCII-Format entsprechend IEC 61131-3 gespeichert.

Es ist eine der großen Innovationen von ibaLogic, dass während der grafischen Projektierung des Arbeitspaketes sofort und ohne Wartezeiten in die Offline-Berechnung (Berechnungs-Modus) zu Test- und Diagnosezwecken umgeschaltet werden kann. Die Funktionalität des erstellten Programms und das Verhalten von Funktionsbausteinen kann auf diese Art und Weise einfach und schnell einem Einzeltest unterzogen werden.

Zum Wechsel in den "Berechnungs"-Modus einfach  anklicken.

Beim Test von Verknüpfungssteuerungen ist auch die Berechnung eines Schrittes bzw. einer definierten Anzahl von Schritten möglich ("Einzelschritt / Mehrfachschritt berechnen "). Im Berechnungs-Modus werden keine Ausgänge aktiviert, es können jedoch Werte eingelesen werden.

Das Aktivieren des Arbeitspaketes und Ausgeben von Variablen an den Prozess erfolgt im "Online"-Modus. Zur Umschaltung in den "Online"-Modus muss der "*Online-Berechnung aktivieren*/ d*eaktivieren*"- Schalter betätigt werden. Die Hintergrundfarbe des Arbeitsbereichs wechselt von grau in violett und zeigt damit an, dass das Projekt im Online-Modus arbeitet.

Bei der Umschaltung in den Online-Modus werden alle Ausgänge entsprechend Projektierung gesetzt. Dies kann unerwünschte Auswirkungen auf die Maschine haben. In geschlossenen Regelkreisen sollten daher nur kleine, überschaubare Änderungen im Online-Modus vorgenommen werden, da auch die zyklische Abarbeitung beinträchtigt werden kann!

Um unerwünschte Auswirkungen auf den Prozess zu vermeiden, sollte eine "Hot Swap"-Ebene erzeugt werden. Eine "Hot Swap"-Ebene ermöglicht es, eine aktuell im Online-Modus laufende Task zu kopieren, zu ändern und anschließend wieder im laufenden Betrieb umzuschalten.

Zur Erzeugung einer "Hot Swap"-Ebene müssen folgende Arbeitsschritte ausgeführt werden.

1. In Online-Modus wechseln mit "*Online-Berechnung aktivieren*"
2. Sperren der aktuellen Online-Ebene mit "*Online-Ebene sperren*"
3. Hot Swap-Ebene erzeugen durch "*Hot Swap & Erstellen*". Durch dieses Kommando wird der Inhalt der aktuellen Online-Ebene dupliziert, ohne den Online-Modus zu verlassen. Die kopierte Hot Swap-Ebene kann nun modifiziert und ohne Auswirkungen auf den Prozess im Berechnungs-Modus getestet werden. Während der Projektierung der Hot Swap-Ebene läuft die Original-Ebene mit höchster Priorität im Hintergrund weiter.

4 Umschaltung des modifizierten Projektes mit dem Menübefehl ↪ *Hot Swap* ↪ *Auf Online-Ebene anwenden*
Die modifizierte Hot Swap-Ebene wird mit diesem Befehl sofort und im laufenden Betrieb Online geschaltet (ohne Verlust eines Regler-Zyklus).

Obwohl ibaLogic in der Lage ist, stoßfrei und ohne Verlust eines Zyklus Aufgabenpakete umzuschalten, so ist doch die Hot Swap-Umschaltung immer mit einem Restrisiko verbunden, z.B. aufgrund von Projektierungsfehlern im Anwendungsprogramm oder falschen Parametern. Es ist immer ratsam, die Umschaltung in einem sicheren Betriebszustand der Maschine vorzunehmen.

3.3 Welche Tasks sollten wie schnell ablaufen?

Die wesentliche Entscheidung über ein Projekt ist die Projektstruktur. Üblicherweise wird ein Projekt in Aufgaben (Tasks) unterteilt, welche sowohl völlig unabhängig voneinander sein können oder sich z.B. in Ihrer Dynamik wesentlich unterscheiden. Es dürfte klar sein, dass es einer Raumtemperaturregelung völlig genügt mit einem Zyklus von 1s zu laufen, während eine hydraulische Anstellung höchstens im zweistelligen Millisekundenbereich ablaufen darf. Zeit stellt also ein wesentliches Kriterium für die Aufteilung eines Projektes in Tasks dar. Der schnellste mögliche Taskzyklus beträgt 1ms.

3.4 Taskzyklus, Rechenzeit und Auslastung (Berechnung%)

iba garantiert, dass Tasks im 1ms Abstand gestartet werden können. Jedoch gilt dies nur dann, wenn weitere Bedingungen erfüllt sind.

Eine Task in ibaLogic (Version 3.xx) ist per Definition ununterbrechbar. Dies hat Auswirkungen auf die Taskzykluszeit.

Beispiel: Es wurden 2 Tasks definiert. Task0 mit 5ms und Task1 mit 100ms Zykluszeit. Task0 benötigt zur Berechnung (Evaluierung) 2ms, Task1 jedoch 8ms (diese Werte können mit Hilfe der Task-Evalierungsstatistik ermittelt werden). Task1 – ununterbrechbar – rechnet also länger als dies für Task0 vorgesehen ist (5ms). Task 0 muss warten, bis Task 1 fertig ist. Demzufolge zeigt die Auslastungsanzeige "Berechnung%" auch mehr als 100% an, da diese Anzeige die längste Rechenzeit in Beziehung zum schnellsten Task anzeigt. Für die Bausteinberechnung stellt dies kein Problem dar, da alle mitgelieferten Bausteine Zeitrelativ programmiert sind (jeder Baustein fragt nach, wieviel Zeit seit seinem letzten Start vergangen ist und verhält sich entsprechend - dies ist bei der Erstellung eigener Bausteine mit Zeitgliedern zu beachten!). Jedoch könnte die Verdrängung dieser Task andere Probleme auslösen, wie wenn z.B. ein Impuls von 5ms definierter Länge damit erzeugt werden soll, indem in einem Zyklus ein- und im nächsten Zyklus ausgeschaltet wird. Solche Probleme lassen sich jedoch eleganter mit Pulsformerfunktionen von z.B. Padu8 O absolut zeitrichtig realisieren.

Damit ergibt sich folgende Faustregel: Evaluierung% sollte 100% nicht überschreiten, da sonst Verdrängungen vorprogrammiert sind! Hinzu kommt die Gefahr anderer Randeffekte, wie unter 3.1 beschrieben.

Im Idealfall sollte (muss jedoch nicht) die Summe der Tasklaufzeiten kleiner sein als die schnellste Taskzykluszeit. Ansonsten können sich ggf. Verdrängungen durch die Überlagerungen der Zyklen und Rechenzeiten ergeben.

Die aktuellen Tasklaufzeiten können mit ↪*Ansicht* ↪*Berechnungsstatistik* ermittelt werden.

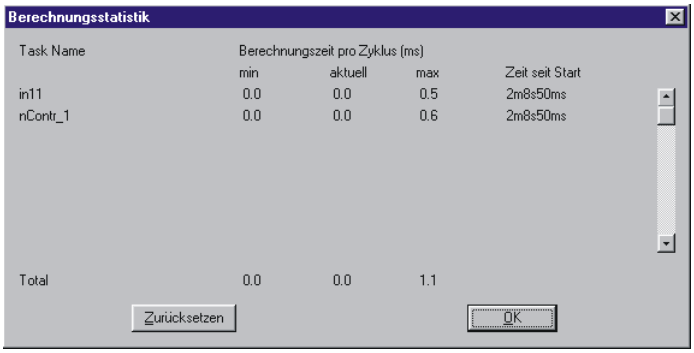
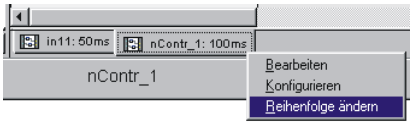


Bild 41 Berechnungsstatistik

3.4.1. Reihenfolge der Abarbeitung von Tasks

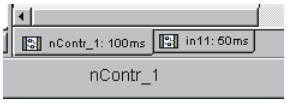
Aus den vorangegangenen Bedingungen kann es sich ergeben, dass die Abarbeitung der Tasks verändert werden muss. Im Normalfall werden die Tasks von links nach rechts abgearbeitet. Die Reihenfolge läßt sich wie folgt ändern:

1 Mit rechter Maustaste auf den Reiter der gewünschten Task klicken (hier Task "nContr_1"). Das Kontextmenü erscheint. "Reihenfolge ändern" anklicken.



2 Dann mit linker Maustaste die Stelle (hier Task "in11") anwählen, die der Task erhalten soll (Cursor verändert sein Form)

3 Die linke Maustaste drücken.



3.5 Das ibaLogic I/O System

Prinzipiell gilt, dass das iba I/O System autark von der PC -Verarbeitung Daten empfangen kann (für Sendedaten ist natürlich eine Applikation erforderlich). Dies geschieht im allgemeinen in einem Zyklus von 1 ms. Signale werden also auch dann transportiert, wenn keine PC Applikation gestartet ist. Ausnahme: Bei Verbindungen, die eine bidirektionale Kommunikation erfordern, z.B. Padu8 M, muss die Softwareapplikation auf dem PC laufen.

In der folgenden Tabelle sind die Komponenten des I/O-Systems von ibaLogic mit den entsprechenden Anschaltbaugruppen aufgeführt.

Peripheriebaugruppe	Anschaltbaugruppe	Ein- und / oder Ausgaben
Padu8, -16, -32	FOB 4i PCI, FOB IO	E
Padu8 M , -ICP	FOB 4i + FOB 4o PCI, FOB IO	E (Ausgabemodule nur für Konfiguration)
Padu8 O	FOB 4o PCI, FOB IO	A
ibaNet 750	FOB 4i + FOB 4o PCI, FOB IO	E / A
SM 64 IO	FOB 4i + FOB 4o PCI, FOB IO	E / A
SM 128 V	FOB 4i + FOB 4o PCI, FOB IO	E / A
CS12/14/16 (Simadyn D) SM64-SD16 Simadyn D (16 Bit)	FOB SD PCI FOB 4i + FOB 4o PCI, FOB IO	E / A E / A
Simatic TDC	FOB TDC PCI	E / A
Simatic S5, MMC	FOB 4i + FOB 4o PCI, FOB IO	E / A
Simatic S7, Profibus	L2B x/8 PCI, DPM64+FOB	E / A

Tabelle 4 I/O-Komponenten

3.5.1. Namen und Kennzeichnung der I/O Ressourcen (Ortskennzeichen)

ibaLogic bietet mehrere Möglichkeiten der Bezeichnung von Ressourcen und I/O Signalen. Prinzipiell können 32 ASCII Zeichen pro Signal verwendet werden. Sonderzeichen und Leerzeichen innerhalb von Namen sind erlaubt.

- ❑ Die Ressourcen können noch im Ressourcenbaum umbenannt werden (gewünschte Variable zweimal anklicken). Wird die Ressource (auch mehrfach) in den Plan gezogen so erhält das (erhalten die) I/O Signal(e) diesen Namen.
- ❑ Gruppen von Ressourcen können als CSV-Dateien exportiert werden *(eine Ressource der Gruppe selektieren / rechte Maustaste / Export)*. Es entsteht eine CSV-Datei, die mit einem ASCII-Editor oder z.B. mit MS Excel bearbeitet werden kann. Wird die Datei dann wieder als CSV-Datei gespeichert, kann ibaLogic sie mit dem Befehl ↪*Ansicht* ↪*Ressourcenbeschreibung laden* importieren. Es können Signale und Signalgruppen (Modulnamen) umbenannt werden. Diese Funktion ist hilfreich, wenn viele Signale editiert werden müssen.

Bitte beachten, dass die geschriebene und die gelesene Datei identisch sind (oft merkt sich Excel den Pfad der gelesenen Datei nicht und legt ihn unter "Eigene Dateien" ab – das ist oft verwirrend!)

Üblicherweise legt ibaLogic die Dateien im Pfad configuration ab.

- ❑ Mit dem Befehl ↪*Ansicht* ↪*Ressourcenbeschreibung abgleichen* können Ressourcennamen sowohl aus dem als auch in den geladenen Plan übernommen werden. Diese Funktion ist dann hilfreich, wenn Pläne anderer Projekteure eingelesen werden, bzw. „standardisierte“ Projekte an unterschiedliche I/O Systeme angepasst werden sollen. Achtung: Als Bindeglied dient hier ausschließlich der interne Variablenname von ibaLogic!
- ❑ Ein im Plan platziertes I/O-Signal kann auch individuell umbenannt werden (Doppelklick auf Signal). Da es sich um eine individuelle Umbenennung handelt, können hier u.U. für das gleiche I/O Signal mehrere Bezeichnungen existieren!

Individuelle Namensänderungen werden u.U. wieder rückgängig gemacht, wenn danach nochmals ein Ressourcenabgleich von den Namen zum Plan durchgeführt wird!

3.6 Die Betriebsarten von ibalogic

ibaLogic bietet eine Reihe von Betriebsarten, um den unterschiedlichen Anforderungen verschiedener Anwendungen Rechnung zu tragen. Da ibaLogic nicht nur als Soft-SPS sondern auch als Signalmanager, Signalprozessor oder Simulator arbeiten soll, wurden mehrere Verarbeitungsmodi implementiert.

3.6.1. Signalmanager

Dieser Modus stellt sicher, dass ibaLogic kein Eingangssample verliert. Dies gilt auch dann, falls einzelne Tasks innerhalb ibaLogic verdrängt werden sollten. Das Ablaufsystem von ibaLogic stellt hier sicher, dass die Daten äquidistant im eingestellten Samplingzyklus zur Verfügung stehen. Bei Taskverdrängungen werden Zyklen sogar nachgeholt. Es kann demzufolge im ungünstigsten Fall dazu kommen, dass ibaLogic nur noch Werte der „Vergangenheit“ berechnet. Jedoch ist immer sichergestellt, dass z.B. für FFT-Analysen äquidistant korrekte Werte zur Verfügung stehen.

Ausgangswerte werden von jeder Task direkt am Ende des Taskzyklus geschrieben (falls Ausgangsressourcen im Plan angeschlossen sind).

3.6.2. SOFT-PLC

In diesem Modus, welcher für Regelungs- und Steuerungsaufgaben geeignet ist, stellt ibaLogic sicher, dass nur die jüngsten Signalzustände verarbeitet werden. Im Gegensatz zum Signalmanager-Modus kommt es hier nicht darauf an, ob Samples verloren gehen oder nicht. Ganz im Gegenteil ist es erwünscht, nur möglichst aktuelle Daten – also aus dem letzten I/O Transferzyklus - zu erhalten.

Die erste Task eines neuen Zyklus liest die Eingangsressourcen ein. Die Alterungszeit der Ressourcen ist durch den Basis Samplingzyklus bestimmt, welcher in den Einstellungen der Hardware festgelegt wird. Ist diese Samplingzeit z.B. 10ms und der erste Task mit Zyklus 50ms parametriert, so findet der erste Task Eingangsdaten vor, die garantiert nicht älter als 10ms sind. Sie können jedoch jünger sein.

Ausgangswerte werden von jeder Task direkt am Ende des Taskzyklus geschrieben (falls Ausgangsressourcen im Plan angeschlossen sind).

3.6.3. Turbo Modus

Der Turbo-Modus kann aktiviert werden, wenn ein PC mit Doppelprozessor verwendet wird. Die Leistungsreserven und die Stabilität des Systems können damit deutlich erhöht werden, weil einer der Prozessoren ausschließlich für die Abarbeitung des Programms (Runtime) zuständig ist und der andere die übliche Windows-Verwaltung übernimmt. Insbesondere bei Steuerungs- und Regelungsaufgaben (Soft-PLC-Modus) sollte man davon Gebrauch machen.

3.6.4. Playback

Der Playback-Modus ist eine besonders hilfreiche Einrichtung für die Simulation von Prozessen. Im Playback-Modus kann eine Messdatei, die mit einem Online-Messsystem von iba, wie z.B. ibaPDA, ibaQDR, ibaScope usw. erzeugt wurde, ähnlich einem Tonband abgespielt und als Eingangssignalquelle verwendet werden. Das Besondere daran ist, dass es sich dabei um reale Daten einer Anlage oder eines Prozesses handelt, so dass man für Simulations- oder Testaufbauten eine weitaus höhere Realitätstreue erzielt als mit Modellrechnungen. Dies ist vor allem für Modernisierungsprojekte interessant. Ausserdem lassen sich im Playbackbetrieb die Signale aus einer Datei mit Signalen der Hardwareeingaben mischen.

Verwendung der Playbackfunktion

1 Voraussetzung für die Nutzung der Playbackfunktion ist, dass in den Systemeinstellungen (Menü ↪Datei ↪Systemeinstellungen ↪Allgemein) die Betriebsart "Playback" angewählt wurde. (siehe Kapitel 2.5.1)

2 Außerdem sollte in den Systemeinstellungen (...↪Sonstige) die Auswahl zwischen Playbackbetrieb mit oder ohne Hardware-E/A getroffen worden sein. (siehe Kapitel 2.5.2)

3 Die Konfiguration der Playback-Funktion wird im Menü ↪Datei ↪Programmeinstellungen ↪Playback vorgenommen (vergl. Kap. 2.4.4).
Wenn eine gültige Messdatei vorhanden ist, dann werden die wichtigsten Daten, wie Startzeit, Sampletime und Anzahl Messpunkte in dem Dialogfenster angezeigt.

4 Sofern noch kein bestimmter Zeitraum in der Messdatei von Interesse ist, sollte die manuelle Eingabe von Start- und Endzeit für die Wiedergabe zunächst deaktiviert bleiben. Wiedergabe und Einspielmodus wählen.

5 Nun die Modul-Rangierung vornehmen, um die aufgezeichneten Signale, die ja mit Modul- und Kanalbezeichnungen identifiziert sind, den Eingangsressourcen von ibaLogic zuzuordnen.

Modulrangierung für Playback

Mit Klicken auf die Schaltfläche "Modul Rangierung >>" im Playback-Dialogfenster erscheint der folgende Dialog:

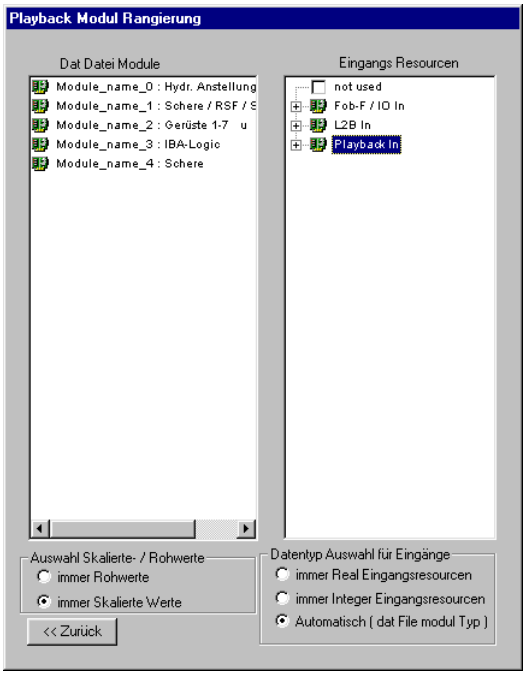


Bild 42 Playback Modul Rangierung

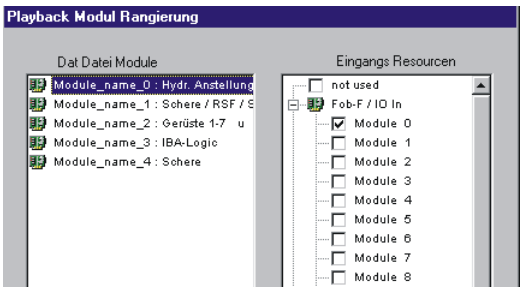
Im linken Teil des Fensters werden die Module des Messsystems angezeigt, so, wie sie in der Messdatei vorliegen, bzw. mit ibaPDA definiert wurden. Auch die Modulnamen sind sichtbar. Die einzelnen Signale können nicht betrachtet werden.

Im rechten Teil des Fensters werden die Eingangsressourcen angezeigt, die für die Playback-Funktion verwendet werden können. Es sind ausschließlich die Typen FOB-F bzw. FOB IO In, L2B In und Playback.

Die Signalrangierung erfolgt modulweise wie folgt:

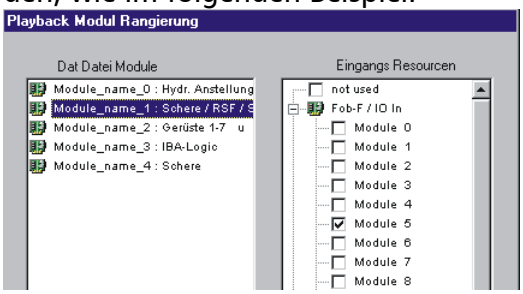
1 Zunächst ein Modul der Datei auf der linken Seite markieren.

2 Dann auf der rechten Seite den Modulbaum mit Klick auf das kleine Pluszeichen öffnen und ein Häkchen in das Modul setzen, das dem markierten Dateimodul zugeordnet werden soll.
Beispiel: Alle Signale vom Dateimodul 0 sollen auch in ibaLogic dem Modul 0 der FOB-F-Eingangsressourcen zugeordnet werden.



Die Rangierung der Datei-Module auf die FOB-F-Eingangsmodule sollte nur gewählt werden, wenn In den Systemeinstellungen der Playbackbetrieb ohne HW E/A gewählt wurde, da die Hardware-Eingangssignale sonst von den Dateisignalen im Playbackbetrieb überschrieben werden. Wenn Hardware-Eingangssignale und Dateisignale im Playbackbetrieb gleichzeitig verarbeitet werden sollen (Mischbetrieb), dann sollten die Dateimodule auf die dafür vorgesehenen PlaybackIn-Module rangiert werden.

3 Die Modulnummern müssen sich jedoch nicht gleichen. Beispielsweise kann das Dateimodul 1 auch dem ibaLogic-Eingangsmodul 5 zugeordnet werden, wie im folgenden Beispiel.



4 Nach Abschluss der Modul-Rangierung können noch die Art der Werte und der Datentyp für die Eingänge bestimmt werden.

- *immer Rohwerte*: ibaLogic nimmt die Signalwerte, wie sie in der Messdatei stehen. Mit dieser Einstellung wird vermieden, dass die Signale, die bereits in PDA skaliert wurden, bzw. in physikalischen Einheiten vorliegen, nochmals in ibaLogic skaliert werden.
- *immer Skalierte Werte*: ibaLogic nimmt die Signalwerte aus der Messdatei und skaliert sie mit den Werten „minscale“ und „maxscale“, die auch in der Messdatei enthalten sind.
- *immer Real Eingangsressourcen*: Alle (Analog-) Eingangsressourcen werden im Real-Format verrechnet.
- *immer Integer Eingangsressourcen*: Alle (Analog-) Eingangsressourcen werden im Integer-Format verrechnet.
- *Automatisch*: Die Eingangsressourcen werden in dem Format verrechnet, wie es in der Messdatei definiert ist.

Diese fünf Einstellungen können in verschiedenen Kombinationen auftreten, die jedoch nicht alle sinnvoll sind:

Datentyp in der Mesdatei	immer Rohwerte	immer skalierte Werte	immer Real	immer Integer	immer Automatisch
INT16		●	●		
INT16		●			●
INT16	●			●	
REAL	●		●		

● = sinnvolle Kombinationen

Der Playback-Betrieb startet schließlich mit Start der Evaluierung bzw. der Aktivierung des Online-Modus.

3.7 Verhalten bei Störungen

3.7.1. Nullen bei Verbindungsabbruch

Die Aktivierung dieser Option bewirkt ein Rücksetzen aller Eingangssignale eines Moduls auf "Null" im Falle einer Kommunikationsstörung zu einem der angeschlossenen Peripheriekomponenten (nur FOB-F). Damit lässt sich im Störungsfall ein definierter Zustand auf der Eingabeseite herbeiführen. Wenn die Option nicht gewählt ist, dann bleiben im Störungsfall die zuletzt gültigen Werte erhalten.

3.7.2. Nicht verfügbare Signale sind invalid

Signale sind dann nicht verfügbar, wenn die PC-Steckkarte, der die Signale zugeordnet sind (Eingangsressourcen), nicht vorhanden oder nicht erreichbar ist.

Wenn eine PC-Karte gesteckt ist und funktioniert, dann gelten die zugehörigen Signale als verfügbar.

Das Abziehen einer Lichtleiterverbindung oder das Abschalten eines Padus bewirkt nicht, dass Signale als „nicht verfügbar" deklariert werden!

Die Signale werden im Layout als „invalid" gekennzeichnet (rote Umrandung). Da der Status „invalid" einer Variablen vererbt werden kann, werden auch alle Variablen als „invalid" gekennzeichnet, die aus Berechnungen und Verknüpfungen mit einer „invalid"-Variablen hervorgehen.

3.8 ibaLogic Handling

3.8.1. Drag & Drop

Die Bedienung von ibaLogic erfolgt größtenteils über das bei Windows-Programmen übliche "Drag & Drop"-Verfahren, d.h. Bausteine, Ein- und Ausgänge usw. werden mit der linken Maustaste selektiert und an den gewünschten Zielort in der Ein- oder Ausgangsrandleiste bzw. in den Arbeitsbereich verschoben.

3.8.2. Rechte Maustaste

Bei Betätigung der rechten Maustaste in den Ein-/Ausgangsrandleisten erscheint ein Kontextmenü, das mit dem "*Bearbeiten*"-Menü übereinstimmt. Beschreibung siehe 2.3.2.

Wird die rechte Maustaste auf einem Ein- oder Ausgangssignal im Ressourcenbereich benutzt, so kann die Ressourcen-Exportfunktion aktiviert werden, wie in Abschnitt 3.5.1 beschrieben.

Ein rechter Mausklick auf den Reiter einer Task in der Taskauswahlleiste öffnet ein Menü mit verschiedenen Task-Einstellmöglichkeiten, wie in Abschnitt 3.4.1 bzw. 3.8.3 beschrieben.

3.8.3. Größe des Arbeitsbereichs pro Task festlegen

Der Arbeitsbereich einer Task besteht bei der Initialisierung aus einer Seite. Reicht diese Größe für das Anwenderprogramm nicht aus, kann der Arbeitsbereich für jede Task individuell angepasst werden. Dies erfolgt entweder

1 durch Ziehen des Cursors an den rechten oder untersten Seitenrändern einer Task (Cursor ändert seine Form in ↔ bzw. ↕) oder

4 mittels Kommando ↪*Bearbeiten* ↪*Task* ↪*Task einstellen*. Die horizontale und vertikale Seitengröße kann im nun geöffneten Dialogfenster "Task Einstellungen" definiert werden (im Beispiel unten 10 Seiten insgesamt, davon 2 Seiten horizontal, 5 Seiten vertikal).

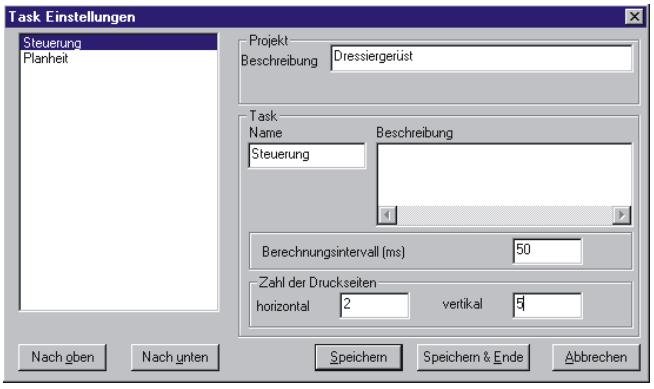


Bild 43 Task Einstellungen

3.9 Auswahl und Verschaltung von Funktionsbausteinen

Die Projektierung der Anwenderfunktionalität erfolgt mittels grafischer Funktionsbausteine. In der Sektion „*Ressourcen*“ wird über die Lasche „*Funktionen*“ vom Ein- oder Ausgangsressourcenbereich auf den Bausteinkatalog umgeschaltet. Zur besseren Übersichtlichkeit ist die Summe aller Funktionsbausteine in sieben Bereiche gegliedert:

- ❑ Basic Functions
- ❑ Basic FBs
- ❑ Global Variables
- ❑ Global FBs and Macros
- ❑ Global DLLs
- ❑ Local FBs and Macros
- ❑ Local DLLs

Die zur Verfügung stehenden Bausteine sind im Kapitel 4 ausführlich beschrieben.

Nach Auswahl des gewünschten Bausteins (im Beispiel Multiplizierer "mul") aus dem Bausteinkatalog "Basic Functions & arithmetics" wird der Baustein einfach über das bei Windows übliche Drag&Drop-Verfahren mit der linken Maustaste selektiert und in den Arbeitsbereich verschoben. Auf diese Art und Weise können weitere Bausteine in den Arbeitsbereich platziert werden

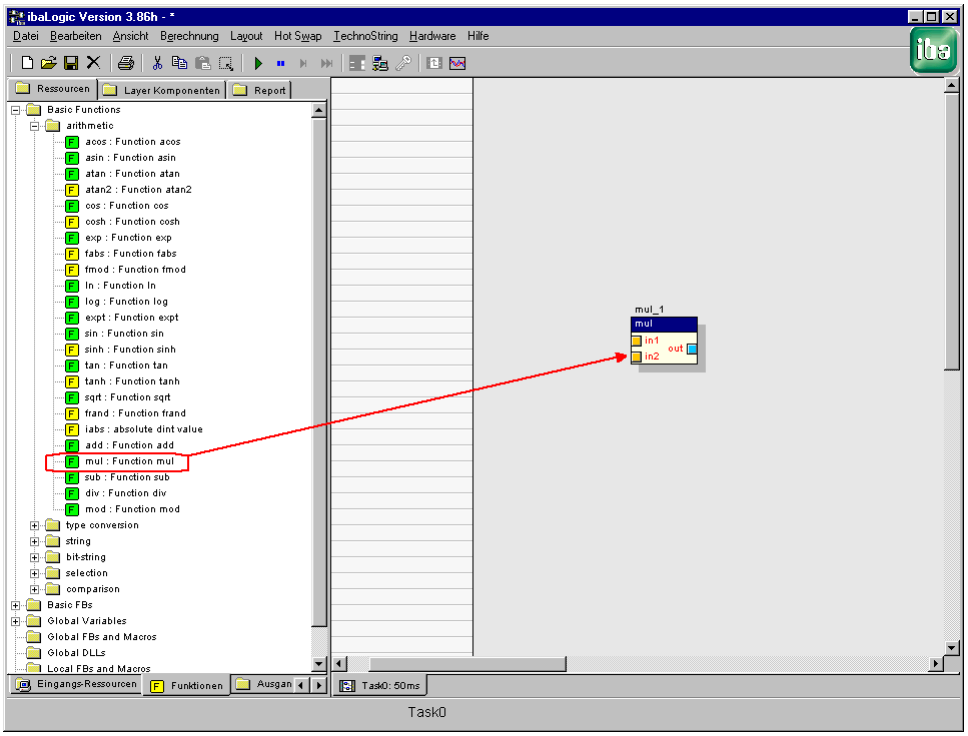


Bild 44 Platzieren eines Funktionsbausteins im Layout

3.9.1. Verbindungslinien und Verzweigungen

ibaLogic stellt drei Klassen von Verbindungen zur Verfügung – Verbindungslinien, IntraPage-Konnektoren und OffTask Konnektoren.

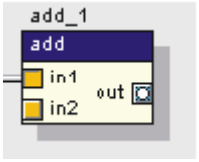
Über grafische Linienverbindungen werden die Funktionsbausteine miteinander verschaltet. Zu diesem Zweck wählt man den Ein- oder Ausgang eines Bausteins an und zieht mit der nun automatisch dargestellten Linie zum Aus- bzw. Eingang eines weiteren Bausteins.

ibaLogic kennt drei Linientypen, welche unterschiedliche Gruppen von Datentypen repräsentieren und in unterschiedlichen Farben dargestellt werden:

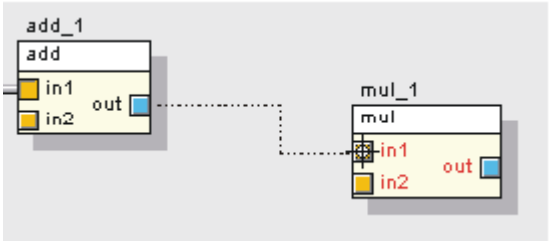
- ❑ Binäre Verbindungslinien; diese zeigen direkt den logischen Status der Linie in diesem Augenblick an
  Blau = aktiv Low; Rot = aktiv High
- ❑ Alle anderen skalaren Datentypen werden durch graue Verbindungslinien charakterisiert (z.B. INT, REAL...)
- ❑ Arrays, also Vektoren werden mit grünen Linienzügen dargestellt. Es könne nur Arrays gleicher Länge und gleichen Datentyps miteinander verbunden werden. Ändert sich z.B. die Größe eines Arrays so muss die Verbindung zunächst gelöscht und anschließend erneut eingefügt werden.

Das *Ziehen* von Linien erfolgt, indem die Maus über das entsprechende sensitive Feld eines Bausteines oder einer I/O-Ressource gezogen wird (Cursorform wechselt zu ¤), die linke Maustaste gedrückt und dann die Maus zum Zielbaustein gezogen und die Maustaste dort losgelassen wird. (Wenn ein möglicher Anschlusspunkt erkannt wird, ändert sich der Cursor zum Fadenkreuz). Die Linie wird vom Autorouter automatisch eingefügt.

Startpunkt der Linie



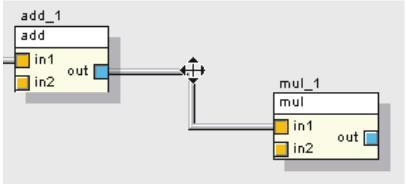
Zielpunkt der Linie



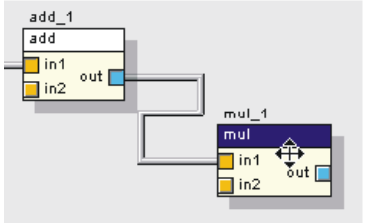
Soll der *Linienverlauf* geändert werden, dann müssen dazu die sensitiven Bereiche von Kreuzungen bzw. von Linienknicken angewählt werden (Cursor verändert sich zu einem Symbol mit 4 Pfeilen) und dann der Linienverlauf in die gewünschte Richtung beeinflusst werden (linke Maustaste drücken und Linie verschieben, dabei Maustaste gedrückt halten).

Liegen die zu verbindenden Objekte zu dicht beieinander, so kann es dazu kommen, dass der Router Schlaufen oder meanderförmige Verbindungen erzeugt. In diesem Fall die Objekte weiter auseinander rücken.

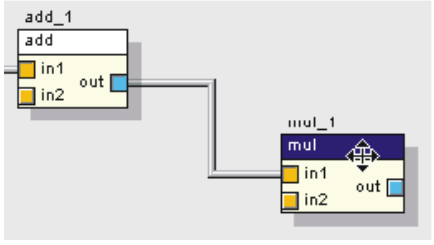
Verändern der Linienführung



Linienführung, Objekte zu eng

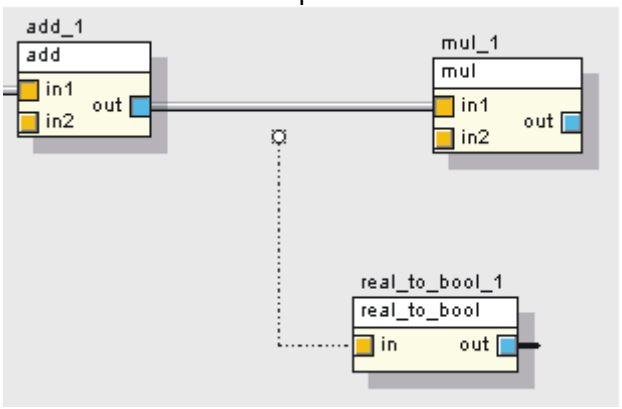


Bessere Linienführung, Objekte entfernt

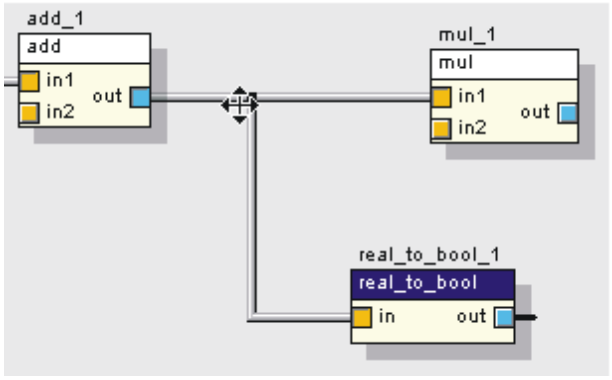


Verzweigungen von Linien werden erzeugt, indem vom Ziel aus rückwärts zu dem Punkt der Hauptlinie gezogen wird, an dem der Abzweig entstehen soll. An dieser Stelle entsteht ein Punkt auf der Verbindungslinie. Auch dieser Punkt läßt sich entlang der Linie verschieben.

Ziehen der Verzweigungslinie vom Endpunkt zur Hauptlinie



Verschieben des Verzweigungspunktes



Zum *Löschen* einer Linie diese am Anfangs- oder Endpunkt selektieren und auf einen freien Platz innerhalb des Planbereiches ziehen. Die (Teil-)Linie verschwindet dann.

Sollen Abzweige oder Knicke von Linien im Plan *fixiert* werden, dann können sie gewissermaßen „festgenagelt" werden. Dazu den gewünschten sensitiven Bereich auf der Linie anwählen und die rechte Maustaste anklicken. Ein kleines Kreuz markiert den fixierten Punkt. Zum Löschen die gleiche Operation an gleicher Stelle durchführen. Werden nun Objekte verschoben, dann bleiben solche Linienpunkte fixiert.

ibaLogic erkennt automatisch, ob das Datenformat von Ein- und Ausgang übereinstimmen. Bei unterschiedlichen Datenformaten wird von ibaLogic die in *↪Datei ↪Einstellungen ↪Konvertierungen* definierte Aktion ausgeführt.

ibaLogic verfügt über "Autorouting", d.h. beim Verschieben eines Bausteins werden die dazugehörigen Verschaltungen automatisch mit verschoben. Falls gewünscht, können die Linienverbindungen auch manuell in ihrer Lage verändert werden.

Bausteine mit untypisierten Ein- und Ausgängen erhalten den Datentyp für die Ein- und Ausgangskonnektoren erst mit dem Anschluss einer Quelle oder eines Ziels, welche/welches einen definierten Datentyp besitzt.

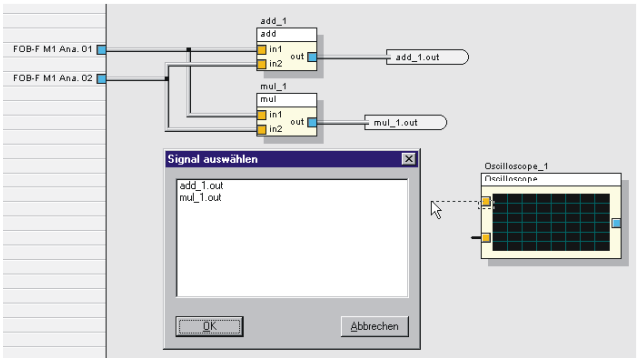
Umgekehrt verlieren diese Bausteine auch wieder die Datentypdefinition, wenn alle typbestimmenden Verbindungen abgetrennt wurden. Gleichzeitig gehen damit auch Defaultwerte verloren, da Defaultwerte nur mit Datentypisierung möglich sind!

3.9.2. IntraPage-Konnectoren (IPC)

Ein IntraPage-Konnektor (IPC) stellt lediglich eine zeichnerische Vereinfachung dar – der IPC ersetzt dabei eine Verbindungslinie. Dies ist insbesondere vorteilhaft, wenn sehr viele Objekte auf einer Seite verbunden werden müssen oder „lange" Verbindungen über mehrere Seiten erforderlich sind. Der IPC ist somit kein spezielles Objekt sondern existiert nur als Linienersatz. Der IPC kann deshalb auch nur Objekte innerhalb einer Hierarchieebene verbinden, z.B. Objekte innerhalb einer Task, Objekte innerhalb eines Makrobausteines – solche Objekte, welche sich auch sonst durch direkte Linienzüge verbinden lassen. Er kann nicht hierarchieübergreifend angewendet werden, z.B. zur Verbindung von einem Baustein innerhalb eines Makros zu einem Baustein außerhalb des Makros.

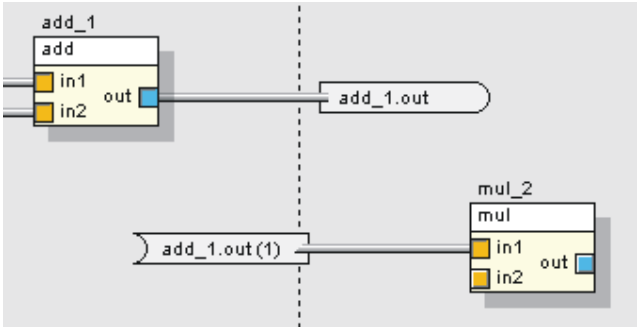
Ein IPC kann auf drei Arten erzeugt werden:

1



Durch drücken der Taste ALT und Ziehen einer Linie, ausgehend von einem Startpunkt (das Ende zeigt auf eine freie Stelle im Planbereich). Ein Startpunkt ist z.B. üblicherweise der Ausgang eines Funktionsblockes. Auf diese Weise erhält man einen „Sende-IPC". Um das Gegenstück, einen Empfangs-IPC, zu kreieren, verfährt man in entsprechender Weise vom Zielpunkt ausgehend. Existieren schon Sende-IPCs (z.B. die Sender *add_1.out* und *mul_1.out*), so wird eine Maske angezeigt, aus welcher dann der gewünschte IPC ausgewählt werden kann.

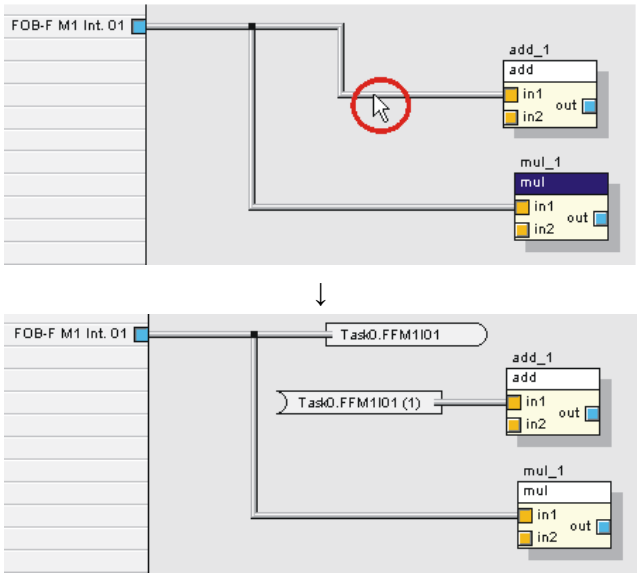
2



... oder man verbindet zwei Objekte miteinander, die auf unterschiedlichen Seiten liegen.

(Bereits bestehende Verbindungen werden durch das Ziehen des Bausteins auf eine andere Seite nicht automatisch aufgebrochen und durch IPCs ersetzt!)

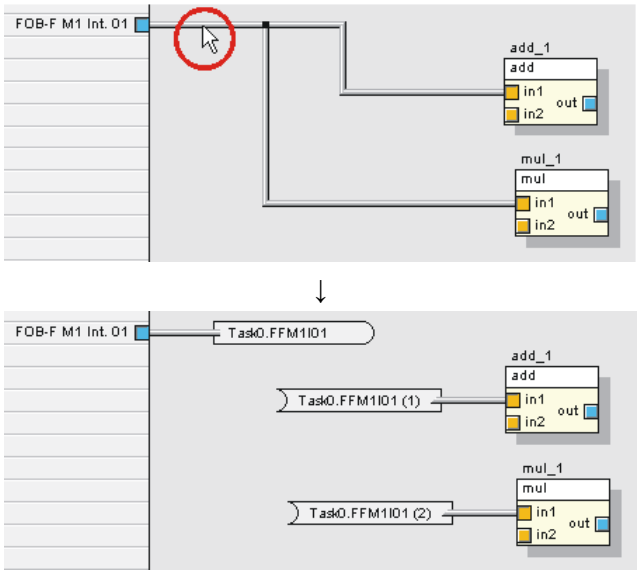
3 a)



... oder durch drücken der Taste ALT und anschließendes Selektieren einer existierenden Verbindungslinie. Diese Linie bzw. die komplette Linienstruktur (Netz) wird dadurch aufgesplittet.

Achtung: Bei verzweigten Netzwerken hängt das Ergebnis der Auftrennung von der Stelle ab, wo der IPC definiert wird! Selektiert man eine Punkt-zu-Punkt-Verbindung oder eine Linie hinter einem Verzweigungspunkt, so entstehen nur ein Sende- und ein Empfangs-IPC.(a)

b)



Wenn die Linie vor einem Verzweigungspunkt selektiert wird, dann werden ein Sende-IPC und so viele Empfangs-IPCs erzeugt, wie Verzweigungen vorhanden sind.(b)

Dem entstandenen IPC wird zunächst ein Name zugewiesen, der abhängig von seiner Anbindung ist, z.B. *Funktionsbaustein.Konnektorname* oder *Taskname.Bezeichner*. Er kann aber durch editieren einen individuellen Namen erhalten. Auch die Größe (Länge des IPC) läßt sich mit den üblichen Methoden verändern, bzw. bereits in den Systemeinstellungen unter ↪*Datei* ↪*Einstellungen* ↪*Bearbeitungs-Einstellungen* vorwählen.

Der IPC wird mit dem gleichen Mechanismus gelöscht wie eine Verbindungslinie. Die Quelle der IPC Linie am Baustein wird selektiert und auf eine freie Stelle innerhalb des Plans gezogen. Die Linie und ihr IPC verschwinden. Wird der Quell-IPC gelöscht, dann werden auch alle Empfänger gelöscht.

Ein Quell-IPC kann erst gelöscht werden, wenn er keine Ziele mehr aufweist.

3.9.3. OffTask-Konnektoren und OPC-Verbindungen

OffTask-Konnektoren (OTC) oder „Task-Task-Konnektoren“ dienen zunächst als taskübergreifende Verbindungselemente. Sie sind immer dann erforderlich, wenn zwischen mehreren Tasks kommuniziert werden soll.

3.9.3.1. Erstellen eines OffTask-Konnektors

1 Platzieren des Mauszeigers an einer freien Stelle im Plan.

2 Öffnen des Menüs ↪*Bearbeiten* ↪*Neu* ↪*Task-Task-Konnektor*" (oder über Kontext-Menü); das unten stehende Dialogfenster wird geöffnet.

3 Soll ein neuer Quell-Konnektor erzeugt werden, dann gibt man zunächst einen eindeutigen Namen in das Feld „Name“ ein. Falls bereits ein Quell-Konnektor mit diesem Namen existieren sollte, gibt ibaLogic eine Fehlermeldung aus und nimmt den Namen nicht an.

4 Um einen Ziel-Konnektor zu erstellen gibt es zwei Möglichkeiten:
a) Den Quell-Konnektor markieren und in die Zwischenablage kopieren, dann in die Task wechseln, wo der Ziel-Konnektor liegen soll und aus der Zwischenablage einfügen (Copy&Paste). Der Konnektor wird automatisch als Ziel-Konnektor eingefügt.
b) In der Zieltask wieder das Menü wie unter Pkt. 2 aufrufen und im Feld „Name“ die Pick-Liste öffnen; dann den gewünschten (Quell-)Konnektor auswählen und das Feld „Ausgangsquelle“ deaktivieren.

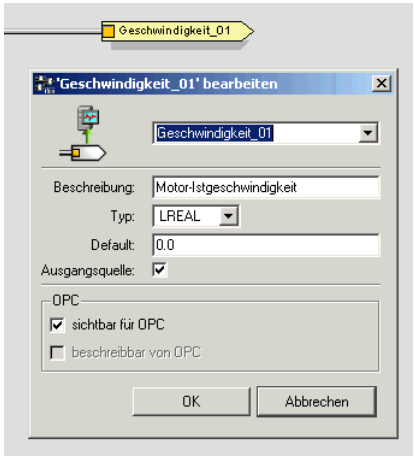
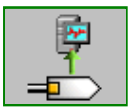


Bild 45 Dialog zur Konfiguration von Off-Task-Konnektoren

❑ ***Einstellungen***

- *Beschreibung*: Hier kann ein erläuternder Klartext eingegeben werden. Dieser Text erscheint auch im Tooltip, wenn der Mauszeiger auf dem Anschluss des OTC positioniert wird.
- *Typ*: Hier kann über Auswahl aus der Pick-Liste der gewünschte Datentyp eingestellt werden.
- *Default*: Anzeige bzw. Eingabemöglichkeit für den Startwert des OTCs. Nach Programminitialisierung nimmt der OTC diesen Defaultwert an. Wenn in den Programmeinstellungen (↪*Datei* ↪*Programmeinstellungen* ↪*Bearbeitung*) die Option "*OPC-Schreiben setzt Defaultwerte*" aktiviert wurde, dann kann dieser Wert von einem OPC-Client (z.B. HMI) überschrieben werden.
- *Ausgangsquelle*: Häkchen machen, wenn der OTC Daten ausgeben soll. Bei Definition von Eingangs-Konnektoren muss diese deaktiviert werden

Da OffTask-Konnektoren auch die Verbindungen von/zu OPC herstellen, existieren zwei weitere Optionen:



- *OPC sichtbar*: ...legt fest, ob dieser Konnektor im OPC Namensraum sichtbar ist oder nicht. Das OPC-Symbol wechselt, wenn diese Option gewäjhlt wird (siehe links).
- *OPC->ibaLogic*: ...legt fest, ob der Konnektor (der dann natürlich nur ein Eingangskonnektor sein kann) von OPC beschrieben werden kann oder nicht.

Damit können OTCs zur Kommunikation mit HMI-Systemen genutzt werden. ibaLogic ist dabei stets OPC-Server. OTCs mit Attribut *OPC sichtbar* sind innerhalb der OPC-Browser am OPC-Client sichtbar und können angewählt werden.

Ist *OPC->ibaLogic* aktiviert, kann ein HMI Daten an ibaLogic senden. Die Option *OPC->ibaLogic* kann nur bei Eingangs-OTCs aktiviert werden, wenn zu denen noch keine Quell-OTCs existieren.

Innerhalb einer Task kann jeweils nur eine Instanz eines OTC existieren (also z.B. keine zwei Empfänger gleichen Namens, dies wird mit IPC erledigt). Empfängerbausteine können auch dann existieren, wenn kein zugeordneter Sender existiert. Der Ausgangsstatus des Konnektors ist dann zwar gültig, jedoch wird nur der Defaultwert ausgegeben. Darüber hinaus werden Konnektoren ohne Sender grau dargestellt.

OTCs sind Objekte und können mit den üblichen Methoden platziert, gelöscht und vergrößert/verkleinert werden.

Ein Off-Task-Konnektor erscheint im Layout dunkelgrau, wenn er weder als Ausgangsquelle defininiert wurde, noch mit einer Datenquelle verbunden ist. Sonst wird er hellgrau dargestellt.

3.9.4. Schalter und Schieberegler - Nützliche Testbausteine

Zur Online-Bedienung von binären Werten stehen sogenannte "Switches" = Schalter zur Verfügung, deren binärer Ausgangswert ein- oder ausgeschaltet werden kann. Bei einem linken Mausklick auf das Schaltersymbol verhält sich der Baustein wie ein Taster, bei einem rechten Mausklick wie ein Schalter. Analoge Werte können mittels "Slider" = Schieberegler vorgegeben werden, indem man den Schieberegler mit der Maus (gedrückte Maustaste) stufenlos zwischen dem Min- und Max-Wert verschiebt. (siehe Beispiel).

Bild: Beispiel für die Anwendung von Schaltern und Schiebereglern

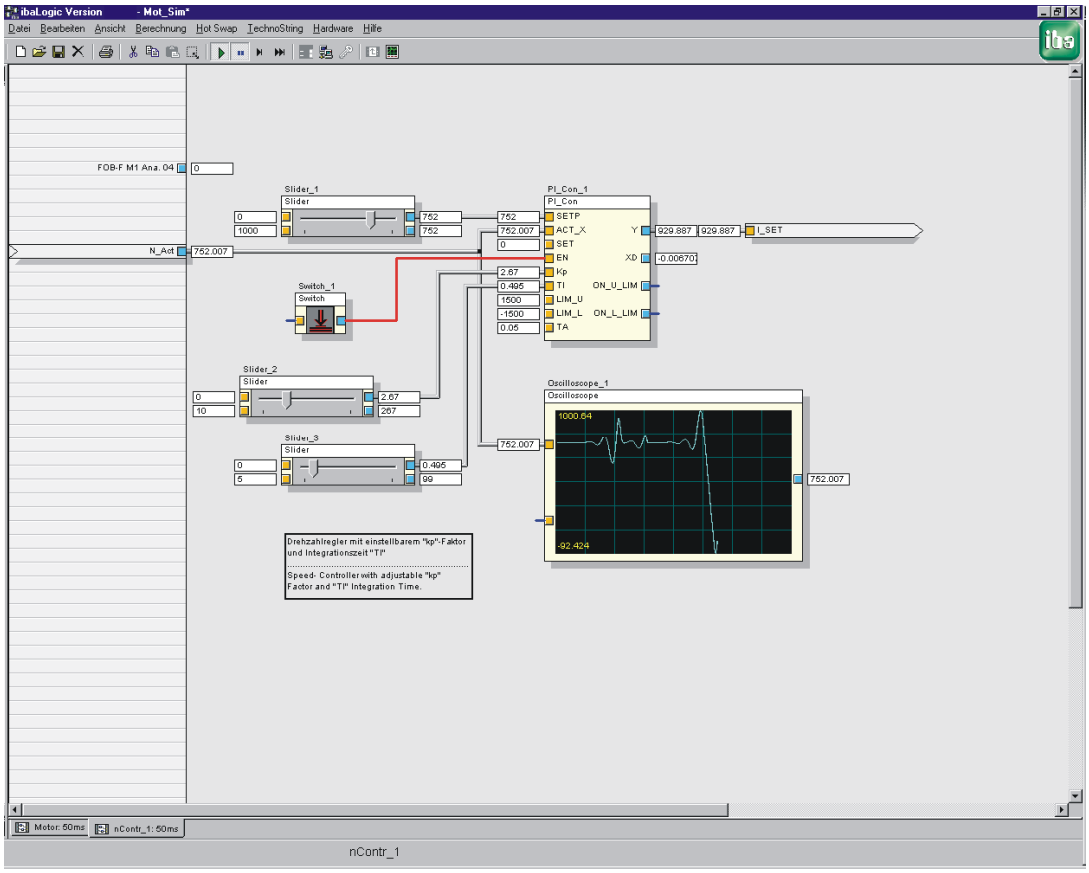
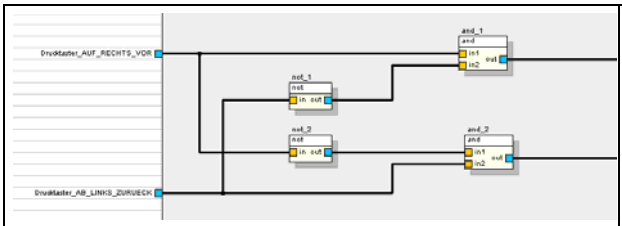
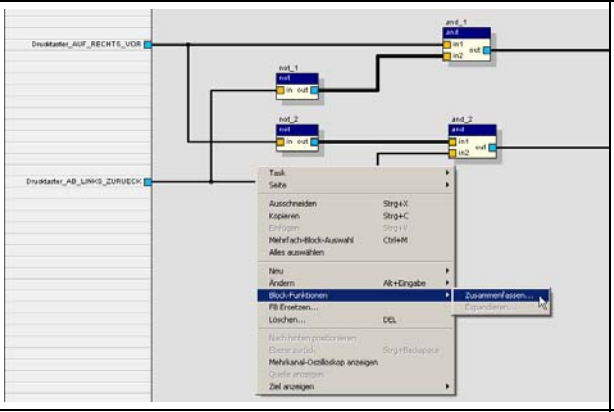
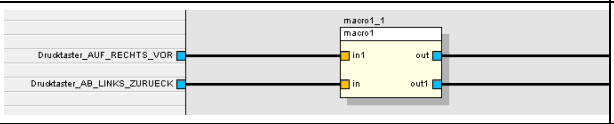
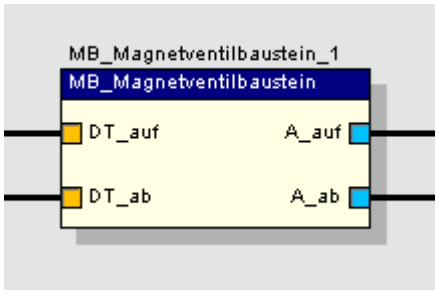


Bild 46 Beispielapplikation mit Schaltern und Schiebereglern

3.10 Zusammenfassen von Objekten und Erstellen von Makros

Eine der herausragenden Funktionen von ibaLogic besteht darin, dass mit wenigen Mausklicks mehrere Bausteine samt ihrer Verbindungslinien zu einem neuen Baustein zusammengefasst werden können. Dadurch wird das sogenannte Bottom-Up Design unterstützt. Diese Funktion ist sowohl dann hilfreich, wenn das Design beginnt unübersichtlich zu werden, als auch dann, wenn eine Schaltung mehrfach verwendet werden soll. Dabei kann eine beliebige Verschachtelungstiefe realisiert werden (Makro im Makro).

	Als Beispiel soll eine einfache Verriegelungslogik dienen, wie sie z.B. für Magnetventile verwendet wird. Um nicht für jedes Ventil immer wieder die vier Bausteine programmieren zu müssen, sollen diese zu einem Makroblock zusammengefasst werden. Um die betreffenden Bausteine zu kombinieren, müssen sie zunächst als Gruppe markiert werden. Dazu die Bausteine einzeln mit gedrückter <SHIFT>-Taste anklicken oder Mehrfach-Block-Auswahl betätigen.
	Sind die Funktionsblöcke und die Verbindungslinien markiert, dann mit <SHIFT>-Taste und rechten Mausklick das Bearbeiten-Menü aufrufen. Darin ↪*Block-Funktionen* ↪*Zusammenfassen* wählen und bestätigen.
	Ein neuer Makrobaustein ist entstanden. Doppelklick auf den Makroblock öffnet wieder die Funktionsplandarstellung der inneren Logik.
	Um den neuen Makroblock unabhängig von seinen originären Ein- und Ausgaben und universell verwendbar zu machen, sollten die Namen der Ein- und Ausgänge sowie der Bausteinname verändert werden. Um dies zu tun, muss man den Makroblock markieren und die Änderungsmaske des Bausteins aufrufen. ↪*Bearbeiten* ↪*Ändern...* ↪*Makroblock*

Analog dazu kann eine neue, zunächst leere Projekthierarchie mittels eines Makros erzeugt werden (Top-Down Designstrategie). Dazu ↪*Bearbeiten* ↪*Neu* ↪*Macroblock* anwählen. Die Bausteinmaske erscheint. Nun müssen die gewünschten Konnektoren dieser Ebene definiert werden. Innerhalb des Makros stehen dann auch nur diese Konnektoren als Ein- und Ausgänge zur Verfügung. Nachdem die Maske mit OK verlassen wurde, steht der Baustein zur Verfügung. Mit einem Doppelklick kann der (zunächst noch leere) Baustein geöffnet und editiert werden.

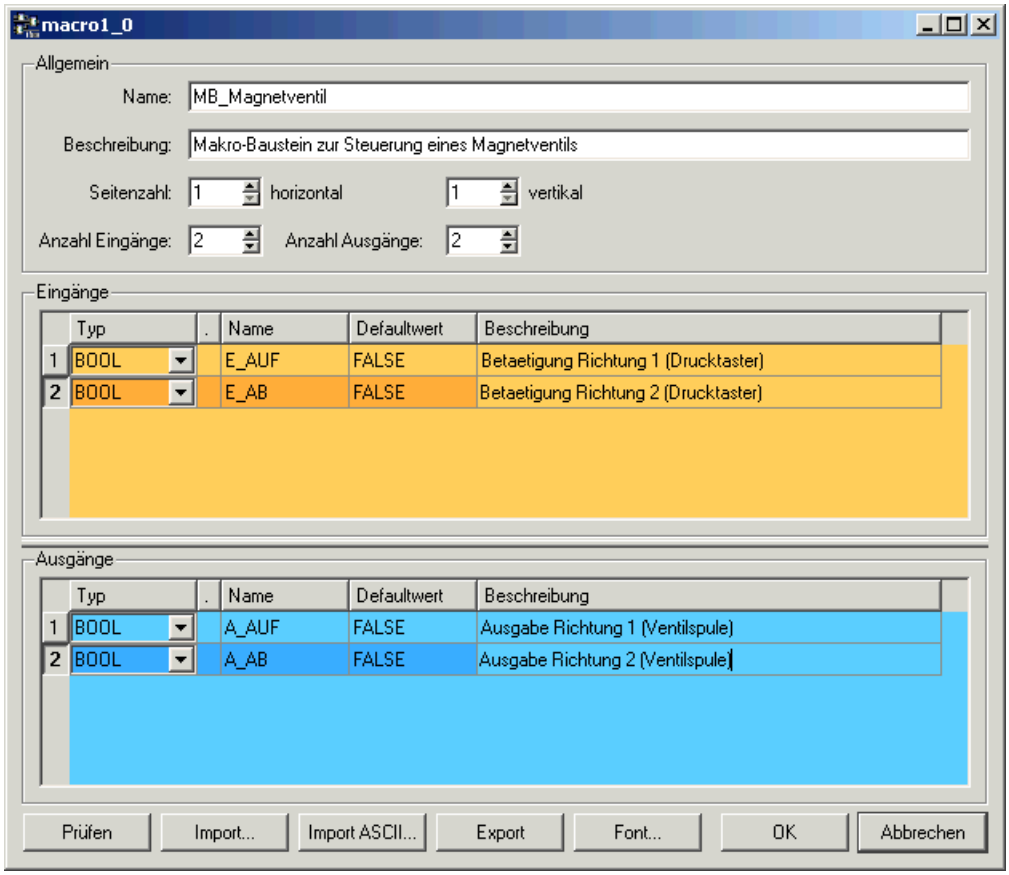


Bild 47 Erstellen eines Makro-Blocks

Makros können mit *Rechte Maustaste* ↪*Ebene zurück* oder *<Strg>+<Backspace>* wieder verlassen werden.

3.11 Erstellung eines neuen Funktionsbausteins

ibaLogic verfügt über eine große Bibliothek von fertig erstellten Funktionsbausteinen (siehe auch Kapitel 4). Trotzdem kann es natürlich vorkommen, dass der Anwender einen ganz neuen Baustein für seine spezielle Problemlösung benötigt. Zu diesem Zweck stellt ibaLogic zwei einfache Methoden zur Verfügung.

3.11.1. Erstellung eines neuen Funktionsbausteins ohne Structured Text

Aufgabenbeispiel: Der neue Funktionsbaustein soll sowohl die Differenz, als auch den Mittelwert zweier Eingangswerte berechnen und ausgeben.

Mit Hilfe des Bearbeiten-Menüs (↪*Bearbeiten* ↪*Neu* ↪*Funktionsbaustein*) wird die Eingabemaske für einen Funktionsbaustein geöffnet:

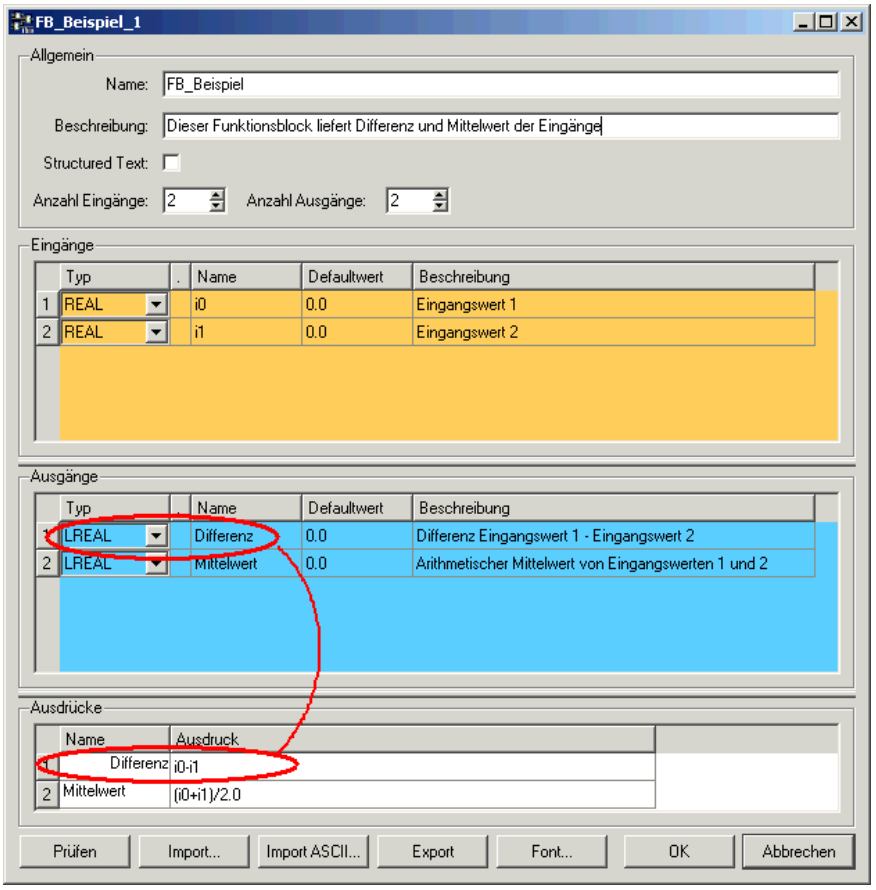


Bild 48 Erstellen eines neuen Funktionsbausteins

Zunächst ist die Anzahl der Eingänge und der Ausgänge auf jeweils 2 zu erhöhen. Anschließend sollte ein Bausteinname (fb_beispiel_1) und eine Beschreibung ("Dieser Baustein liefert...") in die entsprechenden Felder eingegeben werden. Mit jeder Erhöhung der Anzahl von Ein- bzw. Ausgängen werden in den gelben bzw. blauen Bereichen weitere Zeilen hinzugefügt. Reduziert man die Anzahl von Ein- oder Ausgängen, werden die Zeilen wieder gelöscht, nachdem man die Abfrage bestätigt hat.

Die Standard-Datentypen für die Ein- und Ausgänge sind LREAL, gem. der Bearbeitungseinstellungen im Menü ↪*Datei* ↪*Programmeinstellungen*.

In dem Bausteindialog klicken Sie bitte in die Felder der Spalte Typ bei den Eingängen und wählen Sie aus der Pick-Liste jeweils den Typ REAL. Die Signalnamen (i0, i1, o0, o1) können bei Bedarf auch umbenannt werden, so z.B. die Ausgaben in "Differenz" (o0) und "Mittelwert" (o1). Dazu in die entsprechenden Felder klicken. Es können im Prinzip beliebige Namen vergeben werden, jedoch sollten Sie darauf achten, keine vom System reservierten Namen zu verwenden. In solch einem Fall werden Sie jedoch mit einer Fehlermeldung darauf hingewiesen, und der Name wird nicht angenommen.

Des Weiteren können die Defaultwerte verändert werden. In unserem Beispiel macht dies keinen Sinn, es kann jedoch in anderen Fällen erforderlich sein.

Nun müssen die Funktionen des Bausteins programmiert werden.

Wenn nicht in ST programmiert wird, dann können lediglich einfache Zuweisungen und Operationen für jeden Ausgang eingetragen werden. Die Funktionen werden in den unteren, weißen Bereich eingetragen, wobei zuvor der entsprechende Ausgang angewählt werden muss. Dazu klickt man vorn in die Zeile des Ausgangs, so dass dort das Sternchen erscheint.

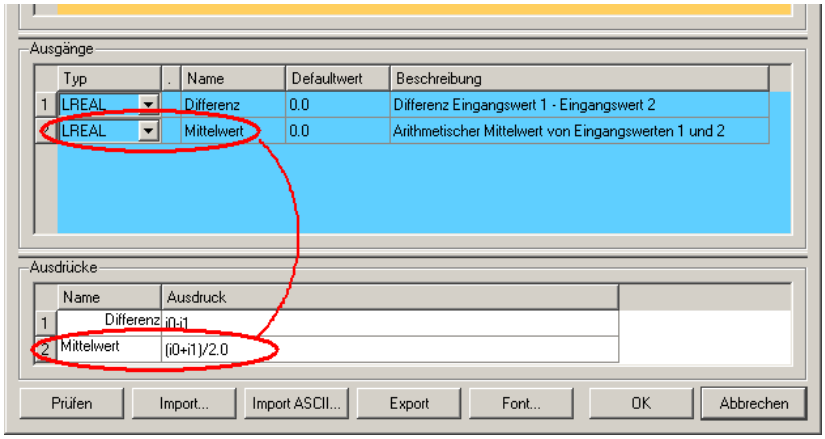


Bild 49 Erstellen eines Funktionsbausteins ohne ST

Alles, was dann in das weiße Feld eingetragen wird, bezieht sich auf diesen Ausgang.

Tabelle 1: Operatoren für einfache FB-Erstellung

Operator	Beispiel	Wert des Beispiels	Beschreibung	Priorität
()	(2+3) * (4+5)	45	Klammerung	höchste
**	3**4	81	Potenzierung	
-	-10	-10	Negation	
NOT	NOT DIG01	FALSE, wenn DIG01=TRUE	logische Negation	
*	10*3	30	Multiplikation	
/	6/2	3	Division	
+	2+3	5	Addition	
-	4-2	2	Subtraktion	
<, >, <=, >=	4 > 12	FALSE	Vergleich	
&, AND	TRUE & FALSE	FALSE	Boolesches UND	
XOR	TRUE XOR FALSE	TRUE	Boolesches Exklusiv Oder	
OR	TRUE OR FALSE	TRUE	Boolesches Oder	niedrigste

Tabelle 5 Operationen für einfache FB-Erstellung (ohne ST)

Die korrekte Programmierung kann mit der Schaltfläche "Prüfen" verifiziert werden.

Betätigen Sie nun die Schaltfläche "OK", um den neuen Baustein mit der Maus im Funktionsplan an der gewünschten Stelle zu platzieren.

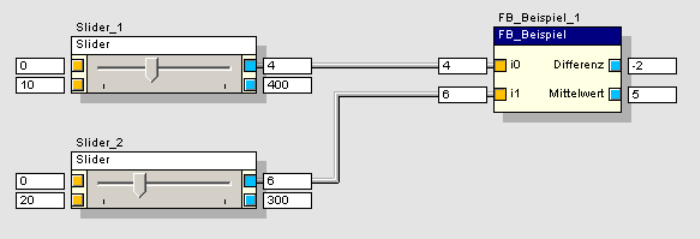


Bild 50 Platzieren des neuen FBs

Wenn Sie die Schaltfläche “Export” betätigen bevor Sie “OK” klicken, dann wird der Funktionsbaustein in dem Verzeichnis “Local FBs and Macros” bei den Funktionsressourcen abgelegt und steht dann wie ein normaler Standardbaustein für die weitere Benutzung mit Drag & Drop zur Verfügung.

*Physikalisch liegt der Funktionsbaustein als *.fbm-Datei in dem Verzeichnis \ibaLogic\configuration\FBs_Macros auf der Festplatte und kann somit auch gesichert oder kopiert werden.*

3.11.2. Erstellung eines neuen Funktionsbausteins mit Structured Text (ST)

Prinzipiell verfährt man genauso wie oben beschrieben, nur, dass die Option „Structured Text" in der Maske für die Funktionsbausteinerstellung aktiviert sein muss.

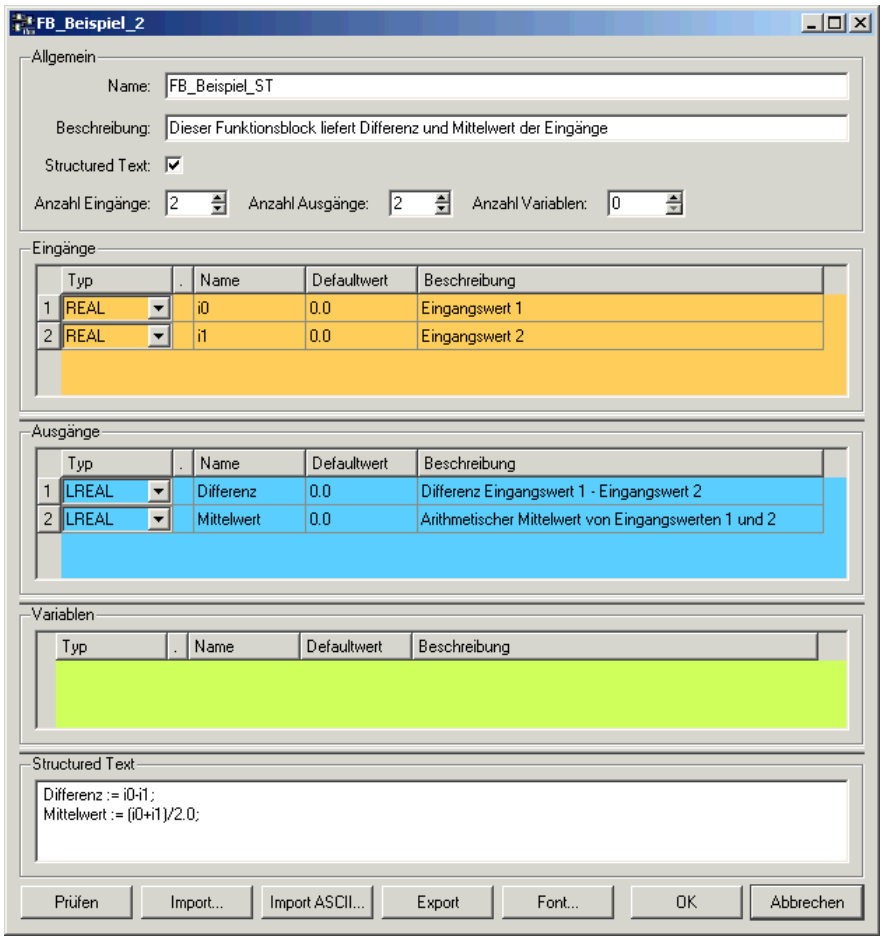


Bild 51 Erstellen eines Funktionsbausteins mit ST

Nun wird in dem weißen Bereich nicht mehr jeder Ausgang getrennt beschrieben, sondern es gibt nur noch einen Quellcode für den gesamten Baustein.

Zunächst sollen einige Grundbegriffe und Sprachelement von ST erläutert werden (siehe Literatur- und Quellenangaben, Anhang B, [1], [2])

3.11.2.1. Operatoren und Anweisungen in Structured Text (ST)

Programme in Structured Text haben große Ähnlichkeit mit Pascal-Programmen. Die Sprache benötigt ein Semikolon als Endezeichen für abgeschlossene Anweisungen. Kommentare werden mit "(*" eingeleitet und mit "*)" abgeschlossen. Ausdrücke und Anweisungen führen zur Verarbeitung von Daten. Ausdrücke bestehen aus Operatoren (siehe folgende Tabelle) und Operanden, sie liefern einen Ergebniswert. Als Operanden in einem Ausdruck sind zugelassen: Literale, Variablen, weitere Ausdrücke oder Funktionsaufrufe.

Tabelle 2: Operatoren in ST

Operator	Beispiel	Wert des Beispiels	Beschreibung	Priorität
()	(2+3) * (4+5)	45	Klammerung	höchste
**	3**4	81	Potenzierung	
-	-10	-10	Negation	
NOT		NOT TRUE	logische Negation	
*	10*3	30	Multiplikation	
/	6/2	3	Division	
MOD	MOD (17,10)	7	Modulo (Divisionsrest)	
+	2+3	5	Addition	
-	4-2	2	Subtraktion	
<, >, <=, >=	4 > 12	FALSE	Vergleich	
=	T#26h = T#1d2h	TRUE	Gleichheit	
<>	8 <> 16	TRUE	Ungleichheit	
&, AND	TRUE & FALSE	FALSE	Boolesches UND	
XOR	TRUE XOR FALSE	TRUE	Boolesches Exklusiv Oder	
OR	TRUE OR FALSE	TRUE	Boolesches Oder	niedrigste

Tabelle 6 Operationen in ST

3.11.2.2. Datendeklarationen bei Structured Text

Bei Structured Text-Anweisungen müssen die Datentypen UDINT und DWORD mit dem "#" Symbol gekennzeichnet werden (z.B. UDINT#0 oder DWORD#0), um sie von vorzeichenbehafteten Integer-Variablen zu unterscheiden.

Konstanten zur Basis 16 (Hexadezimal) werden mit "16#"gekennzeichnet (z.B. 16#2BC1F9) und automatisch als DWORD interpretiert. Konstanten zur Basis 2 werden mit "2#" und zur Basis 8 mit "8#" angegeben.

Zeiten werden mit "T#" gekennzeichnet und im Ausdruck mit dem Zusatz "d" (day), "h" (hour), "m" (minute), "s" (second) und "ms" (milli second) angegeben. Beispiel T#67d12h17m42s.

Datendeklarationen bei Structured Text (Textdarstellung der Deklaration):

```
VAR_INPUT
  in_bool:    BOOL   := FALSE;
  in_int:     INT    := 0;
  in_dint:    DINT   := 0;
  in_udint:   UDINT  := UDINT#0;
  in_dword:   DWORD  := DWORD#0;
  in_real:    REAL   := 0.0;
  in_lreal:   LREAL  := 0.0;
  in_time:    TIME   := T#0ms;
  in_string:  STRING := '';
END_VAR
```

```
VAR_OUTPUT
  out_bool:   BOOL   := FALSE;
  out_int:    INT    := 0;
  out_dint:   DINT   := 0;
  out_udint:  UDINT  := UDINT#0;
  out_dword:  DWORD  := DWORD#0;
  out_real:   REAL   := 0.0;
  out_lreal:  LREAL  := 0.0;
  out_time:   TIME   := T#0ms;
  out_string: STRING := '';
END_VAR
```

3.11.2.3. Anweisungen in Structured Text (ST)

Schlüsselwort	Beispiel	Beschreibung
RETURN	RETURN;	Rücksprunganweisung, vorzeitiger Abbruch des Funktionsbausteins
IF	IF a < b THEN c:=1; ELSIF a = b THEN c:=2; ELSE c:=3; END_IF;	Auswahlanweisung
CASE	CASE f OF 1: a:=3; 2: a:=4; ELSE a:=0; END_CASE;	Auswahlanweisung
FOR	FOR a:= 1 TO 10 BY 2 DO f[a] := b; END_FOR;	Wiederholungsanweisung (unbedingt)
WHILE Nicht unterstützt	WHILE b > 1 DO b := b/2; END_WHILE;	Wiederholungsanweisung (bedingt) Nicht unterstützt, wg. Gefahr von Dauer- schleifen
REPEAT Nicht unterstützt	REPEAT a:= a * b; UNTIL a > 10000 END_REPEAT;	Wiederholungsanweisung Nicht unterstützt, wg. Gefahr von Dauer- schleifen
SET_VALID	SET_VALID (<Variablenname>, FALSE)	Gültig- / Ungültig-Setzen einer Variable (z.B. FB-Anschluss)
SET_DEFAULT	SET_DEFAULT (<Variablenname>, <Wert>)	Defaultwert einer Variable setzen
ARRAY-Zugriff	<Variablenname>[i] <Variablenname>[i,j,k,m]	Zugriff auf eindimensionales Array Zugriff auf vierdimensionales Array
EXIT	EXIT;	Vorzeitiger Abbruch, z.B. von FOR-Schleifen

Tabelle 7 Anweisungen in ST

Bitte beachten Sie, dass FBs nicht in ST-Anweisungen verwendet werden können. Nur Funktionen sind erlaubt.

Außerdem bestehen hinsichtlich der Verwendung von FB-Namen oder Funktionen, die für ibaLogic reserviert sind, einige Einschränkungen.

3.11.2.4. Funktionsbaustein PT1 mit Structured Text (ST)

Die Erstellung eines neuen Funktionsbausteins mit "ST" soll anhand eines Verzögerungsglieds erster Ordnung (PT1) erläutert werden. Die Mathematische Beschreibung des PT1-Glied lautet:

$$Y = Y_{n-1} * e^{-(TA/T1)} + X1_{n-1}(1 - e^{-(TA/T1)})$$

mit:

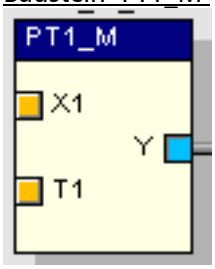
Y = Ausgangswert PT1-Glied
Y_{n-1} = Ausgangswert aus vorhergehendem Programmzyklus
X1 = Eingangswert
$X1_{n-1}$ = Eingangswert aus vorhergehendem Programmzyklus
T1 = Verzögerungszeit [s], Ausgangswert auf 63% des Eingangswerts
TA = Abtastzeit [sec]

Bei einer Vielzahl regelungstechnischer Bausteine wird die Task-Abtastzeit sowie die Zeit seit Beginn der Anwendung benötigt. Diese beiden Zeiten stehen als globale Variable in Structured Text zur Verfügung:

g_EvalDeltaTime = Zeit, seit dem letzten Taskstart (Abtastzeit); bei Verwendung dieser Variable werden ± Abweichungen der Abtastzeit korrigiert.

g_EvalTime = Zeit, seit Start der Anwendung

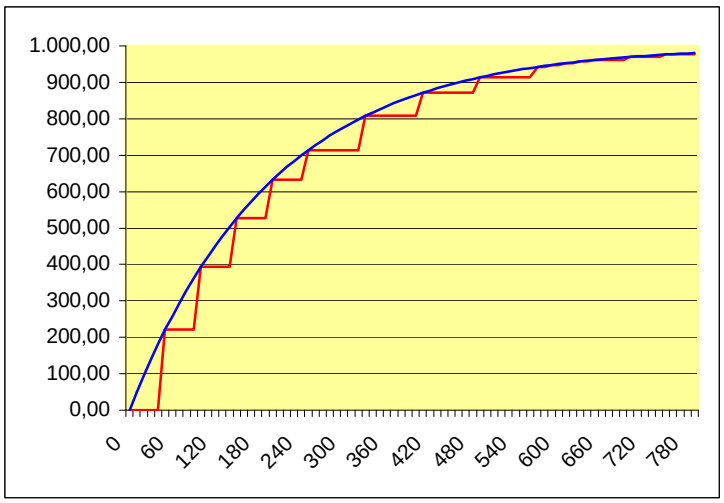
Baustein "PT1_M"



Programmcode "PT1_M" in Structured Text

```
TA_T1:=-time_to_lreal(g_EvalDeltaTime)/T1; (* Get Cy-
cletime TA and evaluate –TA/T1 *)
E_TA_T1:=2.71828**TA_T1; ; (* Evaluate e** TA/T1*)
Y:=Y * E_TA_T1+(X_N1*(1.0-E_TA_T1)); (* Y- calculation *)
X_N1:=X1; (* Copy X1(n-1) = X1 *)
```

Sprungantwort PT1-Baustein
blau: Abtastzeit konst. 10ms
rot: Abtastzeit konst. 50 ms



Die Erstellung eines neuen Funktionsbausteins beginnt mit dem Kommando ↪*Bearbeiten* ↪*Neu* ↪*Funktionsblock*. Daraufhin wird die folgende Bildschirmmaske mit fünf Eingabebereichen angezeigt.

❑ ***Allgemein***

Festlegung der Anzahl Ein- und Ausgänge, Bausteinname und Beschreibung

❑ ***Eingänge***

Definition der Eingangsvariablen mit Datentyp und Beschreibung

❑ ***Ausgänge***

Definition der Ausgangsvariablen mit Datentyp und Beschreibung

❑ ***Variablen***

Festlegung der internen Bausteinvariablen mit Typ und Beschreibung

❑ ***Definition***

Structured Text (ST) Programmanweisungen

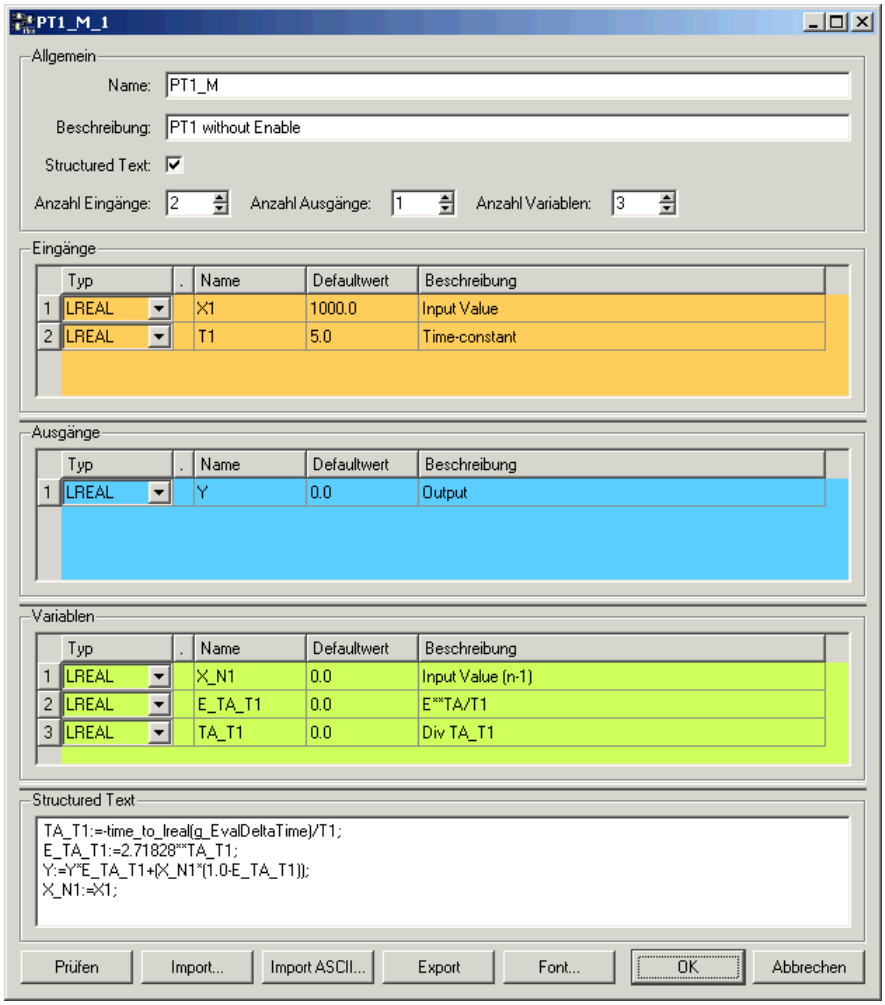


Bild 52 Erstellen von Funktionsbaustein PT1M

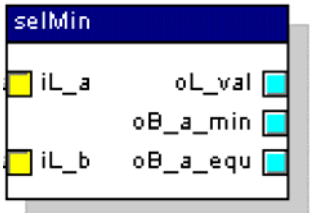
3.11.3. Anwendungsbeispiele Structured Text-Anweisungen

In den folgenden Beispielen sollen die wesentlichen Anweisungen von Structured Text (if, case, for, usw.) anhand einfacher Funktionsbausteine beschrieben werden.

3.11.3.1. IF- und ELSIF-Anweisung

Es soll der Baustein "selMin" erstellt werden, von dessen beiden Eingängen "a" und "b" immer der mit dem kleinsten Wert an den Ausgang "val" ausgegeben wird. Wenn der Eingang "a" kleiner oder gleich "b" ist, wird der boolsche Ausgang "a_min" auf "TRUE" gesetzt. Haben die Eingänge "a" und "b" den gleichen Wert, wird der L-Real-Ausgang "val" auf 0.0 und der boolsche Ausgang "a_equ" auf "TRUE" gesetzt.

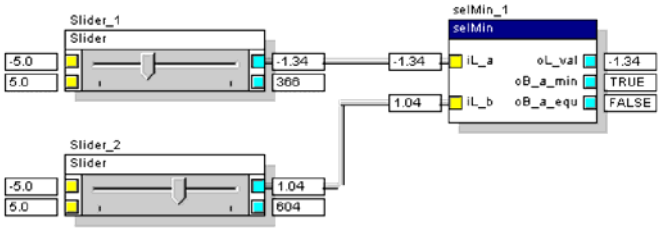
Baustein "selMin"



Programmcode "selMin" in Structured Text

```
oB_a_equ:=FALSE;      (* Default setting *)
if iL_a=iL_b
  then oL_val:=0.0;   (* a=b, Output=0.0 und equ=TRUE *)
  oB_a_min:=TRUE;
  oB_a_equ:=TRUE;
  elsif iL_a<iL_b        (* Check a<b *)
    then oL_val:= iL_a; (* a is smaller, Output to val *)
    oB_a_min:=TRUE;     (* a_min = TRUE *)
    else oL_val:= iL_b; (* b is smaller, Output to val *)
    oB_a_min:=FALSE;    (* a_min = FALSE *)
  end_if;
```

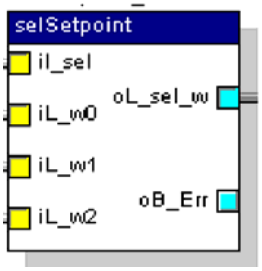
Anwendungsbeispiel "selMin"



3.11.3.2. CASE-Anweisung

Es soll der Baustein "selSetpoint" erstellt werden, von dessen drei LReal-Eingängen "w0", "w1" und "w2" einer über den Inhalt des INT-Eingangs "sel" auf den Ausgang "sel_w" geschaltet werden kann. Wenn der Eingang "sel" ungleich {0,1,2} ist, wird der Ausgang "sel_w" auf 0.0 und der boolsche Ausgang "Err" auf "TRUE" gesetzt.

Baustein "selSetpoint"



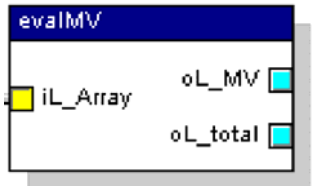
Programmcode "selSetpoint" in Structured Text

```
oB_Err := FALSE;          (* Default setting *)
CASE iI_sel OF            (* CASE- selection 0,1,2 *)
0: oL_sel_w:=iL_w0;       (* CASE = 0 *)
1: oL_sel_w:=iL_w1;       (* CASE = 1 *)
2: oL_sel_w:=iL_w2;       (* CASE = 2 *)
ELSE oL_sel_w:=0.0;       (* value iI_sel unequal 0,1,2 *)
oB_Err:= TRUE;            (* sel_w = 0.0, Err = TRUE *)
END_CASE;
```

3.11.3.3. FOR-Anweisung

Es soll der Baustein "evalMV" erstellt werden, der die Summe und den Mittelwert eines Arrays aus 16 LReal-Variablen berechnet. Der Baustein benutzt die interne Variable "count" als Zählvariable.

Baustein "evalMV"



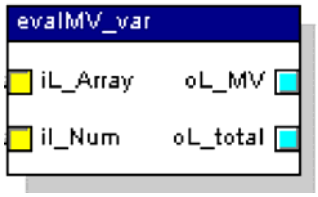
Programmcode "evalMV" in Structured Text

```
oL_total:=0.0;                        (* Default setting *)
FOR count:=0 TO 15 DO                    (* FOR- 0 to 15 *)
  oL_total:= oL_total+iL_Array[count]; (* total-value *)
END_FOR;
oL_MV:=oL_total/16.0;          (* Mean Value evaluation *)
```

3.11.3.4. EXIT- und RETURN-Anweisung

Der bereits erstellte Baustein "evalMV" wird in "evalMV_var" umbenannt und um den INT-Eingang "iI_Num" ergänzt. Dieser Eingang legt fest, bis zu welchem Element des Arrays die Summen- und Mittelwertbildung stattfinden soll (z.B. 7 = Summe und den Mittelwert der Elemente 0...7 des Arrays). Mit einer "Exit"-Anweisung innerhalb der IF-Abfrage wird die FOR-Schleife vorzeitig beendet, die Mittelwertbildung jedoch noch durchgeführt. Mit einer "RETURN"-Anweisung an gleicher Stelle wird der Funktionsbaustein sofort beendet, ohne die Mittelwertbildung zu berechnen.

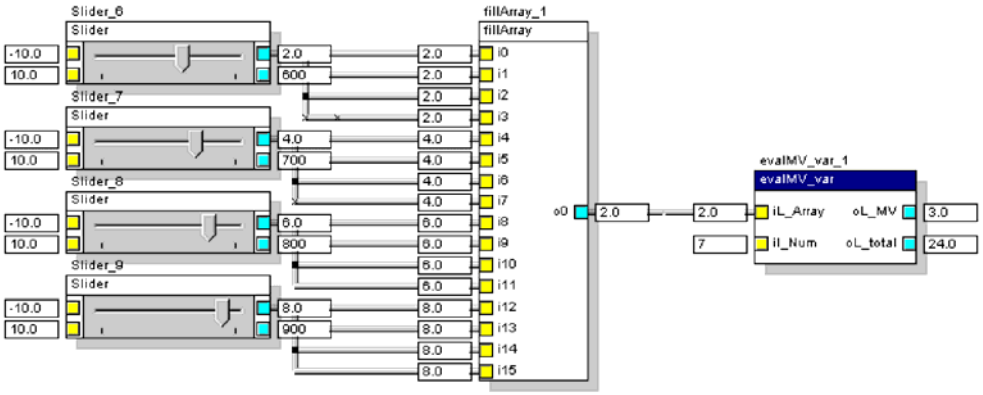
Baustein "evalMV_var"



Programmcode "evalMV_1" in Structured Text

```
oL_total:=0.0;                        (* Default setting *)
FOR count:=0 TO 15 DO                    (* FOR- 0 to 15 *)
  oL_total:= oL_total + iL_Array[count]; (* total-value*)
    IF count>(iI_Num-1)   (* max. Number Input reached *)
    THEN EXIT;     (* or RETURN; FOR- Loop termination *)
    END_IF;
END_FOR;
oL_MV:=oL_total/int_to_real(iI_Num+1);  (* Mean Value *)
```

Anwendungsbeispiel "evalMV_var"



3.12 Erstellen eigener DLLs

Eigene Makros und Funktionsbausteine mit ST zu erstellen ist eine sehr einfache Möglichkeit viele Aufgaben in der Automatisierungstechnik zu lösen. Aber so einfach es ist, die Makros und Funktionsblöcke zu erstellen, so einfach ist es auch, sie zu kopieren und ihren Inhalt, d.h. ihre Funktion zu verstehen.

Manchmal ist es jedoch wünschenswert, die eigene technologische Kompetenz weniger offen zu legen und stattdessen eher, z.B. im Falle einer sehr intelligenten prozesstechnischen Lösung, der weiteren unkontrollierten Verbreitung des eigenen technologischen Know-Hows vorzubeugen.

In solch einem Fall ist die Möglichkeit der Erstellung eigener DLLs, die das Wissen nur in compilierter Form enthalten und somit nicht einfach ausgelesen werden können, von Vorteil.

3.12.1. C – Compiler

Für das Schreiben und Compilieren der DLLs haben wir die folgenden Compiler getestet:

- ❑ Microsoft Visual C++ 5.0
- ❑ Microsoft Visual C++ 6.0
- ❑ Andere, z.B. Borland, sind auch möglich.

3.12.2. Benötigte Quelldateien

Die folgenden Quelldateien, welche mit der ibaLogic Installations-CD geliefert werden, sind für die DLL-Erstellung erforderlich:

- *name*dll.cpp: enthält die Prozeduren und den DLL-Body;
  der Anwender kann Ein- oder Ausgaben hinzufügen oder Änderungen der Prozeduren InitEvaluation, Evaluate und ExitEvaluation vornehmen
- *name*dll.def: enthält die Zuordnung von DLL-Prozeduren und -Numbers;
  der Name der Library muss dem Namen der DLL entsprechen!
- dllForm.hpp: enthält die Schnittstellendefinition;
  hier sind keine Veränderungen erforderlich.

↗ *Siehe dazu auch Kapitel 7.1. Dort sind die Programmlistings der mitgelieferten „sampleDLL“ abgedruckt.*

3.12.3. Vorgehen zum Erstellen neuer DLL’s

Für das Erstellen neuer DLLs empfehlen wir die Verwendung des einfachen "simpleDLL"-Frames:

1. Erstellen Sie ein neues DLL-Projekt mit Ihrem eigenen DLL-Namen.
2. Kopieren Sie die Dateien simpleDLL.cpp, simpleDLL.def und DLLform.hpp in Ihr Projekt-Verzeichnis.
3. Benennen Sie die Dateien simpleDLL.cpp und simpleDLL.def gemäß Ihrer eigenen Namensgebung um.
4. Ändern Sie den Namen der Library in der Datei ...DLL.def in Ihren eigenen DLL-Namen um.
5. Fügen Sie die Dateien .cpp, .def und .hpp zu Ihrem Projekt hinzu.
6. Erzeugen Sie die neue DLL.
7. Kopieren Sie die neue DLL in Ihr ibaLogic-Verzeichnis.

3.12.4. Hindernisse und Hinweise

Bei der Verwendung von DLLs sollten Sie folgendes beachten:

- ❑ Vergessen Sie nicht, die ".def"-Datei Ihrem Projekt hinzuzufügen, da ibaLogic sonst nicht mit der DLL arbeitet.
- ❑ Sie können Variablen zur DLL oder Evaluate-Prozedur hinzufügen und benutzen. Wenn aber mehr als eine Instanz der DLL verwendet wird, dann müssen die Daten zwischen zwei Aufrufen im dynamischen Datenbereich zwischengespeichert werden. Ansonsten existiert jeweils nur eine Variable für alle Instanzen.
- ❑ Für den Fall, dass Sie zykluszeitabhängige Funktionen realisieren wollen, sollten Sie die Variable "pGlobal" verwenden, die ein Zeiger auf eine Datenstruktur ist, welche die relative Zeitvariable enthält.
- ❑ Die DLL-Laufzeit verlängert die Laufzeit der Task in der die DLL aufgerufen wird.
- ❑ Es wird empfohlen, zeitaufwändige Funktionen in Threads auszulagern.
- ❑ Bausteine sollten stets das Invalid-Flag benutzen. Außerdem sollten Bausteine nur Daten nach außen zur Peripherie schreiben, wenn das Online-Flag gesetzt ist.
- ❑ Zum Testen der DLL kann ibaLogic als ausführendes Programm gestartet werden.
- ❑ Die Schnittstellenfunktionen einer DLL werden von ibaLogic direkt aufgerufen.
- ❑ Von ibaLogic können nicht alle Programmierfehler in einer DLL erkannt und abgefangen werden, so dass derartige Fehler durchaus auch den Absturz von ibaLogic verursachen können.

3.12.5. Einbindung der DLL in ibaLogic

ibaLogic benötigt folgende Aufrufe, um das Interface des Bausteins abzufragen:

Aufruf	Funktion
GetInstanceDynamicDataSize()	Abfragefunktion zur Bestimmung der Größe der Dynamischen Daten
GetDllDescription()	Abfragefunktion für Beschreibung der DLL
GetCount()	Abfragefunktion für Anzahl der Ein- und Ausgänge
GetName()	Abfragefunktion für den Namen jedes Ein- und Ausgangs
GetDescription()	Abfragefunktion für die Beschreibung jedes Ein- und Ausgangs
GetType()	Abfragefunktion für den Datentyp jedes Ein- und Ausgangs
GetArrayHeader()	Abfragefunktion für Datentyp Array eines Ein- oder Ausgangs
GetDefaultValue()	Abfragefunktion für den Defaultwert eines Ein- oder Ausgangs

Folgende Aufrufe werden zur Laufzeit benötigt:

Aufruf	Funktion
InitEvaluation()	Einmaliger Aufruf zu Beginn der Evaluierung, dient zur Initialisierung.
SetInputValue()	Zyklischer Aufruf, wird für jeden Eingang einmal pro Durchlauf aufgerufen. Die Bearbeitung erfolgt vor jeder Berechnung.
Evaluate()	Zyklischer Aufruf, wird einmal pro Durchlauf aufgerufen.
GetOutputValue()	Zyklischer Aufruf, wird für jeden Ausgang einmal pro Durchlauf aufgerufen. Die Bearbeitung erfolgt nach jeder Berechnung.
Exit Evaluation()	Einmaliger Aufruf am Ende der Evaluierung, dient zum Aufräumen.

3.13 Projekte Testen und Debuggen

ibaLogic verfügt über einige eingebaute Testhilfen, mit deren Hilfe sich Bausteine und komplexe Schaltungen einfach testen lassen.

3.13.1. Einzel- und Mehrfachschritt-Modus, Projekt anhalten

Wird ein Projekt evaluiert (gerechnet), dann kann das Projekt in einen Einzel- oder Mehrfachschrittmodus versetzt werden. Dies ist z.B. immer dann hilfreich, wenn rein logische Schaltungen getestet werden sollen. Dazu dienen die Schaltflächen in der Symbolleiste  .
(v.l.n.r.: Start-Stop / Pause Berechnung / Einzelschritt / Mehrfachschritt).

Zunächst muss der Pause-Knopf gedrückt werden. Die beiden Knöpfe Einzel- und Mehrfachschritt werden damit aktiv und können nun benutzt werden.

Achtung bei Einzel- und Mehrfachschrittmodus: Wenn das Projekt Online geschaltet ist (Hintergrund ist rosa), findet für die Dauer zwischen den Einzelschritten kein Update der externen Ressourcen (I/Os) statt, d.h. die Werte bleiben unverändert gleich!

Dies kann Gefahr für Mensch und Maschine bedeuten!

Die Anzahl der gewünschten Schritte für Mehrfachschritt kann mit ↪*Berechnung* ↪*Mehrfachschritt einstellen* ↪*...* zwischen 2 und 64 Schritten eingestellt werden.

3.13.2. Was tun, wenn Werte sporadisch ungültig werden?

Bei Berechnungen kann es vorkommen, dass durch ungünstige Projektierung (z.B. ungünstige Startbedingung, Division durch Null) Ausgangswerte von Bausteinen ungültig (invalid) werden. ibaLogic markiert dies dadurch, dass dieser Signalausgang durchkreuzt wird, einen roten Rahmen erhält und zusätzlich dann, wenn die Evaluierung eingeschaltet ist, der Berechnungswert ebenfalls rot dargestellt wird.

Wann kann eine Variable ungültig (invalid) werden?

- ❑ Ungültiger Realwert
- ❑ Division durch Null
- ❑ Gezieltes Setzen des Invalid Bits durch set_valid(<variablenname>, FALSE);
- ❑ Bei der Zuweisung von Arrayelementen, wenn deren Indexgrenzen überschritten sind.
- ❑ Durch Zuweisung von Ausdrücken, in denen bereits ungültige Werte enthalten sind.
- ❑ Wenn sie Bestandteil einer Verkettung (Schleife) von invalid Variablen ist.
- ❑ Eingangsressourcen, wenn die zugehörige PC-Steckkarte (FOB IO, FOB 4i) fehlt oder nicht erreichbar ist und wenn in den Systemeinstellungen die Option „*Nicht verfügbare Signale sind invalid*“ gewählt wurde.

1. Besonderheiten: Arrays haben genau ein Valid-Flag. Ist ein Element des Arrays ungültig, so gilt das für das gesamte Array.
2. Wird eine Variable ungültig, so bleibt der letzte gültige Werte erhalten
3. Kommt es innerhalb von grafischen Rückkopplungen zu einer invalid Variablen, dann kann die Situation korrigiert werden:
   - mit Start/Stop Evaluierung oder
   - durch aufbrechen der Logik (Löschen einer Linie, Einfügen eines Bausteins usw.) oder
   - durch gezieltes Einfügen einer set_valid- Anweisung, set_valid(<variablenname>, TRUE), in die Logik; damit kann ein Plan so projektiert werden, dass sich dieser bei einem eventuellen Auftreten dieser Situation selbst „reparieren“ kann.

Fall 1 und 2 setzen natürlich voraus, dass die Variable nicht immer invalid bleibt.

3.13.3. Das einfache Testoszilloskop

Zum schnellen Überprüfen einer Signalform dient das Oszilloskop. Es wird genau wie ein Standardbaustein platziert und zeigt unmittelbar den Signalverlauf des angeschlossenen Signals an. Der Signaleingang ist vom Typ <untyped> . Arrays können nicht angeschlossen werden (siehe nächstes Kapitel). Das Oszilloskop besitzt keine Achsen zum Vermessen eines Signals.

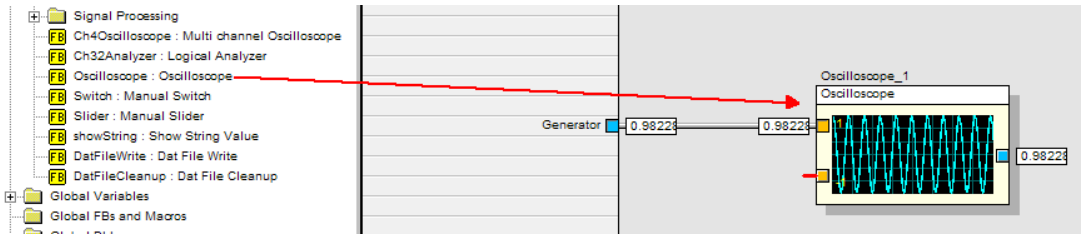
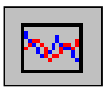


Bild 53 Einfaches Oszilloskop

3.13.4. Mehrkanal-Oszilloskop und Logik-Analysator



Diese beiden Bausteine sind vom Prinzip her identisch, arbeiten nach der gleichen Grundphilosophie, haben aber unterschiedliche grafische Einsatzfälle.

- ❑ Der Logik-Analysator (Ch32Analyzer) dient zum Test von bis zu 32 gleichzeitig angeschlossenen Binärsignalen (kennt nur den Datentyp BOOL am Eingang)
- ❑ Das Mehrkanal-Oszilloskop (Ch4Oscilloscope) dient zur Vermessung von bis zu max. 4 Signalen, zur Regleroptimierung und zur *Darstellung von Arrays* (Vektoren z.B. FFT Ergebnis).

3.13.4.1. Verwendung

Im Funktionsplan wird einer der beiden Bausteine platziert. Er verhält sich zunächst wie ein Tastkopf ohne Anzeige, d.h. die Signale sind lediglich angeschlossen. Da diese beiden Bausteine keine direkte Grafikfunktion ausführen, benötigen sie im Gegensatz zum einfachen Oszilloskop keine Rechenzeit, wenn das Display nicht aktiviert ist. Das ist von großem Vorteil – können doch auf diese Weise eine Vielzahl von Tastköpfen fest „verdrahtet“ und nur bei Bedarf aktiviert werden. Die Tastköpfe können jeweils nur einzeln aktiviert werden, d.h., dass nur eine Displayinstanz des 4-Kanal-Oszilloskops (und des LogikAnalysators) existieren kann. Zur Umschaltung zwischen verschiedenen Tastköpfen genügt es, am unteren Rand im Anzeigefenster den jeweiligen Tastkopf zu aktivieren (Die Umschaltleiste zeigt jeweils den Instanznamen des Tastkopfes an).

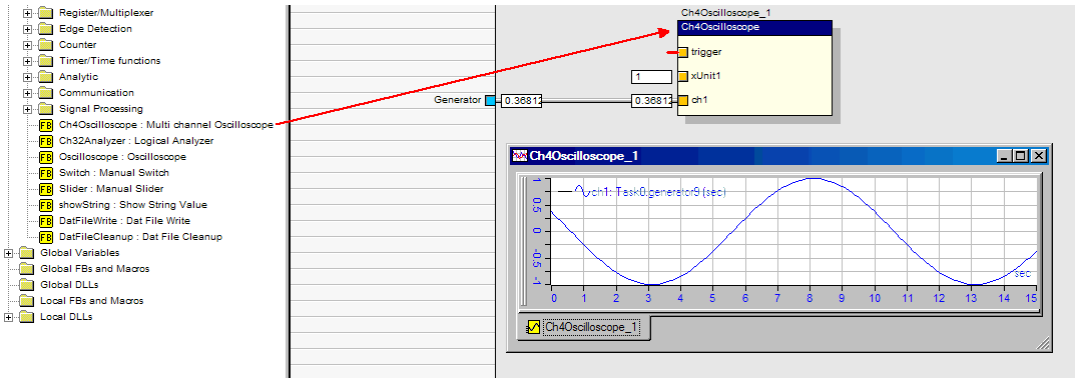


Bild 54 Mehrkanal-Oszilloskop

Die Anzahl der dem Tastkopf zur Verfügung stehenden Eingänge wird durch Doppelklicken auf den Baustein und über die Anzahl der Eingänge eingestellt.

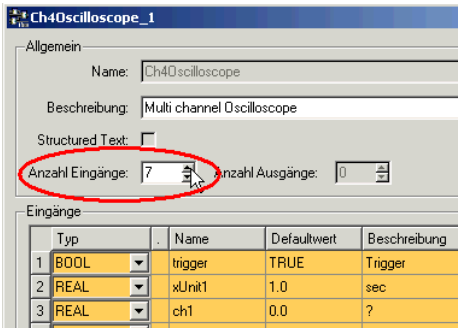


Bild 55 Einstellen der Anzahl der Eingangskanäle für das Mehrkanal-Oszilloskop

Beim Logik-Analysator entsteht mit jedem Hinzufügen ein neuer Eingang (Bool).

Beim Mehrkanal-Oszilloskop entsteht jeweils ein Skaleneingang (*xUnit*) zusammen mit dem Signaleingang (*ch).* Damit läßt sich jeder Signalstreifen individuell skalieren. Ist ein Skaleneingang nicht angeschlossen, so erhält er automatisch die Skala des vorhergehenden Eingangs.

Solange der Triggereingang = TRUE ist, wird die Anzeige aktualisiert. FALSE hält die Anzeige an (Freeze).

3.13.4.2. Skalieren der Eingänge

Defaultwert eines jeden Skaleneinganges ist „1“. Jeder Wert (z.B. skalarer Realwert oder ein Element eines Arrays) wird mit dieser hier eingestellten Größe skaliert. Für Arraygrößen kann dies – sofern Unabhängigkeit von der veränderbaren Basiszykluszeit erwünscht ist - etwas komplexer werden.

↗ *Siehe dazu auch Kasten* **Dynamisch skalieren**, *S. 3-43.*

3.13.4.3. Bedienung

Die Aktivierung der Anzeige erfolgt mit *Selektieren Baustein/ rechte Maustaste / ↪Mehrkanal-Oszilloskop anzeigen* oder mit dem Button  in der Symbolleiste. Mit dem gleichen Kommando wird auch der Logik-Analysator aktiviert.

Die Bedienung des Oszilloskops ist mit der Version 3.86 von ibaLogic noch einmal überarbeitet und verbessert worden. Das neue Bedienkonzept entspricht dem, wie es viele Anwender bereits vom ibaAnalyzer her kennen.

Unterschiedliche Einfärbung der Kurven, stufenloses Stauchen und Strecken der X- und Y-Achsen sowie das Verschieben von Signalen bzw. das Kombinieren meh-

rerer Signale in einem Signalstreifen und schließlich eine Vermessung der Werte über Lineale sind nun auch hier möglich.

Über Kontextmenüs in den Bereichen X-Achse, Y-Achse, Diagramm und Signalname werden entsprechende Einstellungen angeboten.

Mit gedrückter linker Maustaste lässt sich im Signalstreifen ein rechteckiger Bereich aufziehen, der dann gezoomt wird. Mit dem Befehl *Autoskalierung* im Kontextmenü des Signalstreifens gelangt man aus der gezoomten Darstellung zurück in die Gesamtdarstellung der Kurve. Das Kontextmenü wird mit einem rechten Mausklick in den Signalstreifen des Oszilloskopfensters geöffnet. Hier bieten sich verschiedene Einstellmöglichkeiten für die Darstellung der Werte.

Über die Kontextmenüs von X- und Y-Achse ist auch stufenweises Ein- und Auszoomen möglich.

X- und Y-Achse können mit gedrückter Maustaste verschoben werden (Nullpunktverschiebung).

Die Funktion der Autoskalierung ist außerdem hilfreich, wenn die Signale nicht zu sehen sind bzw. über das Diagrammfenster hinaus gehen.

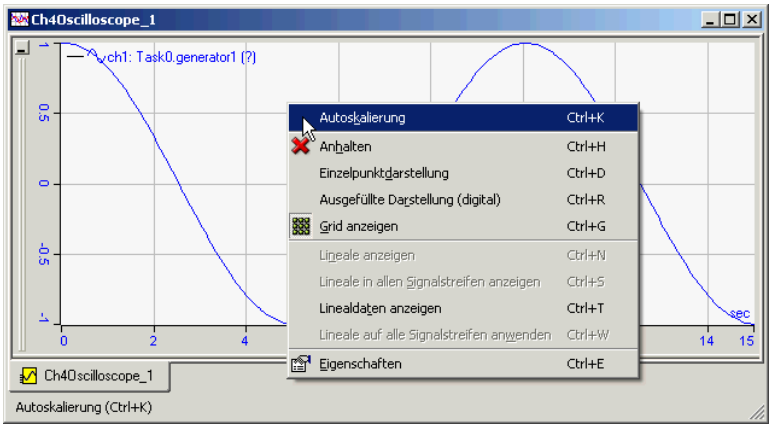


Bild 56 Autoskalierung im Mehrkanal-Oszilloskop

Zur einfachen Vermessung von Werten lassen sich über das Kontextmenü im Signalstreifen zwei Lineale in den aktuellen Signalstreifen einblenden. Dazu muss vorher die Aktualisierung der Kurvendarstellung angehalten werden.

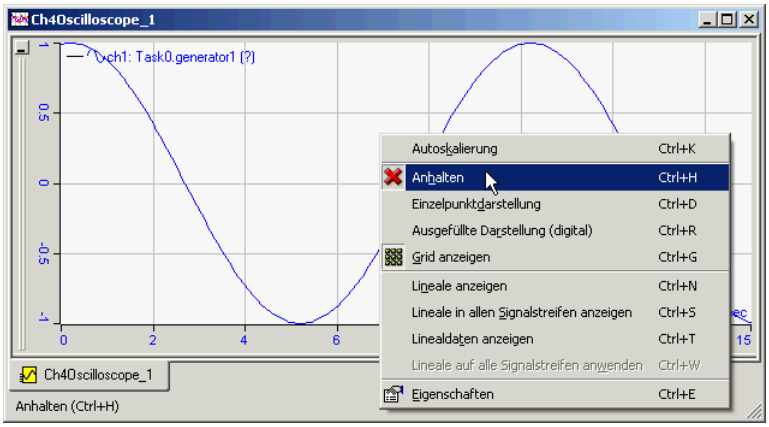


Bild 57 Einfrieren der Anzeige im Mehrkanal-Oszilloskop

Um die Werte ablesen zu können, muss zusätzlich der Menüpunkt ↪*Linealdaten anzeigen* ausgewählt werden. Es wird dann unterhalb der Signalstreifen eine Tabelle mit den X-Positionen der Lineale, den zugehörigen Y-Werten sowie den Differenzen angezeigt.

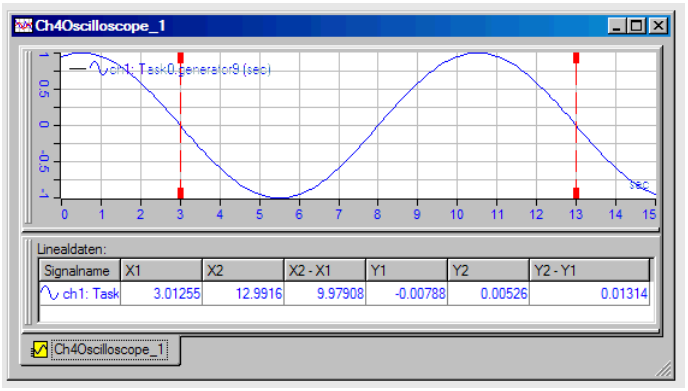


Bild 58 Markierer und Datentabelle im Mehrkanal-Oszilloskop

Wenn mehrere Kanäle des Mehrkanaloszilloskops genutzt werden, erscheint je Kanal ein Signalstreifen. Jeder Signalstreifen hat seine individuelle X-Achse und –Skalierung (entsprechend Eingang xUnit).

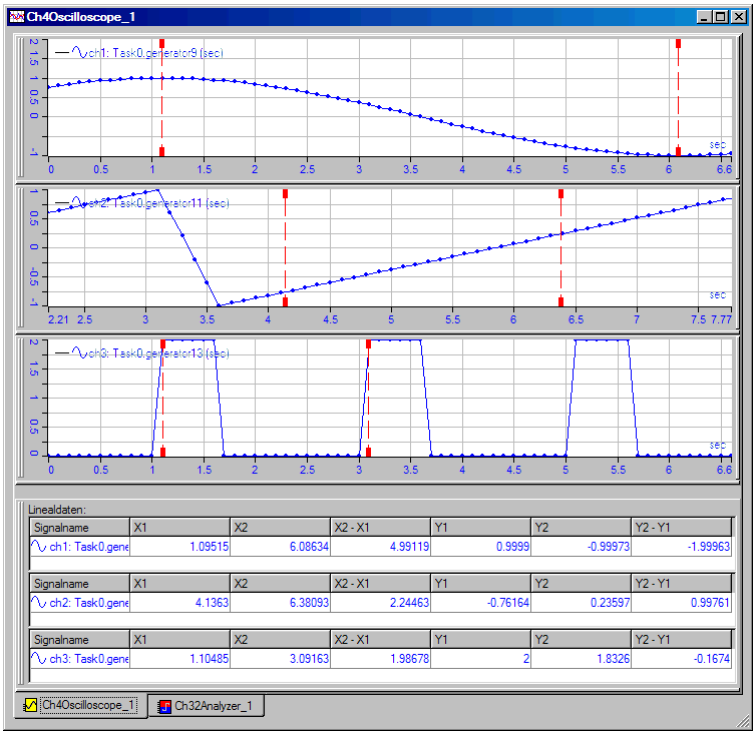


Bild 59 Darstellung mehrerer Kanäle im Mehrkanal-Oszilloskop

Mit dem Menüpunkt ↪*Lineale in allen Signalstreifen anzeigen* werden die Lineale in allen Signalstreifen sichtbar. Sie sind jedoch unabhängig voneinander verschiebbar. Wenn die Lineale in den verschiedenen Signalstreifen genau übereinander stehen sollen, dann stellt man ein Linealpaar wie gewünscht ein, öffnet in demselben Signalstreifen das Kontextmenü und betätigt ↪*Lineale auf alle Signalstreifen anwenden.*

Schließlich können, wie bei ibaAnalyzer, die Signale auch in einem Signalstreifen gemeinsam dargestellt werden.

Dazu mit der Maus auf den Signalnamen des zu verschiebenden Signals zeigen, bis der Mauszeiger seine Form ändert.

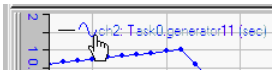


Bild 60 Verschieben eines Signals im Mehrkanal-Oszilloskop

Anschließend das Signal mit gedrückter Maustaste in den anderen Signalstreifen schieben.

Loslassen der Maustaste irgendwo im Signalstriefen: Das Signal bekommt eine eigene Y-Achse.
Loslassen der Maustaste am Signal, das sich bereits im Signalstreifen befindet, wenn ein kleiner Pfeil erscheint: Das Signal wird mit auf die Y-Achse des vorhanden Signals gelegt.

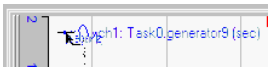


Bild 61 Zusammenfassen von Signalen im Mehrkanal-Oszilloskop

Um die Kurven anschließend noch auseinander halten zu können, sollte der Befehl ↪*Auto-Farbe* im Kontextmenü auf dem Signalnamen ausgeführt werden.

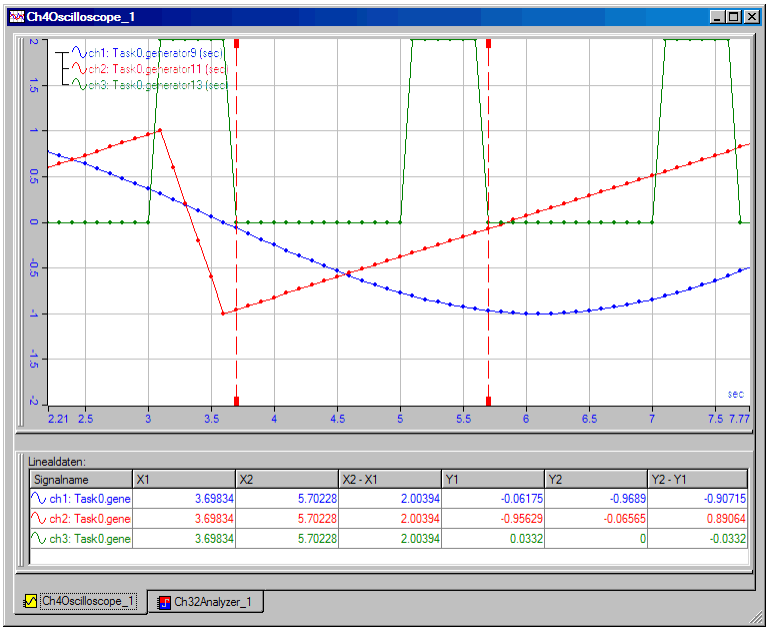


Bild 62 Automatische Farbgebung im Mehrkanal-Oszilloskop

Nun lassen sich alle Signale mit einem Linealpaar vermessen.

Der Logik-Analysator funktioniert entsprechend. Nur gibt es dort keine Y-Achse, da die Signalwerte ja nur TRUE (1) oder FALSE (0) sein können.

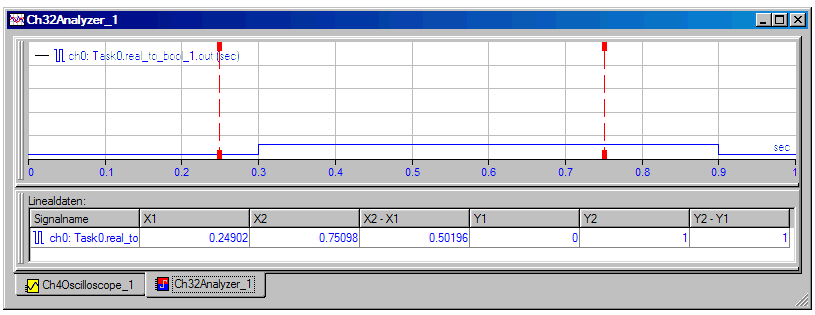


Bild 63 Mehrkanal-Oszilloskop, Variante als Logik-Analysator (CH32 Analyzer)

3

3.13.4.4. Anwendungsbeispiel für Mehrkanal-Oszilloskop und rfft-Fkt.-baustein

Das folgende Beispiel zeigt die Verwendung des Mehrkanal-Oszilloskops in Zusammenhang mit dem rfft-Funktionsbaustein. Besonders hingewiesen sei hier auf die Eingangsgrößen vom Typ ARRAY und die dynamische Berechnung der Skaleneinheiten (xUnit) sowohl für die Zeitachse als auch für die Frequenzachse (FFT).

Funktionsplan

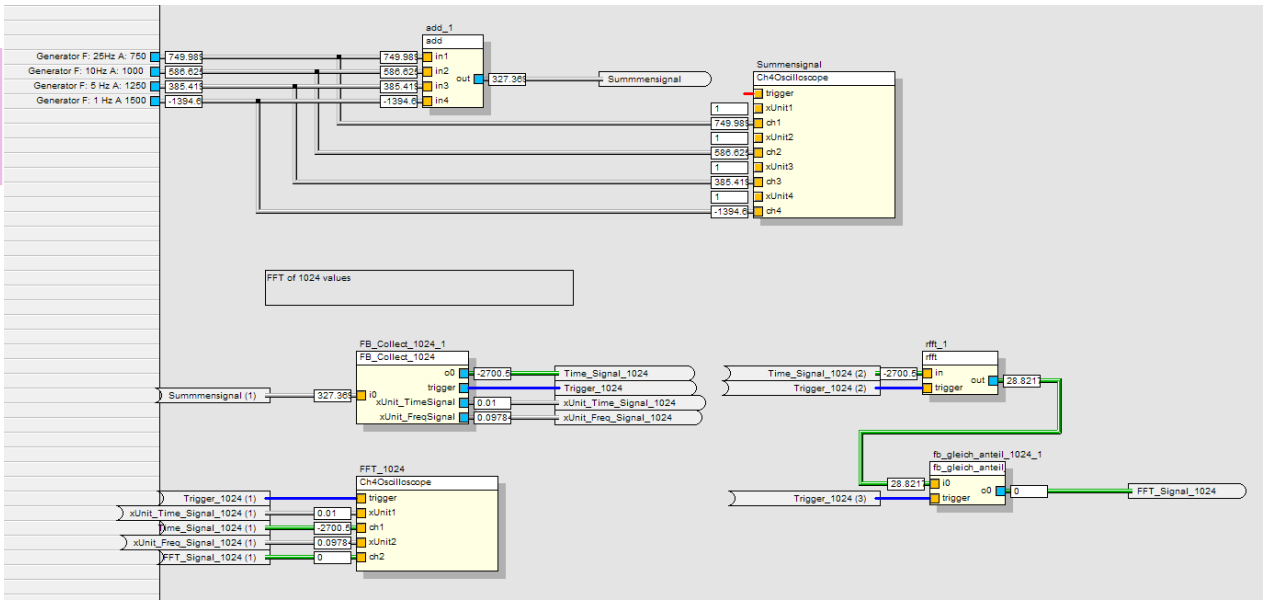


Bild 64 Beispielanwendung für Mehrkanal-Oszilloskop und rfft-Baustein

Erläuterung

1. Vier Signale, von Funktionsgeneratoren erzeugt, werden mittels eines Addier-Bausteins zu einem Summensignal zusammengefasst. Die Einzelsignale werden an einen "Tastkopf" des Mehrkanaloszilloskops angeschlossen. Die Einheiten der x-Achsen (xUnit) haben den Default-Wert 1.

2. Das Summensignal wird mit dem Eingang des speziell für diesen Zweck mit Structured Text erzeugten Bausteins "FB_Collect_1024" verbunden. Dieser Baustein erzeugt aus dem zeit-kontinuierlichen Eingangssignal ein Ausgangssignal vom Typ ARRAY, eindimensional mit 1024 Feldern (Time_Signal_1024). Gleichzeitig werden die xUnits für Zeit- und Frequenzachse (FFT) berechnet.
   Siehe dazu Kasten **Dynamisch skalieren** weiter unten.
   Diese Werte werden mit den Eingängen xUnit1 bzw. xUnit2 eines weiteren "Tastkopfes" für das Mehrkanaloszilloskop verbunden.
   Schließlich wird von diesem Baustein noch ein binäres Triggersignal generiert, das mit jedem vollendeten Beschreiben des Arrays (alle 1024 Zyklen) für einen Zyklus auf TRUE gesetzt wird.

3. Das Signal *Time_Signal_1024* wird auf den Eingang eines rfft-Funktionsbausteins gelegt. Mit jedem Triggersignal übernimmt der Baustein das Eingangsarray mit den Werten des Summensignals (1024 Samples). Die FFT-Funktion ermittelt das Frequenzspektrum des Eingangssignals und liefert als Ausgangssignal wieder ein Array, diesmal jedoch mit 512 Feldern, mit den Amplituden der einzelnen Frequenzen. In dem Ausgangsarray entspricht jedes Feld (Index) einer Frequenz. Das erste Feld (Index = 0) entpricht dem Gleichanteil der Eingangsgröße. Jeder weitere Index entspricht einer um die zuvor errechnete *xUnit_FreqSignal* höhere

Frequenz. Mit *xUnit_FreqSignal= 0,0978* und einem Index von 0 bis 511 können also Frequenzen bis ca. 50 Hz (511 * 0,0978) abgebildet werden.

4. Der Baustein *fb_gleich_anteil_1024* eleminiert den Gleichanteil, indem mit jedem Triggersignal die erste Zelle des FFT-Ergebnis-Arrays mit dem Wert 0.0 gefüllt wird.

Die Anzeige im Mehrkanaloszilloskop sieht dann wie folgt aus:

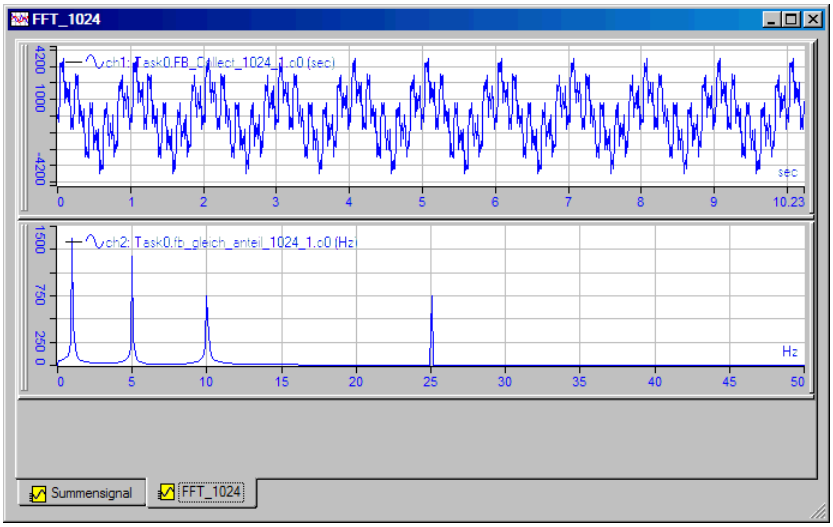


Bild 65 Mehrkanal-Oszilloskop an rfft, Darstellung des Ergebnisses

Im oberen Signalstreifen ist ein Zeitsignalvektor von dem Summensignal mit 1024 Samples dargestellt.

Im unteren Signalstreifen ist der resultierende FFT-Vektor dieses Signales zu sehen, mit deutlichen Ausschlägen bei den Einzelfrequenzen 1 Hz, 5 Hz, 10 Hz und 25 Hz.

Dynamisch skalieren:

In dem Beispiel oben entspricht die xUnit für die Zeitachse der Task-Zykluszeit (10 ms = 0,01 s). Die xUnit der Frequenzachse für die FFT-Darstellung errechnet sich aus der Anzahl der Samples und der Zeitabstände der einzelnen Samples (xUnit Zeit) unter Berücksichtigung des Abtasttheorems.

$$xUnit\_FreqSignal = \frac{1}{(xUnit\_TimeSignal * 2)(1024/2)}$$

Die xUnit der Frequenzachse ist beim Mehrkanaloszilloskop der Skalenwert für ein Sample des FFT-Ergebnisvektors (Ausgang vom rfft-Funktionsbaustein), d.h. der Abstand zwischen zwei FFT-Ergebniseinträgen auf der Frequenzachse.

Für eine korrekte FFT-Berechnung sollte ibaLogic im Signalmanager-Modus laufen.

3.14 Projekt gegen unbeabsichtigte Änderungen sichern

Auf der Menüleiste befindet sich das Icon mit dem Schlüsselsymbol  .

Mit Hilfe dieses Befehles kann das Projekt verriegelt werden. Es ist dann nicht mehr möglich Änderungen durchzuführen. Jedoch kann im Plan weiterhin navigiert werden, z.B. durch Doppelklick ein Makro öffnen.

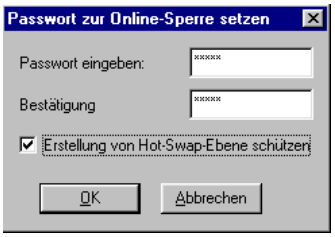
Ist der Button aktiv (gedrückt) dann sind

- ...alle Editierfunktionen abgeschaltet
- ...die unter Windows üblichen Fensterbefehle (verkleinern, vergrößern, schließen) ausgeblendet
- ...Speichern, Lesen und Beenden des Programmes gesperrt.
- ...Hardwaresystemeinstellungen nur lesbar.

Wird der Button erneut gedrückt, dann ist die Editierfunktion wieder freigeschaltet (siehe auch Kapitel Passwortschutz).

3.15 Passwortschutz und andere Schutzfunktionen

Das Projekt kann mit einem Passwort geschützt werden. Ist das Passwort gesetzt, kann der Ver- / Entriegelungsbefehl  nur mit Kenntniss des Passwortes benutzt werden. Mit aktiviertem Schloss und Passwort kann das Projekt gegen unbefugtes Schließen gesichert werden.



Ist die Checkbox „Erstellung von HOT-Swap-Ebene schützen“ ebenfalls aktiviert, dann kann ohne Kenntnis des Passwortes keine HOT_SWAP Ebene aktiviert werden.

Bild 66 Passwortschutz aktivieren

3.16 Der Hot Swap Layer

ibaLogic verfügt über die einzigartige Funktion mehrere Änderungen online durchzuführen zu können, während der reale Online-Prozess ungestört weiterläuft. Zu einem beliebigen späteren Zeitpunkt werden die Änderungen aktiviert (swapped).

Hierzu dient der sogenannte Hot Swap Layer. Es lassen sich damit z.B. auch Änderungen durchführen, die eine Trennung von Netzen erfordern, wie z.B. das Einfügen eines Bausteins mit kurzzeitigem Löschen einer existenten Verbindungslinie zwischen den beiden Bausteinen. Für kontinuierliche Prozesse an Maschinen, welche nicht abgeschaltet werden können (z.B.Papiermaschinen), ist eine solche Funktionalität eine unbedingte Voraussetzung.

Der Hot Swap Layer stellt, wenn er mit ↪*Hot Swap* ↪*Erstellen* oder der Taste  erzeugt worden ist, eine genaue Kopie des Online - Planes dar. Während dieser jedoch einen rosa Hintergrund hat (rosa = aktiv), ist der Planhintergrund des Hot Swap Layers grau. Vor der Erzeugung des Hot Swap-Layers muss der Online-Layer gesichert und abgeschlossen werden.

Der Plan innerhalb des Hot Swap Layers läßt sich verändern und testen (evaluieren). Der Hot Swap Layer selbst wird jedoch nie real online geschaltet – d.h. Änderungen werden zwar compiliert und auch evaluiert, haben jedoch zunächst keine Auswirkung auf die darin enthaltenen Ausgänge. Für die Berechnung des Hot Swap-Layers werden aber die realen Eingangsressourcen verwendet.

Erst wenn der Hot Swap Layer aktiviert wird (↪*Hot Swap* ↪*Auf Online-Ebene anwenden*), wird der Hot Swap Layer zum Online Layer, d.h. es wird zyklusgerecht der neue Plan aktiviert.

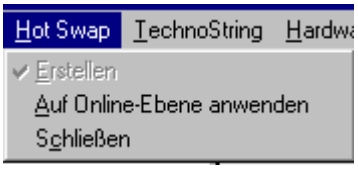


Bild 67 Hot-Swap Layer erstellen

Zu jedem beliebigen Zeitpunkt kann durch drücken des Befehles  zwischen Hot Swap Layer und Online Layer hin und her geschaltet werden.

↪*Hot Swap* ↪*Schließen* verwirft die Änderungen, wenn die Speicheroption verneint wird.

3.16.1. Prinzip des Datenhandlings bzw. der Gedächtnisse bei Hot Swap

Ist ein Hot Swap Layer aktiv, dann dürfen zwischenzeitlich erfolgte Bedienungen über OPC keinesfalls verloren gehen. Ebenfalls dürfen lokale Gedächtnisse von Bausteinen des Online Layers bei der Umschaltung nicht verloren gehen.

ibaLogic berücksichtigt dies, indem nur die Gedächtnisse neu hinzugekommener Bausteine zum Online Plan hinzugefügt werden und die existenten Bausteingedächtnisse 1:1 übernommen werden.

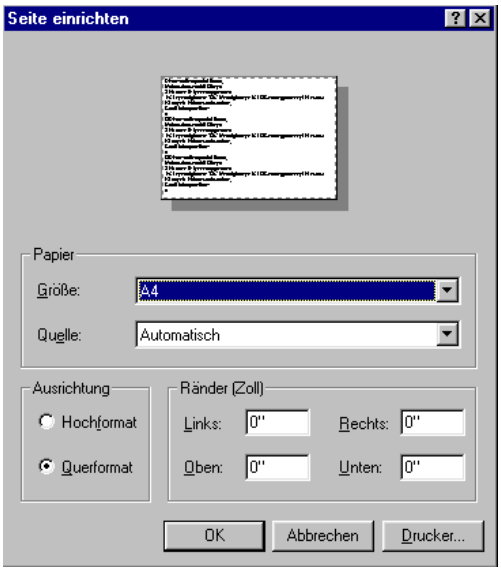
OPC-Eingangs- und Ausgangskonnektoren werden im Hot Swap Bereich nicht evaluiert.

3.17 Projekt drucken

IbaLogic verfügt über umfangreiche Druckfunktionen, welche über Voreinstellungen gesteuert werden können. Prinzipiell wird das WYSIWYG - Verfahren (What you see is what you get) hierfür angewendet.

3.17.1. Festlegen der Seitengrößen für das Gesamtprojekt

Die Größe und Ausrichtung der Druckseiten sollte immer zu Beginn eines neuen Projektes vorgegeben werden. Dadurch werden Nacheditierarbeiten, die aufgrund von Änderungen der Druckseiten ggf. erforderlich werden könnten, vermieden.



Mit dem Befehl ↪*Datei* ↪*Seite einrichten* öffnet sich die links abgebildete Dialogbox.

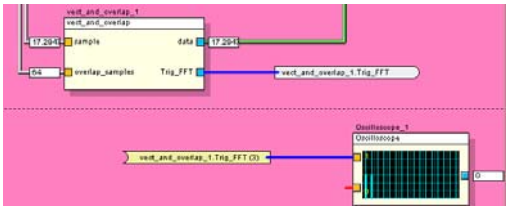
Hier werden die Druckseiteneigenschaften für das gesamte Projekt eingestellt.

Günstig ist A4 Querformat oder größer (andere Formate wie z.B. Letter stehen als Templates zur Verfügung).

Anhand dieser Vorgaben werden die Seiten im ibaLogic Funktionsplan markiert.

Die Ränder sind entweder die I/O Ressourcenleisten (wenn es sich um Seiten ganz links bzw. rechts handelt, oder gestrichelte Linien (wenn z.B. 2 Seiten untereinander / nebeneinander zu liegen kommen).

Auf diesen gestrichelten Linien sollen keine Objekte plaziert werden, da diese sonst beim Drucken abgeschnitten werden.



Links ist eine solche Seitengrenze dargestellt.

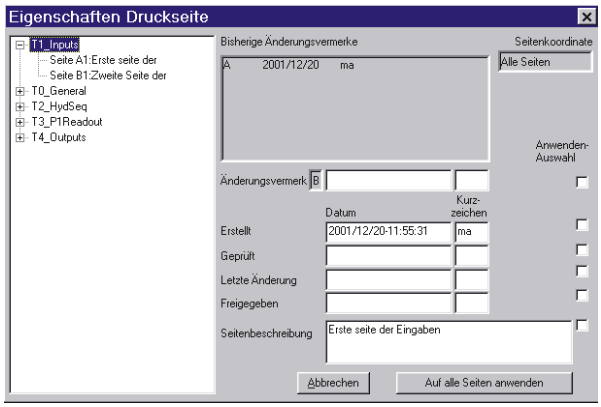
Es erhöht die Übersichtlichkeit des Druckbildes ganz erheblich, wenn für senkrechte Linien IntraPageKonnektoren platziert werden, da dann eine Zielverfolgung gegeben ist (ab V3.80 werden für senkrechte Linien diese Konnektoren zwangsweise erzeugt).

3.17.2. Druckbeschriftung von Seiten

Um den gängigen Anforderungen der Anlagendokumentation gerecht zu werden, wurde das Drucklayout nach internationalen Kriterien ausgelegt.

Jede Druckseite verfügt über Referenzen auf den Ersteller und Erstelldatum (wird automatisch beim Generieren der Seite vom User-Login übernommen) sowie Änderungsvermerke (Änderungshistorie) und Seitentitel. Diese Eigenschaften werden mit ↪*Bearbeiten* ↪*Seite* ↪*Seiteneigenschaften* eingestellt (ein Klick mit der rechten Maustaste in einen freien Bereich einer Seite öffnet ebenfalls das „Bearbeiten"-Menü).

Es erscheint folgende Maske:



Links ist der Strukturseitenbaum sichtbar.

Mitte oben kann die Änderungshistorie nachgelesen werden.

Änderungsvermerke werden darunter eingetragen. Erstellungsdatum und Kurzzeichen werden bereits bei der Seitengenerierung eingetragen, können aber z.B. für Pläne älterer ibaLogic-Funktionen manuell nachgetragen werden. Das Kurzzeichen des Erstellers wird der Angabe unter ↪Datei ↪Einstellungen ↪Bearbeitungs-Einstellungen entnommen.

Rechts unten wird die Seitenbeschreibung eingetragen.

Die Koordinaten der Seite werden oben rechts angezeigt.

Je nachdem, ob in der Baumstruktur eine Seite oder eine Task markiert ist, beziehen sich die Angaben in dieser Maske auf eine einzelne Seite (Kästen für Anwenden-Auswahl nicht sichtbar) oder auf alle Seiten einer Task. Mit Hilfe der Kästchen kann man steuern welche der eingetragenen Informationen bei Betätigung von "Auf alle Seiten anwenden" übernommen werden.

Die Bezeichnung der Seiten (Seitennummerierung) wird dabei wie folgt vorgenommen: (Zeilen sind Buchstaben, Spalten Nummern)

A1	A2	...
B1	B2	...
...	...	...

3.17.3. Druckvoreinstellungen

Der Druckvorgang von ibaLogic kann gesteuert werden, damit keine unnötigen Druckvorgänge erzeugt werden.

Es ist möglich mit einem Vorgang ein Projekt komplett oder auch nur partiell auszudrucken.

Wesentlich ist hier, dass die Selektion nach Druckobjekten erfolgen kann. Mit "Rekursiv" druckt man dann z.B. für jede Instanz eines verwendeten Makros die Inhalte aus.

Auch die Codeinhalte von ST-Bausteinen können mit der "FB Structured Text"-Option ausgedruckt werden.

Neben einem Inhaltsverzeichnis können für den Ausdruck auch Querverweislisten (Querreferenzen) generiert werden.

Querreferenz FB-Name liefert eine Liste aller im Plan verwendeten Objekte, sortiert nach FB-Namen (=FB-Typ). Jeder FB ist mit seiner Beschreibung und Seitenangabe eingetragen.

Querreferenz Beschreibung liefert eine Liste aller im Plan verwendeten Objekte, sortiert nach der Beschreibung (editierbarer Klartext oberhalb eines FBs). Jeder FB ist mit Name und Seitenangabe eingetragen.

Große Schrift verwendet beim Ausdruck einen größeren Schriftfont. Es kommt auf das Papierformat an, welcher Font hier günstiger ist.

In der folgenden Tabelle sind einige typische Ausdrucksformen und die dazu passenden Einstellungen aufgeführt:

Drucken....	Ebene	Aktuelles Fenster	Ausgewählte(r) FB	Task	Makro	Funktionsblock	Rekursiv	Kopfzeile	FB ST	Grafik
eines kompletten Layouts mit allen Tasks, Task-Parametern, Task als ST-code und grafische Funktionsplandarstellung aber ohne Makro-Inhalte und ohne ST-codes der Bausteine	❑			❑				❑	❑	❑
wie oben, nur zusätzlich mit Makro-Inhalten (grafisch)	❑			❑	❑		❑	❑	❑	❑
wie oben, nur zusätzlich mit ST-code der lokalen Funktionsbausteine	❑			❑	❑	❑	❑	❑	❑	❑
nur den grafischen Funktionsplan des aktuell angewählten Tasks aber ohne Makro-Inhalte		❑		❑						❑
nur Parameter der selektierten (lokalen) FBs ohne ST-code			❑			❑				

Tabelle 8 Kombinationen von Druckereinstellungen

3.17.4. Einbinden von firmeneigenen Logos in die Druckseiten

Auf der Startseite des Druckprojektes kann eine individuelle Grafik eingefügt werden. ibaL ogic generiert beim erstmaligen Start die beiden Dateien *ibaLogo.bmp* und *ibaTitle.jpg* und legt diese im Directory *Configuration* ab. Ebenfalls geschieht dies, wenn diese Dateien nicht vorhanden sein sollten.

Sollen nun firmenspezifische Logos (*ibaLogo.bmp*) und Drucktitel (*ibaTitle.jpg*)eingebunden werden, so genügt es, diese beiden Dateien durch andere Dateien gleichen Typs und Namens zu ersetzen.

Die Skalierung (vergößern und verkleinern auf den vorhandenen Druckplatz) erfolgt implizit beim Druckvorgang.

3.17.5. Einbinden eines firmeneigenen Copyrightvermerks

Auf den Druckseiten kann ein individueller Copyrightvermerk eingebracht werden. Dies geschieht wie im vorangegangenen Kapitel ebenfalls dadurch, dass eine spezielle Datei modifiziert wird. Die Datei heißt *ibaCopy.txt*, befindet sich im Verzeichnis *Configuration* und kann einen beliebigen Text beinhalten. Der voreingestellte Text lautet Copyright © 2001. Diese Datei kann mit einem beliebigen ASCII-Editor verändert werden.

3.17.6. Druckbilder

Auf der nächsten Seite ist eine Planseite mit den zugehörigen Eigenschaftsfeldern sowie deren Ursprung (Parametriermasken) abgebildet.

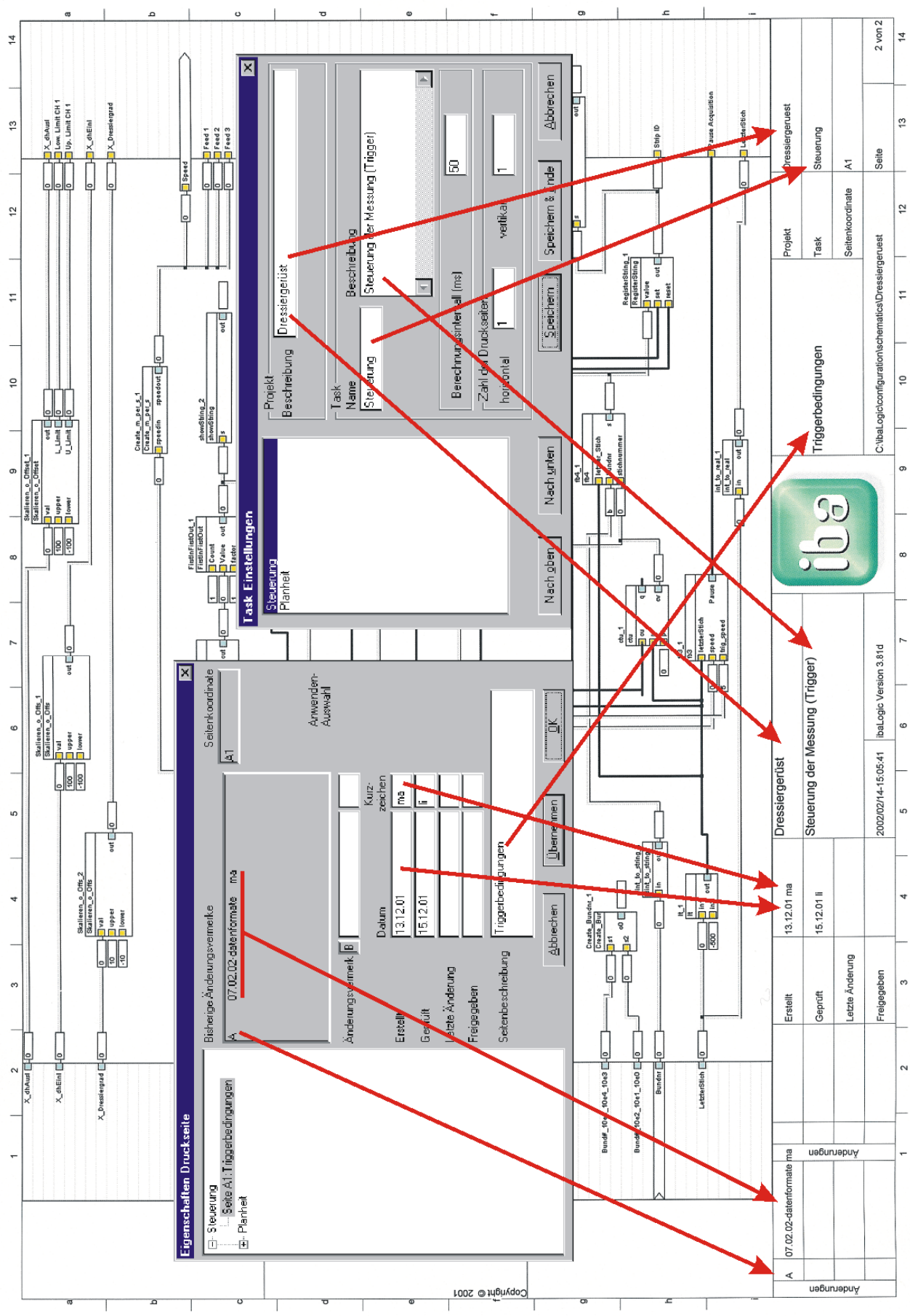
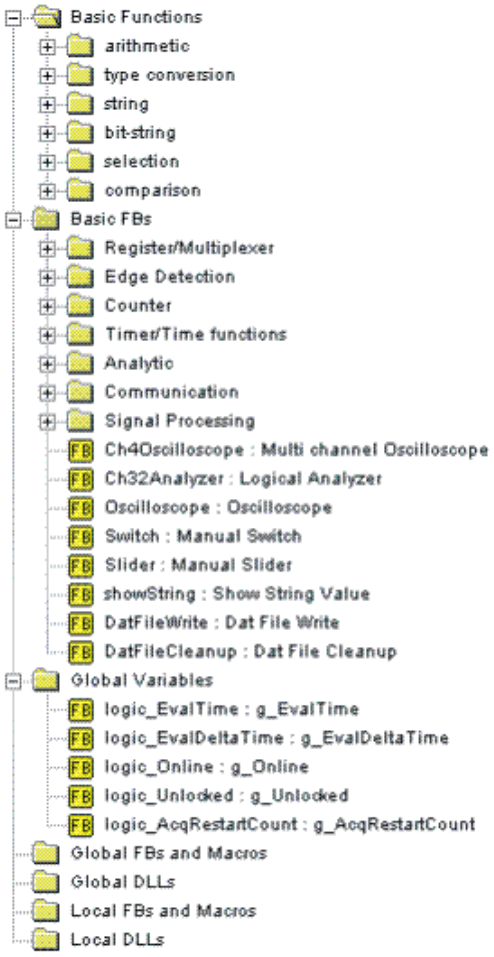


Bild 68 Beispiel einer Druckseite

4 Funktionen und Funktionsbausteine

Zur besseren Übersichtlichkeit ist die Summe aller ibaLogic Funktionen und Funktionsbausteine in sieben Bereiche gegliedert: Über die Lasche "*Funktionen*" der Ressourcenauswahl schaltet man in den Bausteinkatalog um.



ibaLogic Bausteinkatalog

Es gibt folgende Gruppen von ibaLogic-Funktionen und Funktionsbausteinen:

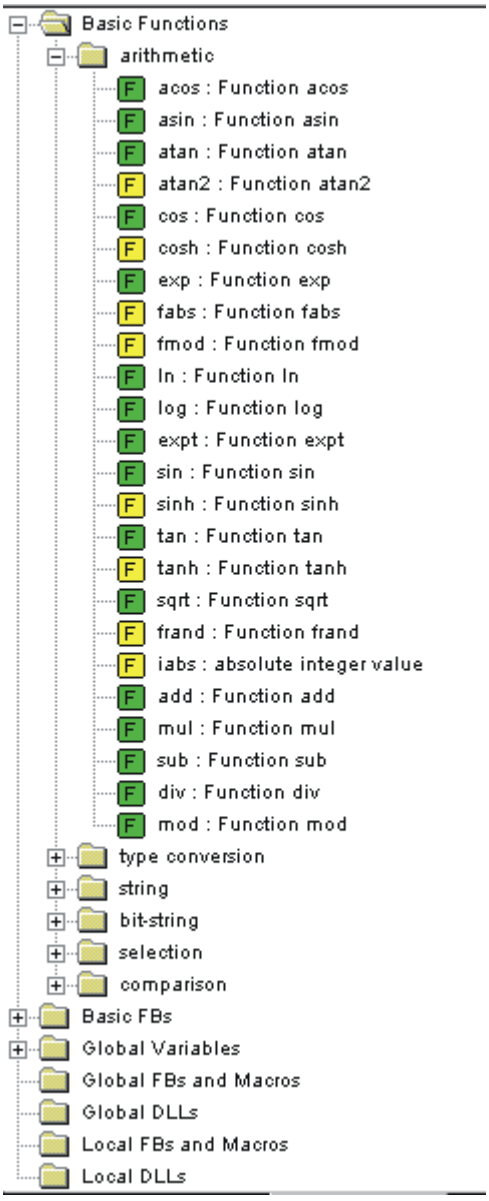
- Basic Functions
- Basic FBs
- Global Variables
- Global FBs and Macros
- Global DLLs
- Local FBs and Macros
- Local DLLs

Die von der IEC Norm 1131-3 vorgeschlagenen Standard-Funktionen und Funktionsbausteine sind mit einem grünen Symbol gekennzeichnet:

Zusätzliche, von ibaLogic bereitgestellte Funktionen oder hilfreiche Funktionsbausteine sind mit einem gelben Symbol gekennzeichnet:

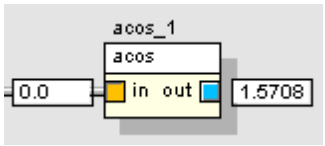
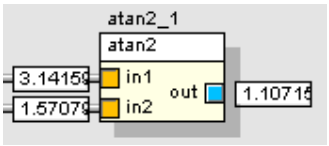
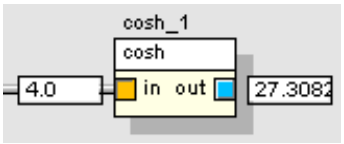
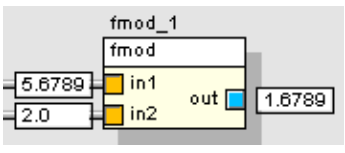
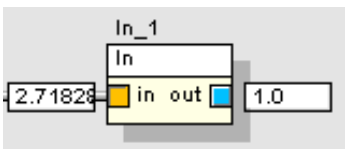
4.1 Basic Functions

4.1.1. Arithmetische Funktionen, Übersicht

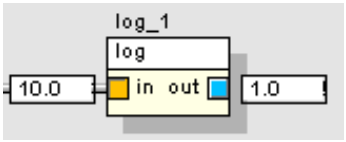
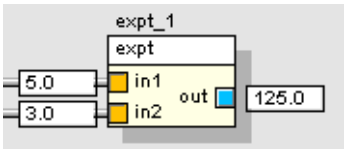
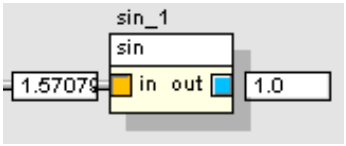
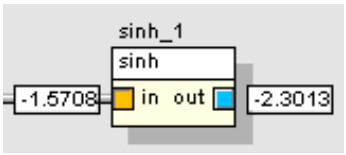
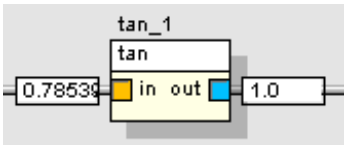
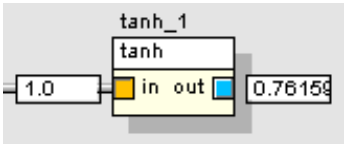
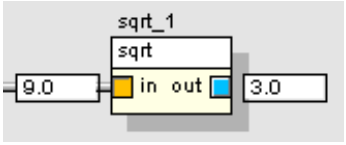
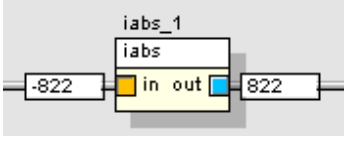


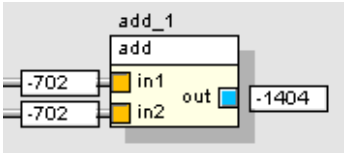
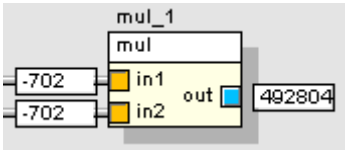
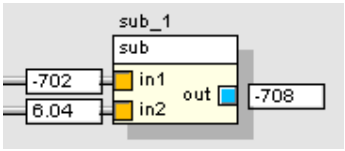
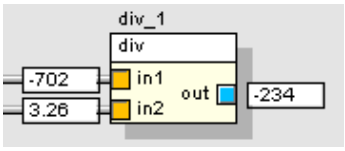
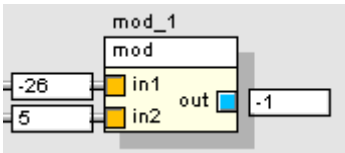
Bausteinübersicht "Basic Functions", "Arithmetic"

- acos arkus cosinus von arg
- asin arkus sinus von arg
- atan arkus tangens von arg
- atan2 arkus-tangens von arg1 über arg2
- cos cosinus von arg
- cosh cosinus hyperbolicus von arg
- exp Natürlicher Exponent e**arg
- fabs Absolutwert der Zahl arg
- fmod Divisionsrest (Real) arg1/arg2
- ln Natürlicher Logarithmus von arg
- log Logarithmus zur Basis 10 von arg
- expt Exponent arg1 ** arg2
- sin sinus von arg
- sinh sinus hyperbolicus von arg
- tan tangens von arg
- tanh tangens hyperbolicus von arg
- sqrt Wurzel von arg
- frand Zufallszahl im Bereich 0...arg
- iabs Absolutwert der Integer-Zahl arg
- add Addition arg1+arg2
- mul Multiplikation arg1*arg2
- sub Subtraktion arg1–arg2
- div Division arg1/arg2
- mod Divisionsrest arg1/arg2

Nr.:	Quelltyp	Arithmetische Funktionen	Zieltyp	Beschreibung, Beispiel
1	LREAL		LREAL	acos: arkus-cosinus(arg); {Bogenmaß} `Ergebnis:= acos(arg);` `Beispiele: 1.57079= acos(0.0);` `3.14159= acos(-1.0);`
2	LREAL		LREAL	asin: arkus-sinus(arg); {Bogenmaß} `Ergebnis:= asin(arg);` `Beispiele: -1.57079= asin(-1.0);` `1.57079= asin(+1.0);`
3	LREAL		LREAL	atan: arkus-tangens(arg); {Bogenmaß} `Ergebnis:= atan(arg);` `Beispiele: 1.0000= atan(π/2.0);` `1.2626= atan(π);`
4	LREAL LREAL		LREAL	atan2: arkus-tangens(arg1 über arg2) `Ergebnis:= atan2(arg1,arg2);` `Beispiele: 1.1071= atan2(π,π/2.0);`
5	LREAL		LREAL	cos: cosinus(arg); {Bogenmaß} `Ergebnis:= cos(arg);` `Beispiele: -1.0000= cos(π);` `+1.0000= cos(2.0*π);`
6	LREAL		LREAL	cosh: cosinus hyperbolicus(arg) `Ergebnis:= cosh(arg);` `Beispiele: +27.3082= cosh(4.0);` `+201.7156= cosh(-6.0);`
7	LREAL		LREAL	exp: Natürlicher Exponent zur Basis e {e^{arg}} `Ergebnis:= exp(arg);` `Beispiele: +2.71828= exp(1.0);` `+0.13533= exp(-2.0);`
8	LREAL		LREAL	fabs: Absolutwert der Zahl (arg) `Ergebnis:= fabs(arg);` `Beispiele: +4.06= fabs(-4.06);` `+3.89= fabs(+3.89);`
9	LREAL LREAL		LREAL	fmod: Divisionsrest (Modulo) (arg1/arg2) `Ergebnis:= fmod(arg);` `Beispiele:` `+1.6789=fmod(5.6789,2.0);` `+1.862= fmod(+3.862,2.0);`
10	LREAL		LREAL	ln: Natürlicher Logarithmus der Zahl (arg) `Ergebnis:= ln(arg);` `Beispiele: +1.00= ln(2.71828);` `-4.00= ln(0.01831);`

4

Nr.:	Quelltyp	Arithmetische Funktionen	Zieltyp	Beschreibung, Beispiel
11	LREAL		LREAL	log: Logarithmus zur Basis 10 der Zahl (arg) `Ergebnis:= log(arg);` `Beispiele: +1.00= log(10.0);` `-4.00= log(0.0001);`
12	LREAL jede Zahl		LREAL	expt: Exponent ($arg1^{arg2}$) `Ergebnis:= expt(arg1;arg2);` `Beispiele: +125.0= expt(5.0,3.0);` `+4.00= expt(16.0,0.5);`
13	LREAL		LREAL	sin: sinus der Zahl (arg) `Ergebnis:= sin(arg);` `Beispiele: +1.00= sin(π/2.0);` `-0.8414= sin(-1.00);`
14	LREAL		LREAL	sinh: sinus hyperbolicus(arg) `Ergebnis:= sinh(arg);` `Beispiele: -2.3013= sinh(-π/2.0);` `+2.3013= sinh(+π/2.0);`
15	LREAL		LREAL	tan: tangens der Zahl (arg) `Ergebnis:= tan(arg);` `Beispiele: +1.00= tan(+π/4.0);` `-1.00= tan(-π/4.0);`
16	LREAL		LREAL	tanh: tangens hyperbolicus(arg) `Ergebnis:= tanh(arg);` `Beispiele: +0.76159= tanh(1.00);` `-0.99627= tanh(-π);`
17	LREAL		LREAL	sqrt: Wurzel(arg) `Ergebnis:= sqrt(arg);` `Beispiele: +3.00= sqrt(9.00);` `+1.4142= sqrt(2.00);`
18	LREAL		LREAL	frand: Zufallszahl im Bereich {0 ... arg} `Ergebnis:= frand(arg);` `Beispiele: +0.07116= frand(1.00);` `+0.92457= frand(1.00);`
19	DINT/INT		INT/DINT	iabs: Absolutwert der INT/DINT-Zahl (arg) `Ergebnis:= iabs(arg);` `Beispiele: +822= iabs(-822);` `+342= iabs(+342);`

Nr.:	Quelltyp	Arithmetische Funktionen	Zieltyp	Beschreibung, Beispiel
20	jede Zahl jede Zahl		jede Zahl	add: Addition der Argumente arg1 + arg2 `Ergebnis:= add(arg1,arg2);` `Beispiele: -1404= add(-702,-702);` `+5.27= add(5.00,0.27);`
21	jede Zahl jede Zahl		jede Zahl	mul: Multiplikation der Argumente arg1 * arg2 `Ergebnis:= mul(arg1,arg2);` `Beispiele: 492804= mul(-702,-702);` `+1.350= mul(5.00,0.27);`
22	jede Zahl jede Zahl		jede Zahl	sub: Subtraktion der Argumente arg1 - arg2 `Ergebnis:= sub(arg1,arg2);` `Beispiele: -708= sub(-702,6.04);` `+4.73= sub(5.00,0.27);`
23	jede Zahl jede Zahl		jede Zahl	div: Division des Argumentes arg1/arg2 `Ergebnis:= div(arg1,arg2);` `Beispiele: -234= div(-702,3.26);` `+18.51= div(5.00,0.27);`
24	INT/DINT INT/DINT		INT/DINT	mod: Divisionsrest (Modulo) arg1/arg2 `Ergebnis:= mod(arg1,arg2);` `Beispiele: -1= mod(-26,5);` `+4= mod(326,7);`

Bemerkungen:

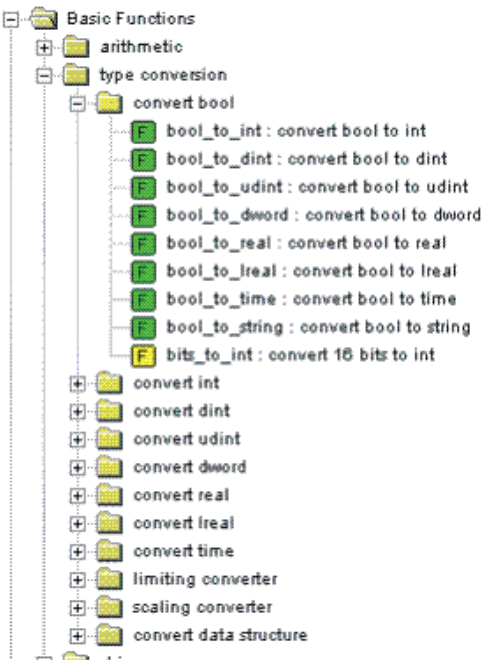
IEC Funktionen sind grün gekennzeichnet, iba-zusätzliche Funktionen gelb

LREAL FABS(ARG); Absolutwert der LREAL-Zahl arg (IEC-Funktionsname ist "ABS")

DINT IABS(ARG); Absolutwert der DINT-Zahl arg (IEC-Funktionsname ist "ABS")

4

4.1.2. Type Conversion (Typumwandlung), Übersicht



Übersicht "Basic Functions, Type Conversion"
Die Konvertierungsbausteine sind in folgende Bereiche gegliedert:

- Convert BOOL
- Convert INT
- Convert DINT
- Convert UDINT
- Convert DWORD
- Convert REAL
- Convert LREAL
- Convert TIME
- Limiting Converter
- Scaling Converter
- Convert data structure

Konvertierungs-Regeln

ibaLogic überprüft auf Übereinstimmung des Datentyps, so dass normalerweise Konvertierungs-Funktion bei der Verschaltung unterschiedlicher Datenformate eingesetzt werden müssen. Darüber hinaus gelten folgende Regeln:

- ❑ Alle vorzeichenbehafteten Integer-Operationen werden mit **32-bit DINT**-Genauigkeit berechnet.
- ❑ Falls notwendig, werden Nicht-STRING Werte (mit Ausnahme von ARRAYs) automatisch in das Datenformat STRING konvertiert.
- ❑ Standard-Konvertierungs-Funktionen werden eingesetzt, um einen Wert in ein ganz bestimmtes Datenformat zu wandeln. Der Funktionsname ergibt sich aus: **<Quelltyp>_to_<Zieltyp>** (siehe Beispiele).
- ❑ Beim Zieltyp BOOL wird der Eingangswert in "FALSE" konvertiert, wenn er den Wert 0 /0.0 /16#0 oder T#0ms hat, andernfalls wird er in "TRUE" gewandelt.
- ❑ DINT, UDINT und DWORD Umwandlungen werden über eine Kopie des aktuellen 4-Byte (32-bit) Datums erzeugt.
- ❑ **REAL** zu **DWORD** Umwandlungen werden über eine Kopie des aktuellen 4-Byte Datums erzeugt.
- ❑ **LREAL** zu **DWORD** Umwandlungen werden wie REAL zu DWORD bearbeitet, wobei LREAL zunächst in REAL konvertiert wird.
- ❑ Die Umwandlung von **REAL/LREAL** in **DINT/UDINT** Datenformat erfolgt über eine numerische Berechnung, wobei davon ausgegangen wird, dass die maximalen Wertebereiche nicht überschritten werden.
- ❑ Beim Zieltyp "**TIME**" wird der Eingangswert "1" / "1.0" mit der Einheit "Sekunde" angenommen.
- ❑ Die spezielle Konvertierungsfunktion "**TRUNC**" wandelt von LREAL zu DINT ohne Rundung.

❑ Bei Konvertierungen von Datentypen mit einem größeren Wertebereich in Datentypen mit einem kleineren Wertebereich wird automatisch ein Limiting Converter angeboten.

Tabelle der Konvertierungsfunktionen

	BOOL	INT	DINT	UDINT
BOOL	N/A	bool_to_int	bool_to_dint	bool_to_udint
INT	int_to_bool	N/A	int_to_dint	int_to_udint
DINT	dint_to_bool	dint_to_int limit_dint_to_int	N/A	dint_to_udint limit_dint_to_udint
UDINT	udint_to_bool	udint_to_int limit_udint_to_int	udint_to_dint limit_udint_to_dint	N/A
DWORD	dword_to_bool	dword_to_int	dword_to_dint	dword_to_udint
REAL	real_to_bool	real_to_int limit_real_to_int	real_to_dint limit_real_to_dint	real_to_udint limit_real_to_udint
LREAL	lreal_to_bool	lreal_to_int limit_lreal_to_int	lreal_to_dint limit_lreal_to_dint trunc	lreal_to_udint limit_lreal_to_udint
TIME	time_to_bool	time_to_int	time_to_dint	time_to_udint
STRING	N/A	N/A	(atoi)	N/A
Bits	N/A	bits_to_int	N/A	N/A

	DWORD	REAL	LREAL	TIME
BOOL	bool_to_dword	bool_to_real	bool_to_lreal	bool_to_time
INT	int_to_dword	int_to_real	int_to_lreal	int_to_time
DINT	dint_to_dword	dint_to_real	dint_to_lreal	dint_to_time
UDINT	udint_to_dword	udint_to_real	udint_to_lreal	udint_to_time
DWORD	N/A	dword_to_real	dword_to_lreal	dword_to_time
REAL	real_to_dword	N/A	real_to_lreal	real_to_time
LREAL	lreal_to_dword	lreal_to_real limit_lreal_to_real	N/A	lreal_to_time
TIME	time_to_dword	time_to_real	time_to_lreal	N/A
STRING	N/A	(atof, atofFmt)	N/A	N/A
Bits	N/A	N/A	N/A	N/A

	STRING	bits	char
BOOL	bool_to_string	N/A	N/A
INT	int_to_string	int_to_bits	N/A
DINT	dint_to_string	N/A	N/A
UDINT	udint_to_string	N/A	N/A
DWORD	dword_to_string	N/A	dword_to_char
REAL	real_to_string	N/A	N/A
LREAL	lreal_to_string	N/A	N/A
TIME	time_to_string	N/A	N/A
STRING	N/A	N/A	N/A
Bits	N/A	N/A	N/A

Beim Zieltyp "**STRING**" werden die Eingangsgrößen gewandelt zu:

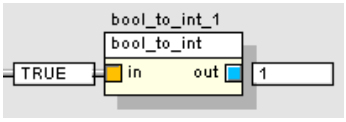
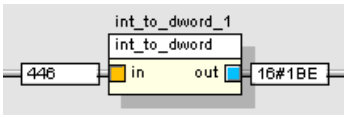
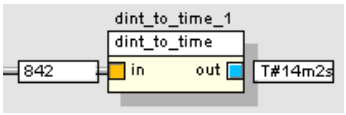
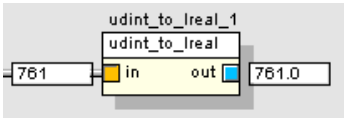
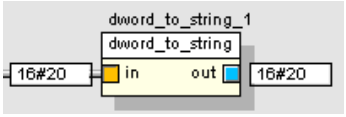
❑ "FALSE" oder "TRUE", beim Eingangstyp BOOL

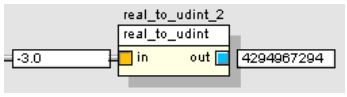
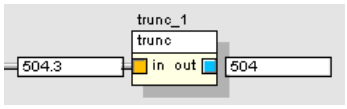
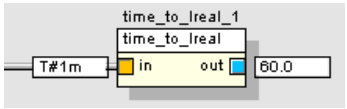
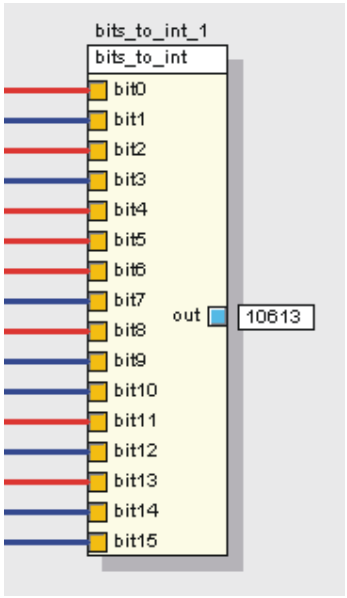
❑ Dezimal-Zeichenfolge (z.B. "-1234" oder "123.456") beim Eingangstyp INT, DINT, UDINT, REAL oder LREAL.

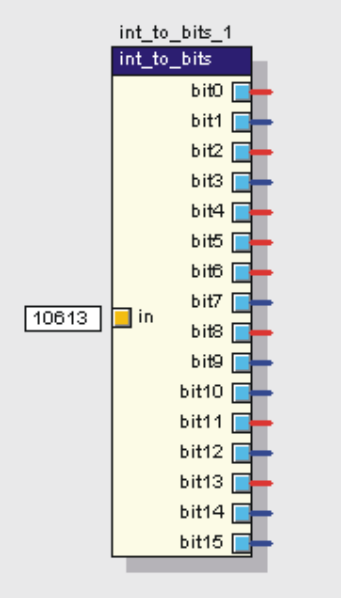
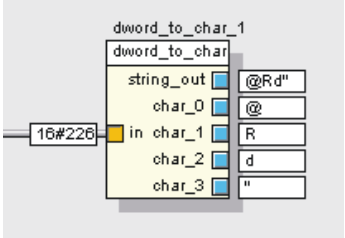
❑ Hex-Zeichenfolge (z.B. "16#56AF3") beim Eingangstyp DWORD.

❑ Zeit-Zeichenfolge (z.B. "T#5m35s200ms") beim Eingangstyp TIME

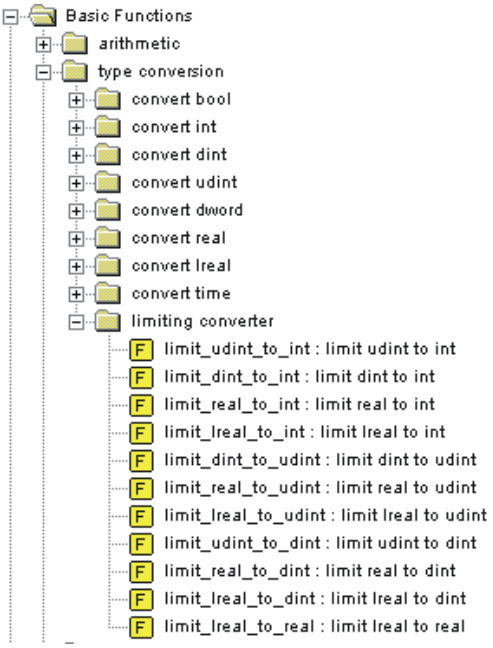
4.1.2.1. Allgemeine Konvertierungsfunktionen

Nr.:	Quelltyp	Typumwandlung	Zieltyp	Beschreibung, Beispiele
1	BOOL	Beispiel: bool_to_int 	INT DINT UDINT DWORD REAL LREAL TIME STRING	Convert BOOL bool_to_int: TRUE => 1; bool_to_dint: FALSE => 0; bool_to_udint: TRUE => 1; bool_to_dword: TRUE => 16#1; bool_to_real: TRUE => 1.0; bool_to_lreal: FALSE => 0.0; bool_to_time: TRUE => T#1s; bool_to_string: TRUE => TRUE; (bits_to_int, siehe Nr. 9)
2	INT	Beispiel: int_to_dword 	BOOL DINT UDINT DWORD REAL LREAL TIME STRING	Convert INTEGER int_to_bool: 446 => TRUE; int_to_dint: 446 => 446; int_to_udint: 446 => 446; int_to_dword: 446 => 16#1BE; int_to_real: 446 => 446.0; int_to_lreal: 446 => 446.0; int_to_time: 446 => T#7m26s; int_to_string: 446 => 446; (int_to_bits, siehe Nr. 10)
3	DINT	Beispiel: dint_to_time 	BOOL INT UDINT DWORD REAL LREAL TIME STRING	Convert Double INTEGER dint_to_bool: 842 => TRUE; dint_to_int: 842 => 842; dint_to_udint: 842 => 842; dint_to_dword: 842 => 16#34A; dint_to_real: 842 => 842.0; dint_to_lreal: 842 => 842.0; dint_to_time: 842 => T#14m2s; dint_to_string: 842 => 842;
4	UDINT	Beispiel: udint_to_lreal 	BOOL INT DINT DWORD REAL LREAL TIME STRING	Convert Unsigned Double INTEGER udint_to_bool: 761 => TRUE; udint_to_int: 761 => 761; udint_to_dint: 761 => 761; udint_to_dword: 761 => 16#2F9; udint_to_real: 761 => 761.0; udint_to_lreal: 761 => 761.0; udint_to_time: 761 => T#12m41s; udint_to_string: 761 => 761;
5	DWORD	Beispiel: dword_to_string 	BOOL INT DINT UDINT REAL LREAL TIME STRING	Convert Double WORD dword_to_bool: 16#20 => TRUE; dword_to_int: 16#20 => 32; dword_to_dint: 16#20 => 32; dword_to_udint: 16#20 => 32; dword_to_real: 16#20 => 4.48416E-04; dword_to_lreal: 16#20 => 4.48416E-04; dword_to_time: 16#20 => T#3.2ms; dword_to_string: 16#20 => 16#20; (dword_to_char, siehe Nr. 11)

Nr.:	Quelltyp	Typumwandlung	Zieltyp	Beschreibung, Beispiele
6	REAL	Beispiel: real_to_udint 	BOOL INT DINT UDINT DWORD LREAL TIME STRING	Convert REAL real_to_bool: -3.0 => TRUE; real_to_int: -3.0 => -3; real_to_dint: -3.0 => -3; real_to_udint: -3.0 => 4294967294; real_to_dword: -3.0 => 16#C0400000; real_to_lreal: -3.0 => -3.0; real_to_time: -3.0 => T#-3s; real_to_string: -3.0 => -3.0;
7	LREAL	Beispiel: trunc (LREAL zu DINT ohne Runden) 	BOOL INT DINT UDINT DWORD REAL TIME STRING TRUNC	Convert LREAL lreal_to_bool: 0.0 => FALSE; lreal_to_int: 504.3 => 504; lreal_to_dint: 1.6 => 2; lreal_to_udint: 504.3 => 504; lreal_to_dword: 504.3 => 16#43FC2666; lreal_to_real: 504.3 => 504.3; lreal_to_time: 504.3 => T#8m24s300ms; lreal_to_string: 504.3 => 504.3; trunc: 1.6 => 1;
8	TIME	Beispiel: time_to_lreal 	BOOL INT DINT UDINT DWORD REAL LREAL STRING	Convert TIME time_to_bool: T#1m => TRUE; time_to_int: T#1m => 60; time_to_dint: T#-2s500ms => -3; time_to_udint: T#1s => 1; time_to_dword: T#1m => 16#927C0; time_to_real: T#1m => 60.0; time_to_lreal: T#10.5ms => 0.0104; time_to_string: T#1m => T#1m;
9	BOOL	Beispiel: bits_to_int 	INT	Convert 16 bits to int **bits_to_int**: bit0 = TRUE bit1 = FALSE bit2 = TRUE bit3 = FALSE bit4 = TRUE bit5 = TRUE bit6 = TRUE bit7 = FALSE bit8 = TRUE bit9 = FALSE bit10 = FALSE bit11 = TRUE bit12 = FALSE bit13 = TRUE bit14 = FALSE bit15 = FALSE } = 10613

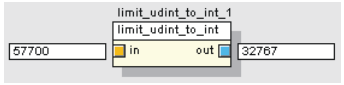
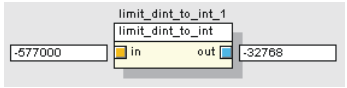
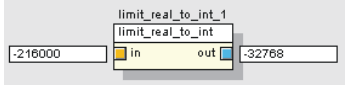
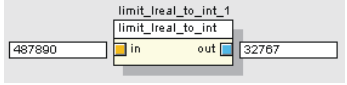
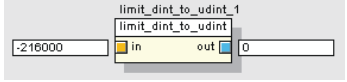
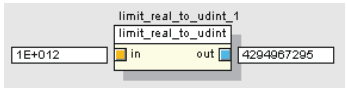
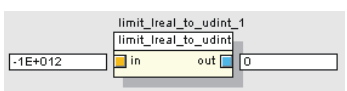
Nr.:	Quelltyp	Typumwandlung	Zieltyp	Beschreibung, Beispiele
10	INT	Beispiel: int_to_bits 	BOOL	Convert int to 16 bits **int_to_bits**: 10613 = { bit0 = TRUE, bit1 = FALSE, bit2 = TRUE, bit3 = FALSE, bit4 = TRUE, bit5 = TRUE, bit6 = TRUE, bit7 = FALSE, bit8 = TRUE, bit9 = FALSE, bit10 = FALSE, bit11 = TRUE, bit12 = FALSE, bit13 = TRUE, bit14 = FALSE, bit15 = FALSE
11	DWORD	Beispiel: dword_to_char 	STRING (4 chars)	Convert DWORD in 4 chars STRING **dword_to_char**: 16#22645240 = { @ R d "

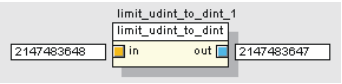
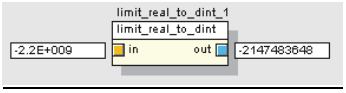
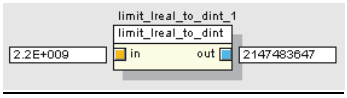
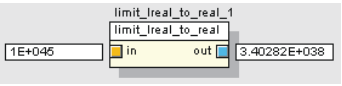
4.1.2.2. Limiting Converters



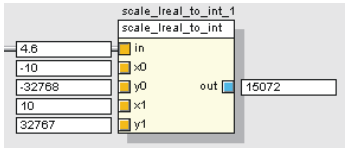
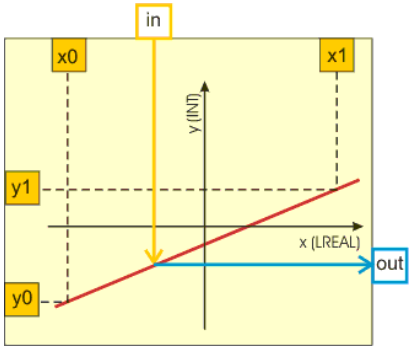
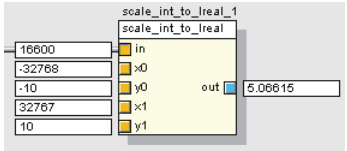
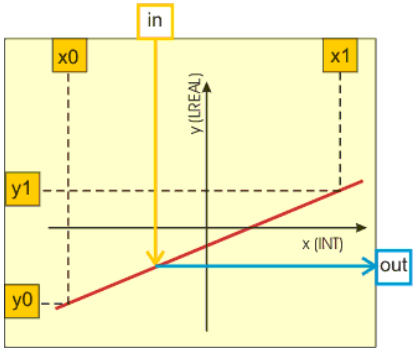
Übersicht "Limiting Converter"
Die limitierenden Konvertierungsbausteine nehmen eine Sonderstellung ein, da sie neben der Typ-Konvertierung auch den Wertebereich der Typen berücksichtigen und bei der Wandlung den Wert ggf. auf die Maximal- bzw. Minimalwerte des Zieltyps begrenzen.

- Limit UDINT to INT
- Limit DINT to INT
- Limit REAL to INT
- Limit LREAL to INT
- Limit DINT to UDINT
- Limit REAL to UDINT
- Limit LREAL to UDINT
- Limit UDINT to DINT
- Limit REAL to DINT
- Limit LREAL to DINT
- Limit LREAL to REAL

Nr.:	Quelltyp	Limitierende Wandler	Zieltyp	Beschreibung, Beispiele
1	UDINT		INT	Limit udint to int limit_udint_to_int: 577000 => 32767;
2	DINT		INT	Limit dint to int limit_dint_to_int: 577000 => 32767;
3	REAL		INT	Limit real to int limit_real_to_int: -216000 => -32768;
4	LREAL		INT	Limit lreal to int limit_lreal_to_int: -216000 => -32768;
5	DINT		UDINT	Limit dint to udint limit_dint_to_udint: -216000 => 0;
6	REAL		UDINT	Limit real to udint limit_real_to_udint: 1*E+12 => 4294967295;
7	LREAL		UDINT	Limit lreal to udint limit_lreal_to_udint: -1*E+12 => 0;

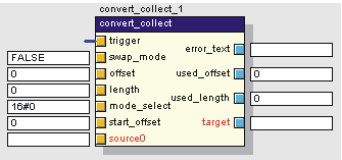
Nr.:	Quelltyp	Limitierende Wandler	Zieltyp	Beschreibung, Beispiele
8	UDINT		DINT	Limit udint to dint limit_udint_to_dint:2147483648 =>2147483647;
9	REAL		DINT	Limit real to dint limit_real_to_dint: -2.2*E+09 => -2147483648;
10	LREAL		DINT	Limit lreal to dint limit_lreal_to_dint: 2.2*E+09 => 2147483648;
11	LREAL		REAL	Limit lreal to real limit_lreal_to_real: 1*E+45 => 3.402823466*E+38

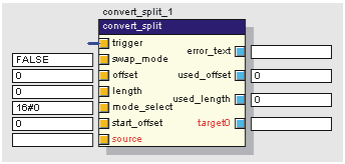
4.1.2.3. Scaling Converters

Nr.:	Quelltyp	Skalierende Wandler	Zieltyp	Beschreibung, Beispiele
1	LREAL LREAL INT LREAL INT		INT	<u>Scale lreal to int</u> Dieser Baustein wandelt einen LREAL-Wert (z.B. physikalische Größe) in einen INTEGER-Wert (z.B. Analogausgang) und skaliert linear. scale_lreal_to_int: 4.6 => 15072 mit –10.0 => -32768 und +10.0 => 32767  Implementierung: `diff_x := x1 - x0;` `if ( diff_x <> 0.0 )` `then` `  a := ( y1 – y0 ) / diff_x;` `  b := y0 - a * x0;` `  dout := a * in + b;` `  out := limit_lreal_to_int( dout );` `else` `  set_valid( out, FALSE );` `end_if;`
2	INT INT LREAL INT LREAL		LREAL	<u>Scale int to lreal</u> Dieser Baustein wandelt einen INTEGER-Wert (z.B. Analogeingang) in einen LREAL-Wert (z.B. physikalische Größe) und skaliert linear. scale_int_to_lreal: 16600 => 5.06615 mit -32768 =>–10.0 und 32767 => +10.0  Implementierung: `diff_x := x1 - x0;` `if ( diff_x <> 0.0 )` `then` `  a := ( y1 - y0 ) / diff_x;` `  b := y0 - a * x0;` `  out = a * in + b;` `else` `  set_valid( out, FALSE );` `end_if;`

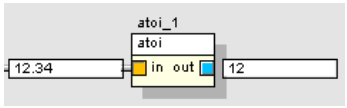
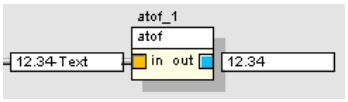
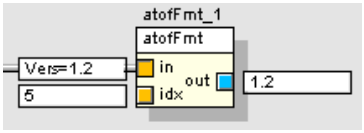
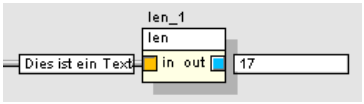
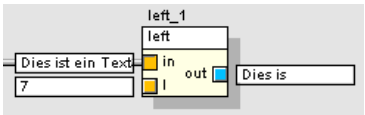
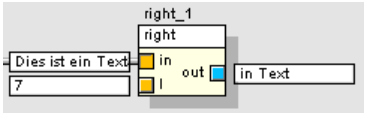
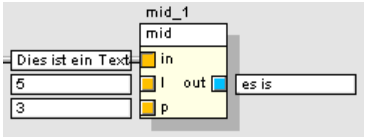
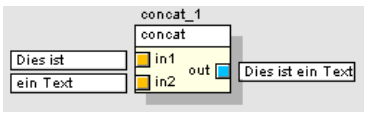
4.1.2.4. Convert data structure

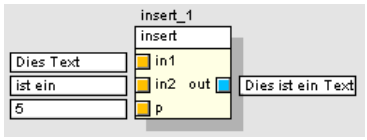
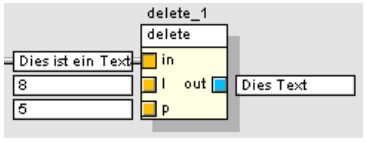
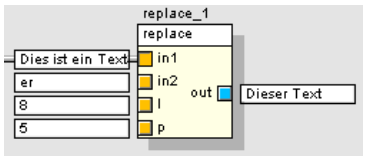
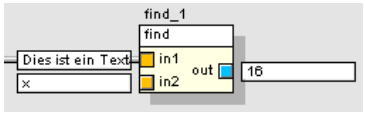
Mit Hilfe dieser Bausteine können Datenstrukturen mit externen Systemen ausgetauscht werden, die komplexere Datenstrukturen verwenden.

Nr.:	Quelltyp	Datenstruktur-Wandler	Zieltyp	Beschreibung, Beispiele
1	BOOL ARRAY ARRAY ARRAY ARRAY UDINT UNTYPED		STRING ARRAY ARRAY UNTYPED	Convert collect Dieser Baustein dient zum Zusammenfassen von mehreren Datenelementen unterschiedlichen oder gleichen Typs (sourceX) zu einer gemeinsamen Datenstruktur (target). Es können bis zu 58 Eingangsgrößen (source0...57) verarbeitet werden. Die Source-Eingänge sind individuell überladbar, d.h. es können unterschiedliche Datentypen angeschlossen werden, inclusive vierdimensionaler Arrays. Eingangsparameter: *trigger* (BOOL): Der Baustein wird nur bearbeitet, wenn trigger = TRUE ist. *swap_mode* (Array of BOOL): Wenn TRUE, dann wird das Datenelement des entsprechenden Source-Eingangs geswappt (zielsystemabhängig). *offset* (Array of UDINT): Byte-Offset pro Source in der Zielstruktur (target); wenn = 0 wird das Datenelement direkt hinter das vorangehende geschrieben. *length* (Array of UDINT): Byte-Länge pro Source in der Zielstruktur; wenn = 0 wird die maximale Länge belegt. *mode_select* (Array of DWORD): Nicht verwendet. Der Index dieser Arrays (0...57) ist den Source-Eingängen zugeordnet. *start_offset* (UDINT): Startadresse in der Zielstruktur, ab der die o.g. Source-Daten eingetragen werden sollen. *sourceX* (untyped): Eingangsdatenelement Ausgangsparameter: *error_text* (STRING): Statusmeldung *used_offset* (Array of UDINT): benutzter Offset pro Source *used_length* (Array of UDINT): benutzte Länge pro Source *target* (untyped): Ziel-Datenstruktur.

Nr.:	Quelltyp Datenstruktur-Wandler	Zieltyp	Beschreibung, Beispiele
2	BOOL ARRAY ARRAY ARRAY ARRAY UDINT UNTYPED  	STRING ARRAY ARRAY UNTYPED	Convert split Dieser Baustein bietet die umgekehrte Funktion wie Convert_collect und arbeitet entsprechend. Eine eingehende Datenstruktur wird in beliebige Bestandteile, d.h. Datenelemente gleichen oder unterschiedlichen Typs zerlegt. Um eine Eingangsstruktur korrekt "auspacken" und lesen zu können, müssen Position, Länge und Typ der darin befindlichen Datenelemente bekannt sein. Eingangsparameter: *trigger* (BOOL): Der Baustein wird nur bearbeitet, wenn trigger = TRUE ist. *swap_mode* (ARRAY of BOOL): Wenn TRUE, dann wird das entsprechende Datenelement geswappt (quellsystemabhängig) *offset* (Array of UDINT): Byte-Offset pro Target in der Quellstruktur (source); wenn = 0 liegt das Datenelement direkt hinter dem vorhergehenden. *length* (Array of UDINT): Byte-Länge pro Target in der Quellstruktur; wenn = 0 wird die maximale Länge gelesen. *mode_select* (Array of DWORD): Nicht verwendet. Der Index dieser Arrays (0...57) ist den Target-Ausgängen zugeordnet. *start_offset* (UDINT): Startadresse in der Quellstruktur, ab der die o.g. Target-Daten eingetragen wurden. *source* (untyped): Eingangsdatenstruktur Ausgangsparameter: *error_text* (STRING): Statusmeldung *used_offset* (Array of UDINT): benutzter Offset pro Target *used_length* (Array of UDINT): benutzte Länge pro Target *targetx* (untyped): Ziel-Datenelemente

4.1.3. String-Funktionen

Nr.:	Quell-typ	String Funktionen	Zieltyp	Beschreibung, Beispiele
1	STRING		DINT	**atoi**: Konvertiert STRING in INTEGER `Ergebnis:= atoi(string);` `Beispiele: 12= atoi('12.34');` `1234= atoi('1234-Text');`
2	STRING		REAL	**atof**: Konvertiert STRING in REAL `Ergebnis:= atof(string);` `Beispiele: 12.34= atof('12.34');` `12.34= atof('12.34-Text');`
3	STRING DINT		REAL	**atofFmt**: Konvertiert STRING in REAL, beginnend mit Startindex "idx" `Ergebnis:= atofFmt(string,idx);` `Beispiele:` `1.2= atofFmt('Vers=1.2',5);` `54.32= atofFmt('a:=54.32',3);`
4	UDINT		UTC-Time-String	**UtcTimeToString**: Konvertiert eine UTC-Zeit-Konstante in einen Zeit-STRING `Ergebnis:= UtcTimeToString(arg);` `Beispiele: '2001/08/10.11:05:49'= UtcTimeToString(997441549);` `'1970/01/01.00:00:01'= UtcTimeTo-String(1);`
5	STRING		DINT	**len**: Länge eines String ermitteln `Ergebnis:= len(string);` `Beispiele:` `17= len('Dies ist ein Text');` `4= len('Text');`
6	STRING DINT		STRING	**left**: Linker Teil eines String mit Länge "l" `Ergebnis:= left(string,l);` `Beispiele: 'Dies is'= left('Dies ist ein Text',7);` `'Die'= left('Dies ist ein Text',3);`
7	STRING DINT		STRING	**right**: Rechter Teil eines String mit Länge "l" `Ergebnis:= right(string,l);` `Beispiele: 'in Text'= right('Dies ist ein Text',7);` `'ext'= right('Dies ist ein Text',3);`
8	STRING DINT DINT		STRING	**mid**: Ausschnitt eines String mit Länge "l" ab Position "p" `Ergebnis:= mid(string,l,p);` `Beispiele: 'es is'= mid('Dies ist ein Text',5,3);` `'ein'= mid('Dies ist ein Text',3,10);`
9	STRING STRING		STRING	**concat**: Verknüpfung von 2 Teilstrings `Ergebnis:= concat(string1,string2);` `Beispiele: 'Dies ist ein Text'= concat ('Dies ist','ein Text');` `'ABCD'= concat('AB','CD');`

Nr.:	Quell-typ	String Funktionen	Zieltyp	Beschreibung, Beispiele
10	STRING STRING DINT		STRING	**insert**: Einfügen von String "in2" in String "in1" ab Position "p" `Ergebnis:=insert(string1,string2,p);` `Beispiele: 'Dies ist ein Text'= insert ('Dies Text','ist ein ',5);` `'ABCDE'= insert ('AE','BCD', 1);`
11	STRING DINT DINT		STRING	**delete**: Löschen von "l"-Zeichen eines String ab Position "p" `Ergebnis:= delete(string,l,p);` `Beispiele: 'Dies Text'= delete('Dies ist ein Text',8,5);` `'BCD'= delete('ABCDE',3,1);`
12	STRING STRING DINT DINT		STRING	**replace**: Ersetzen von "l"-Zeichen des String "in1" durch "in2" ab Position "p" `Ergebnis:= replace(string1,string2,l,p);` `Beispiele: 'Dieser Text'= replace('Dies ist ein Text', 'er', 8, 5);` `'ABXE'= replace('ABCDE','X', 2,3);`
13	STRING STRING		DINT	**find**: Suchen der ersten Übereinstimmung eines Zeichens aus "in2" im String "in1" `Ergebnis:= find(string1,string2);` `Beispiele: 16= find('Dies ist ein Text', 'x');` `1= find('Dies ist ein Text', 'exD');`

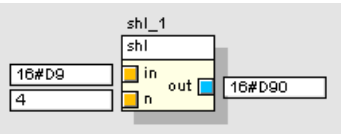
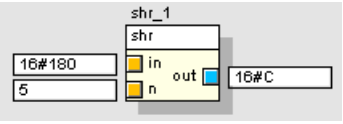
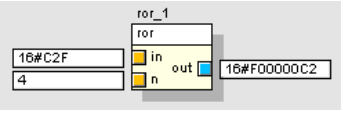
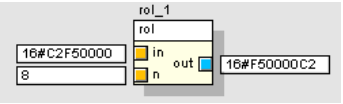
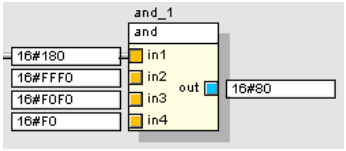
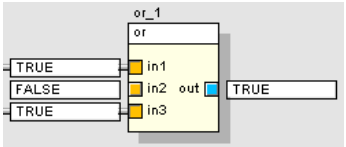
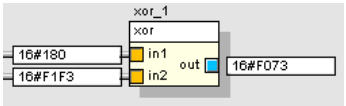
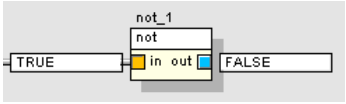
Bemerkungen:

IEC Funktionen sind grün gekennzeichnet, iba-zusätzliche Funktionen gelb.

Die Integer-Variablen dürfen keine negativen Werte (<0) annehmen.

4

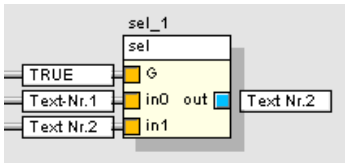
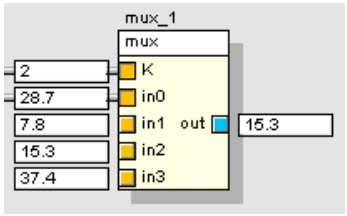
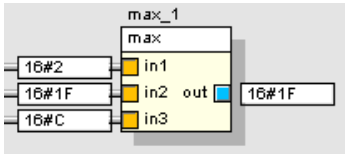
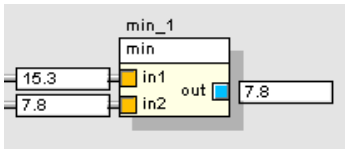
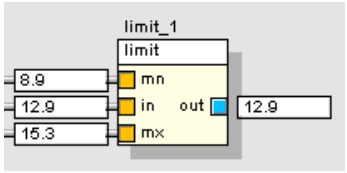
4.1.4. Bit-String-Funktionen und Logische Verknüpfungen

Nr.:	Quelltyp	Bit Shift Funktionen	Zieltyp	Beschreibung, Beispiele
1	DWORD DINT		DWORD	**shl**: Links-Shift von "in" um "n" MOD 32 bits, mit Nullen von rechts auffüllen `Ergebnis:= shl(in,n);` `Beispiele:16#D90= shl(16#D9,4);` `16#180= shl(16#C,5);`
2	DWORD DINT		DWORD	**shr**: Rechts-Shift von "in" um "n" MOD 32 bits, mit Nullen von links auffüllen `Ergebnis:= shr(in,n);` `Beispiele:16#C= shr(16#180,5);` `16#D9= shr(16#D90,4);`
3	DWORD DINT		DWORD	**ror**: Rechts-Rotate von "in" um "n" MOD 32 bits `Ergebnis:= ror(in,n);` `Beispiele:` `16#F00000C2= ror(16#C2F,4);` `16#F500000C= ror(16#CF5,8);`
4	DWORD DINT		DWORD	**rol**: Links-Rotate von "in" um "n" MOD 32 bits `Ergebnis:= rol(in,n);` `Beispiele:` `16#F50000C2= rol (16#C2F50000,8);` `16#45678123= rol (16#12345678,12);`
5	JedesBit ... ... JedesBit		DWORD / BOOL	**and**: Logische UND-Verknüpfung der Eingangs-Variablen (DWORD / BOOL) `Ergebnis:= and(in1,in2,...in_n);` `Beispiele:16#80= and(16#180,` `16#FFF0, 16#F0F0, 16#F0);` `FALSE= and(TRUE,FALSE,TRUE);`
6	JedesBit ... JedesBit		DWORD / BOOL	**or**: Logische ODER-Verknüpfung der Eingangs-Variablen (DWORD / BOOL) `Ergebnis:= or(in1,in2,...in_n);` `Beispiele:TRUE=or(TRUE,FALSE,TRUE)` `;` `16#F1F3=or(16#180,16#F0F0,16#3);`
7	JedesBit ... JedesBit		DWORD / BOOL	**xor**: Logische XOR-Verknüpfung der Eingangs-Variablen (DWORD / BOOL) `Ergebnis:= xor(in1,in2,...in_n);` `Beispiele: FALSE= xor(TRUE,TRUE);` `16#F073=xor(16#180,16#F13);`
8	JedesBit		DWORD / BOOL	**not**: Logische NOT-Verknüpfung der Eingangs-Variable (DWORD / BOOL) `Ergebnis:= not(in);` `Beispiele:FALSE= not(TRUE);` `16#FFFFFE7F=not(16#180);`

Bemerkungen:

Die Anzahl der Eingänge bei den Funktionsbausteinen "AND", "OR" und "XOR" kann frei definiert werden. Dazu Baustein im Arbeitsbereich positionieren, Doppelklick auf Baustein und die "In"-Variable der I/O Konnektors auf den gewünschten Wert einstellen.

4.1.5. Selection-(Auswahl- und Min/Max-) Funktionen

Nr.:	Quelltyp	Selection Funktionen	Zieltyp	Beschreibung, Beispiele
1	BOOL JederTyp JederTyp		JederTyp	**sel**: Auswahl (1 aus 2) mit binären Schalter "G" `out:= in0, wenn G = FALSE [0];` `out:= in1, wenn G = TRUE [1];` `Ergebnis:= sel(G,in0,in1);`
2	BOOL JederTyp .... JederTyp		JederTyp	**mux**: Auswahl (1 aus n) über DINT-Selektor "K" `out:= in0, wenn K = 0;` `out:= in1, wenn K = 1;` `out:= in2, wenn K = 2;` `out:= inn, wenn K = n;` `out:= letzter Wert, wenn K >3;` `Ergebnis:= mux(K,in0,in1,in2,in3);`
3	JederTyp ..... JederTyp		JederTyp	**max**: Maximalwert der Eingänge (1..n) `Ergebnis:= max(in1,in2,inn);` `Beispiele:` `16#1F= max(16#2,16#1F,16#C);` `15.3 = max(12.3,7.8,15.3);`
4	JederTyp ..... JederTyp		JederTyp	**min**: Minimalwert der Eingänge (1..n) `Ergebnis:= min(in1,in2,inn);` `Beispiele:` `16#2= min(16#2,16#1F,16#C);` `7.8 = min(15.3,7.8);`
5	JederTyp JederTyp JederTyp		JederTyp	**limit**: Begrenzung der Eingangsvariablen "in" zwischen "mn" (min.) und "mx" (max.) `Ergebnis:= limit(in,mn,mx);` `Beispiele:` `12.9 = limit(12.9,8.9,15.3);` `15.3 = limit(17.6,8.9,15.3);` `8.9 = limit(2.0,8.9,15.3);`

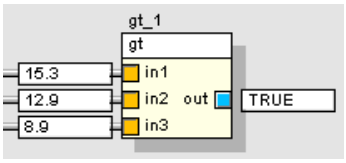
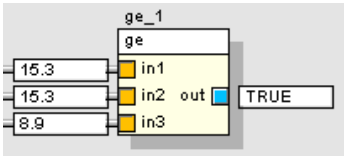
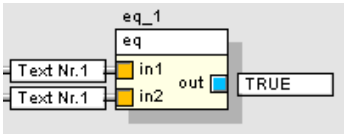
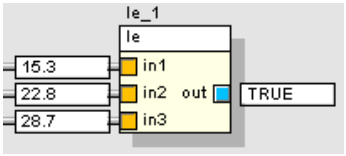
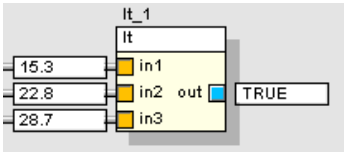
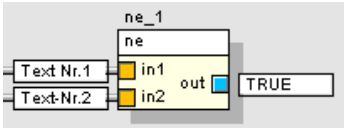
Bemerkungen:

"JederTyp": Jeder elementare Datentyp BOOL/INT/DINT/UDINT/DWORD/REAL/LREAL/TIME/STRING

Die Anzahl der Eingänge bei den Funktionsbausteinen "mux", "max" und "min" kann frei definiert werden. Dazu "In"-Variable der I/O Konnektors auf den gewünschten Wert einstellen.

4

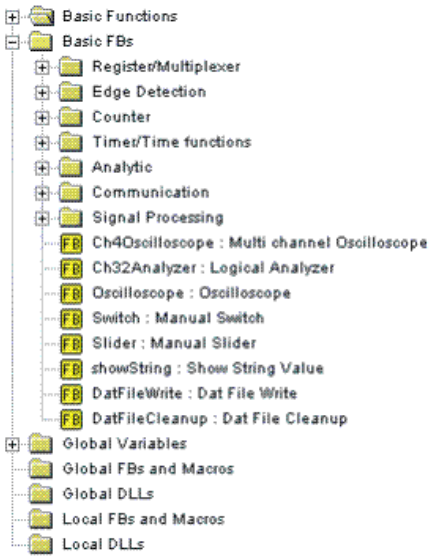
4.1.6. Comparison- (Vergleichs-) Funktionen

Nr.:	Quelltyp	Vergleichsfunktionen	Zieltyp	Beschreibung, Beispiele
1	JederTyp ..... JederTyp		BOOL	**gt**: "greater than" [größer als] (1 aus n) `TRUE, wenn in1 > in2 > in3;` `FALSE, wenn in1 <= in2 <= in3` `Ergebnis:= gt(in1,in2,in3);` `Beispiele:TRUE= gt(15.3,12.9,8.9);` `FALSE= gt(15.3,6.8,8.9);`
2	JederTyp ..... JederTyp		BOOL	**ge**: "greater than or equal" [größer/gleich] (1 aus n) `TRUE, wenn in1 >= in2 >= in3;` `FALSE, wenn in1 < in2 < in3` `Ergebnis:= ge(in1,in2,in3);` `Beispiele: TRUE= ge(15.3,15.3,8.9);` `FALSE= ge(15.3,15.3,18.6);`
3	JederTyp JederTyp		BOOL	**eq**: "equal" [gleich] (1 aus n) `TRUE, wenn in1 = in2 = in3;` `FALSE, wenn in1 <> in2 <> in3` `Ergebnis:= eq(in1,in2,in3);` `Beispiele:` `TRUE= eq('Text 1','Text 1');` `FALSE= eq(15.3,15.3,18.6);`
4	JederTyp ..... JederTyp		BOOL	**le**: "less than or equal" [kleiner/gleich] (1 aus n) `TRUE, wenn in1 <= in2 <= in3;` `FALSE, wenn in1 > in2 > in3` `Ergebnis:= le(in1,in2,in3);` `Beispiele:TRUE= le(15.3,22.8,28.7);` `FALSE= le(15.3,8.9,6.8);`
5	JederTyp ..... JederTyp		BOOL	**lt**: "less than" [kleiner als] (1 aus n) `TRUE, wenn in1 < in2 < in3;` `FALSE, wenn in1 >= in2 >= in3` `Ergebnis:= lt(in1,in2,in3);` `Beispiele:` `TRUE= lt(15.3,22.8,28.7);` `FALSE= lt(15.3,15.8,28.7);`
6	JederTyp JederTyp		BOOL	**ne**: "not equal" [ungleich] (1 aus 2) `TRUE, wenn in1 <> in2;` `FALSE, wenn in1 = in2` `Ergebnis:= ne(in1,in2);` `Beispiele:` `TRUE= ne('Text 1','Text 2');` `FALSE= ne(15.3,15.3);`

Bemerkungen:
- "JederTyp": Jeder elementare Datentyp BOOL/INT/DINT/UDINT/DWORD/REAL/LREAL/TIME/STRING
- Die Anzahl der Eingänge bei den Funktionsbausteinen "gt", "ge" "eq", "le" und "lt" kann frei definiert werden. Dazu "In"-Variable des I/O Konnektors auf den gewünschten Wert einstellen.

4.2 Basic FBs (Basis-Funktionsbausteine)

(FBs) haben beliebig viele, klar definierte Ein-, und Ausgangsparameter und können interne Variablen verwenden, d.h. sie besitzen ein Gedächtnis. Ein Zähler ist ein gutes Beispiel für einen Funktionsbaustein. Er kann in der gleichen Task oder von verschiedenen Tasks mehrfach und mit jeweils anderen Datensätzen genutzt werden.



Übersicht "Basic FBs", Funktionsbausteine
Die ibaLogic-Funktionsbausteine (FBs) sind in folgende Gruppen unterteilt:

- Register/Multiplexer
- Edge Detection
- Counter
- Timer/Time functions
- Analytic
- Communication
- Signal Processing
- Debug- und Hilfsbausteine
  - Multi channel Oscilloscope
  - Logical Analyzer
  - Oscilloscope
  - Manual Switch
  - Manual Slider
  - Show String Value
  - DatFileWrite
  - DatFileCleanup

4

4.2.1. Register / Multiplexer

Register sind Speicherelemente. Mit "1" am Steuereingang "set" wird der Inhalt des Eingangswertes "value" gespeichert und auf den Ausgang geschaltet. Jede Änderung des Eingangswertes wird nur übernommen, wenn die Steuervariable "set" den Wert "1" hat. Mit Eingang "reset" = 1 wird der Ausgangswert zurückgesetzt. Steuereingang "set" hat Priorität gegenüber "reset" (siehe Timing-Diagramm).

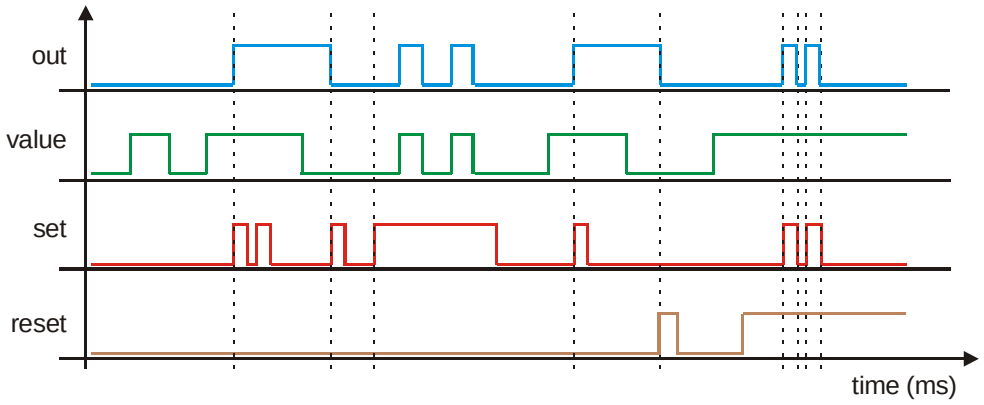
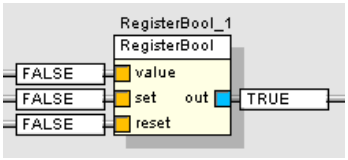
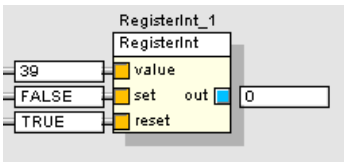
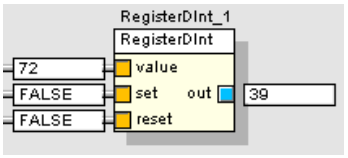
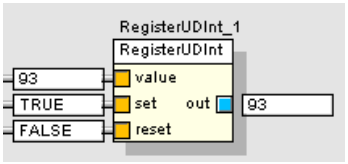
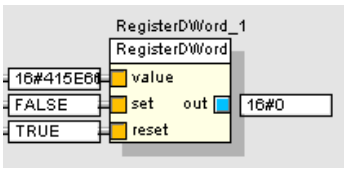
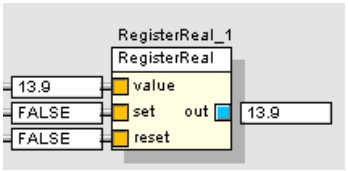
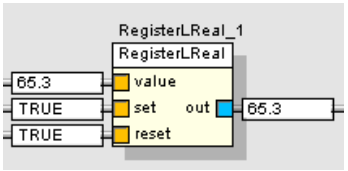
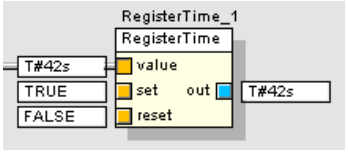
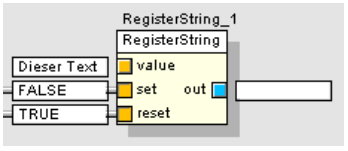
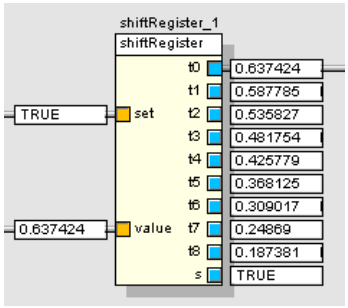
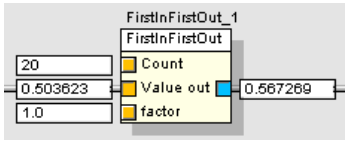
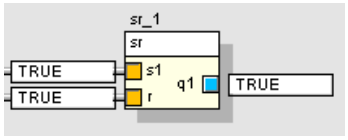
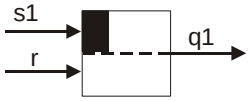
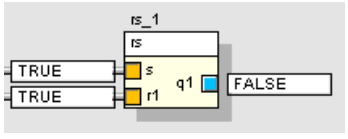
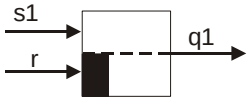


Bild 69 Zeitdiagramm der Register- und Multiplexer-Bausteine

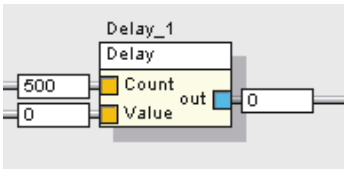
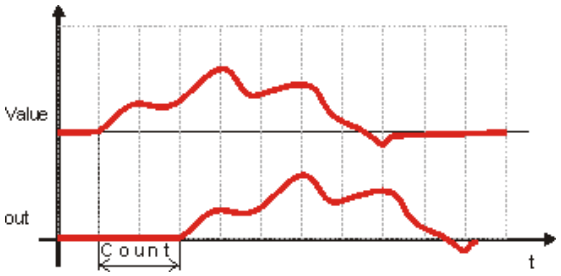
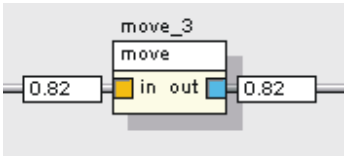
4.2.1.1. Register

Nr.:	Quelltyp	Register-Funktionen	Zieltyp	Beschreibung, Beispiele
1	BOOL BOOL BOOL		BOOL	**RegisterBool**: Datentyp BOOL speichern `Ergebnis:= RegisterBool(value, set, reset);` Beispiele: siehe Timing-Diagramm
2	INT BOOL BOOL		INT	**RegisterInt**: Datentyp INT speichern `Ergebnis:= RegisterInt(value, set, reset);` Beispiele: siehe Timing-Diagramm
3	DINT BOOL BOOL		DINT	**RegisterDInt**: Datentyp DINT speichern `Ergebnis:= RegisterDInt (value, set, reset);` Beispiele: siehe Timing-Diagramm
4	UDINT BOOL BOOL		UDINT	**RegisterUDInt**: Datentyp UDINT speichern `Ergebnis:= RegisterUDInt(value, set, reset);` Beispiele: siehe Timing-Diagramm
5	DWORD BOOL BOOL		DWORD	**RegisterDWord**: Datentyp DWORD speichern `Ergebnis:= RegisterDWord(value, set, reset);` Beispiele: siehe Timing-Diagramm
6	REAL BOOL BOOL		REAL	**RegisterReal**: Datentyp REAL speichern `Ergebnis:= RegisterReal (value, set, reset);` Beispiele: siehe Timing-Diagramm
7	LREAL BOOL BOOL		LREAL	**RegisterLReal**: Datentyp LREAL speichern `Ergebnis:= RegisterLReal (value, set, reset);` Beispiele: siehe Timing-Diagramm
8	TIME BOOL BOOL		TIME	**RegisterTime**: Datentyp TIME speichern `Ergebnis:= RegisterTime (value, set, reset);` Beispiele: siehe Timing-Diagramm
9	STRING BOOL BOOL		STRING	**RegisterString**: Datentyp STRING speichern `Ergebnis:= RegisterString (value, set, reset);` Beispiele: siehe Timing-Diagramm

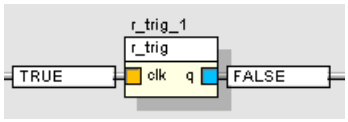
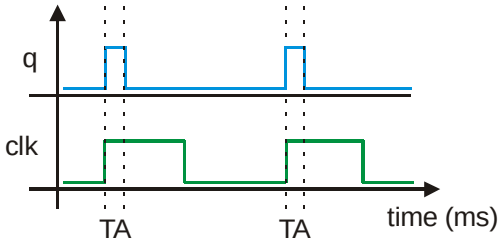
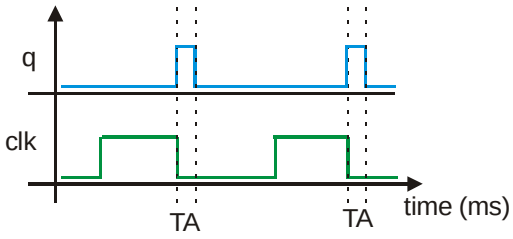
4.2.1.2. Schieberegister, FIFO und Binärwertspeicher

Nr.:	Quelltyp	Register-und Speicher	Zieltyp	Beschreibung, Beispiele
1	BOOL LREAL		LREAL LREAL LREAL LREAL LREAL LREAL LREAL LREAL BOOL	**shiftRegister**: Datentyp REAL/LREAL in einem Schieberegister t0...t8 speichern shift, wenn "set" = TRUE `t0:= value(t[n])aktueller Zyklus` `t1:= value(t[n-1]) letzter Zyklus` `t2:= value(t[n-2])` `...` `t8:= value(t[n-8])` `mit "n" = Taskzyklus`
2	DINT LREAL LREAL		LREAL	**FirstInFirstOut**: Datentyp REAL/LREAL in einem FIFO-Register speichern In jedem Taskzyklus wird das FIFO-Register um eine Position weitergeschoben. Der Eingangswert "value" multipliziert mit "factor" wird in das Schieberegister eingetragen. Count: Länge des FIFO Registers Value: Eingangswert factor: Multiplikator für Eingangs-wert "value"
3	BOOL BOOL		BOOL	**sr**: RS-Flip-Flop (statischer Binärwertspeicher), S-dominant  Wahrheitstabelle Eingangswerte / Ausgang s1 / r / q1 0 / 0 / q1 0 / 1 / FALSE 1 / 0 / TRUE 1 / 1 / TRUE
4	BOOL BOOL		BOOL	**rs**: RS-Flip-Flop (statischer Binärwertspeicher), R-dominant  Wahrheitstabelle Eingangswerte / Ausgang s1 / r / q1 0 / 0 / q1 0 / 1 / FALSE 1 / 0 / TRUE 1 / 1 / FALSE

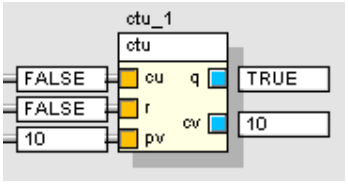
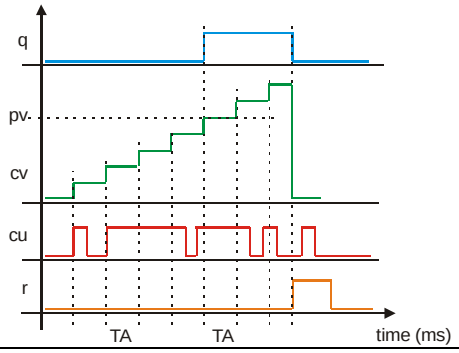
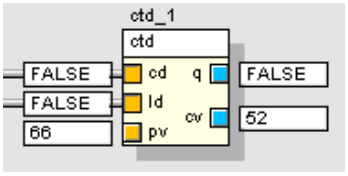
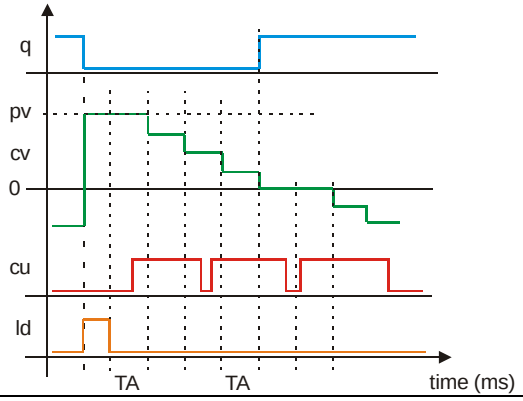
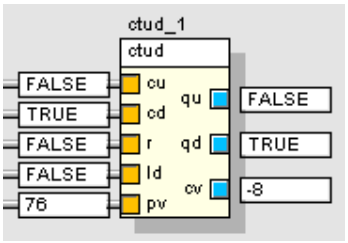
4

Nr.:	Quelltyp	Register-und Speicher	Zieltyp	Beschreibung, Beispiele
5	DINT Jeder Typ		Jeder Typ	**Delay**: Verzögerung bei Werte-Weiterleitung Der Ausgangswert "out" folgt dem Eingangswert "Value" mit einer Verzögerung, die über den Eingang "Count" in Anzahl Zyklen vorgegeben wird.  Bei Verwendung des Datentyps ARRAY ('Value' und 'out') ist der Baustein aus Gründen der Speicherkapazität begrenzt. Wenn die Anzahl der ARRAY-Elemente 64 überschreitet, dann wird der Wertebereich der Verzögerungstiefe von 65536 entsprechend reduziert.
6	Jeder Typ		Jeder Typ	**move**: Feedback register Der Ausgangswert "out" ist eine exakte Kopie des Eingangswertes "in". Der Baustein dient ausschließlich dazu, für die Berechnung in geschlossenen Regelkreisen oder Steuerungsschleifen einen definierten Startpunkt zu setzen. Dieser Baustein gewährleistet, dass die Berechnung einer Schleife stets an derselben Stelle, im Sinne des Signalflusses, beginnt und garantiert so eine eindeutige Abarbeitungsreihenfolge.

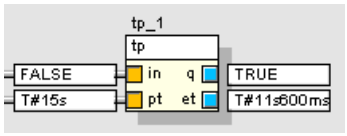
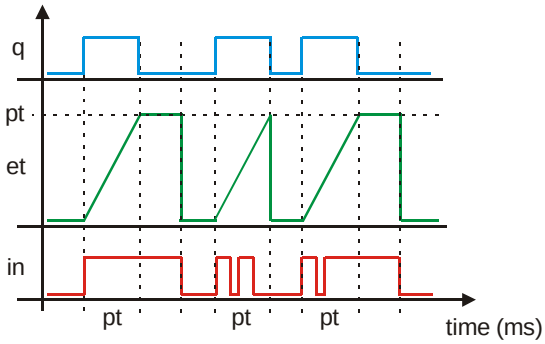
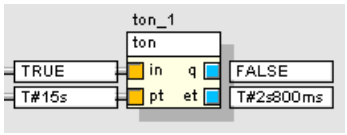
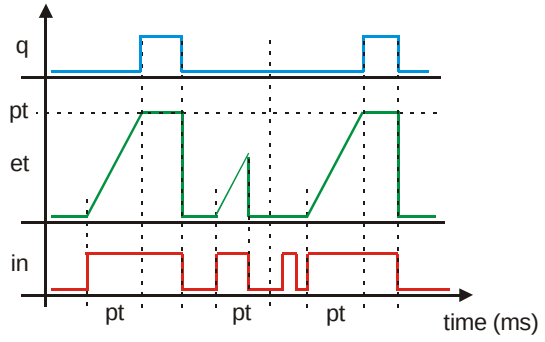
4.2.2. Edge detection (Flankenerkennung)

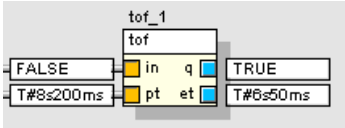
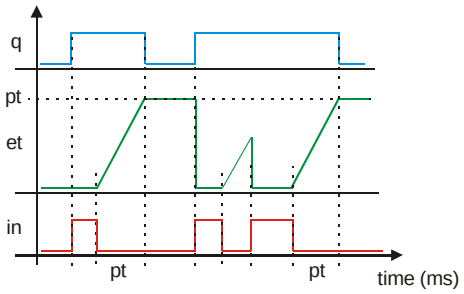
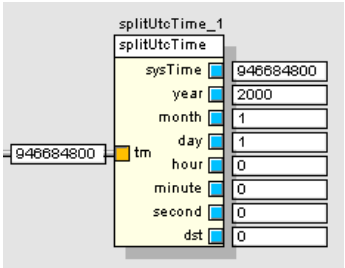
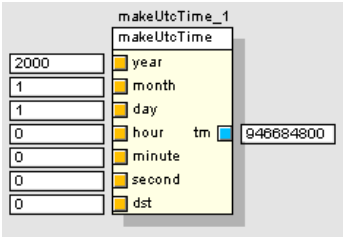
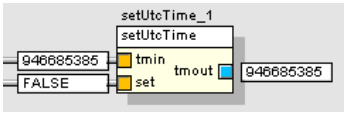
Nr.:	Quelltyp	Flankenerkennung	Zieltyp	Beschreibung, Beispiele
1	BOOL		BOOL	**r_trig**: Rising Edge Detector (Erkennung steigende Flanke) Bei steigender Flanke an Eingang "clk" (0->1) wird Ausgang "q" für einen Taskzyklus = 1 gesetzt. Wenn der Eingang "clk" zum Zeitpunkt des Einschaltens (Systemstart) = TRUE ist, dann generiert der Funktionsbaustein einen Impuls am Ausgang q = TRUE für die Dauer eines Zyklus'. 
2	BOOL		BOOL	**f_trig**: Falling Edge Detector (Erkennung fallende Flanke) Bei fallender Flanke am Eingang "clk" (1->0) wird Ausgang "q" für einen Taskzyklus = 1 gesetzt. Wenn der Eingang "clk" zum Zeitpunkt des Einschaltens (Systemstart) = FALSE ist, dann generiert der Funktionsbaustein einen Impuls am Ausgang q = TRUE für die Dauer eines Zyklus'. 

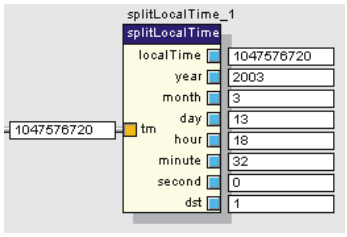
4.2.3. Counter (Zähler)

Nr.:	Quelltyp	Zähler-Funktionen	Zieltyp	Beschreibung, Beispiele
1	BOOL BOOL DINT		BOOL DINT	**ctu**: Up-Counter (Aufwärtszähler) Mit Eingangswert "cu" = 1 wird der Zählwert "cv" pro Abtastzeit um eins inkrementiert. Sobald Zählausgang "cv" >= Zählwert "pv", wird der Ausgang "qu" = 1 gesetzt. 
2	BOOL BOOL DINT		BOOL DINT	**ctd**: Down-Counter (Abwärtszähler) Mit Setzeingang "ld" = 1 wird der Zählwert "cv" auf den Startwert "pv" gesetzt. Bei "cd" = 1 startet die Zählung und der Zählwert "cv" wird pro Abtastzeit um eins dekrementiert. Sobald Zählausgang "cv" <= 0 wird der Ausgang "qd" = 1 gesetzt. 
3	BOOL BOOL BOOL BOOL DINT		BOOL BOOL DINT	**ctud**: Up-Down-Counter (Auf-/Abwärtszähler) Wenn Eingangswert "cu" = 1 wird der Zählwert "cv" pro Abtastzeit um eins inkrementiert. Bei Zählausgang "cv" >= Zählwert "pv" wird der Ausgang "qu" = 1 gesetzt. (Ablaufdiagramm siehe "ctu"-FB) Mit Setzeingang "ld" = 1 wird der Zählwert "cv" auf den Startwert "pv" gesetzt. Bei "cd" = 1 startet die Zählung und der Zählwert "cv" wird pro Abtastzeit um eins dekrementiert. Sobald Zählausgang "cv" <= 0 wird der Ausgang "qd" = 1 gesetzt. (Ablaufdiagramm siehe "ctd"-FB) Mit Rücksetzeingang "r" = 1 wird der Zählausgang auf 0 gesetzt.

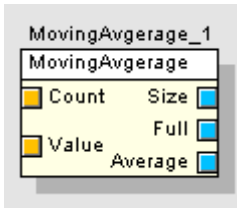
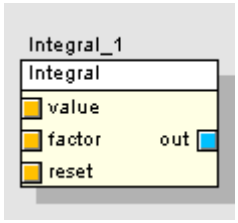
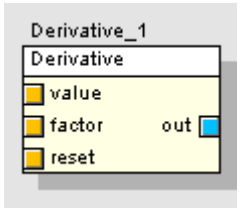
4.2.4. Timer / Time functions (Zeitfunktionen)

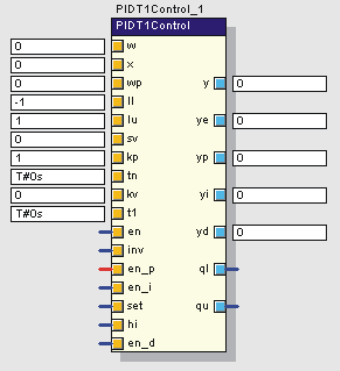
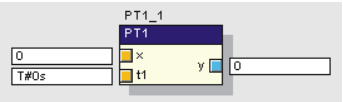
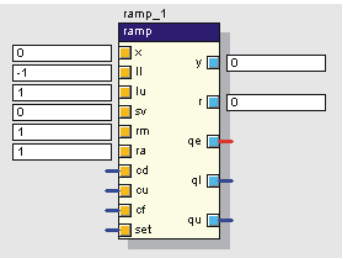
Nr.:	Quelltyp	Zeitfunktionen	Zieltyp	Beschreibung, Beispiele
1	BOOL TIME		BOOL TIME	**tp**: Pulse Timer (Impulsverlängerung) Die ansteigende Flanke am Eingang "in" setzt den Ausgang "q" für die Impulszeit "pt" auf 1. Der Ausgang "q" kann während der Impulszeit nicht rückgesetzt werden. Ausgang "et" zeigt die bereits abgelaufene Zeit an. 
2	BOOL TIME		BOOL TIME	**ton**: On-Delay (Einschaltverzögerung) Die ansteigende Flanke am Eingang "in" startet die Verzögerungszeit "pt". Nach Ablauf der Verzögerungs-zeit wird der Ausgang "q" auf 1 gesetzt, bis 0 am Eingang "in". Der Ausgang "q" wird nicht gesetzt, wenn die Einschaltzeit von "in" kürzer als die Verzögerungszeit ist. Ausgang "et" zeigt die bereits abgelaufene Zeit an. 

Nr.:	Quelltyp	Zeitfunktionen	Zieltyp	Beschreibung, Beispiele
3	BOOL TIME		BOOL TIME	**tof**: Off-Delay (Ausschaltverzögerung) Mit 1 am Eingang "in" wird der Ausgang "q" unverzögert auf 1 gesetzt Die fallende Flanke am Eingang "in" startet die Verzögerungszeit "pt". Nach Ablauf der Verzögerungszeit wird der Ausgang "q" auf 0 gesetzt. Ausgang "q" bleibt unverändert, wenn die Ausschaltzeit von "in" kürzer als die Verzögerungszeit ist. Ausgang "et" zeigt die bereits abgelaufene Zeit an. 
4	UDINT		UDINT DINT DINT DINT DINT DINT DINT	**splitUtcTime**: Aufteilung UTC-Zeit Der Funktionsbaustein wandelt aus dem Eingang "tm" (UTC-Zeit in sec) die Ausgangsvariablen Jahr, Monat, Tag, Stunde, Minute und Sekunde. Bei der UTC-Zeit wird die Zeit in Sekunden ab dem 01.01.1970, 00:00 Uhr gezählt. Beispiele: `tm = 1;` `01.01.1970/00:00:01` `tm = 2592000` `31.01.1970/00:00:00` `tm = 946684800` `01.01.2000/00:00:00` `entspricht 30 Jahre (60*60*24*365) plus 7 Schalttage (60*60*24)`
5	DINT DINT DINT DINT DINT DINT DINT		UDINT	**makeUtcTime**: Erzeugung UTC-Zeit Der Funktionsbaustein erzeugt aus den Eingangsvariablen Jahr, Monat, Tag, Stunde, Minute und Sekunde die UTC-Zeit in sec ab dem 01.01.1970, 00:00 Uhr. Beispiele: `01.01.2000/00:00:00 tm = 946684800` `08.06.2000/12:00:00 tm = 960462000`
6	UDINT BOOL		UDINT	**setUtcTime**: Setze UTC-Zeit Der Funktionsbaustein setzt die UTC-Systemzeit auf den Wert, der am Eingang "tmin" anliegt, wenn Eingang "set" TRUE ist. (Layer muss online sein)

Nr.:	Quelltyp	Zeitfunktionen	Zieltyp	Beschreibung, Beispiele
7	UDINT		UDINT DINT DINT DINT DINT DINT DINT DINT	**splitLocalTime**: Aufteilung der lokal. Systemzeit Der Funktionsbaustein wandelt aus dem Eingang "tm" (lokale Systemzeit in sec) die Ausgangsvariablen Jahr, Monat, Tag, Stunde, Minute und Sekunde sowie die Information zu Sommer-/Winterzeit (dst).

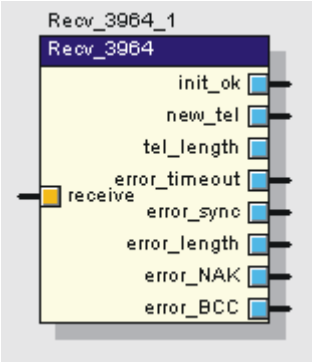
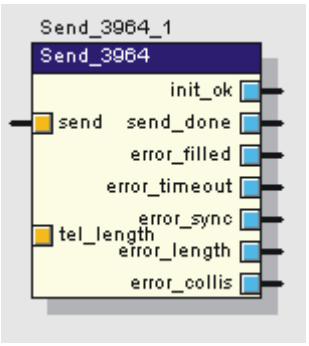
4.2.5. Analytische Funktionen

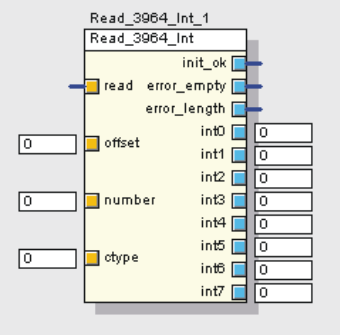
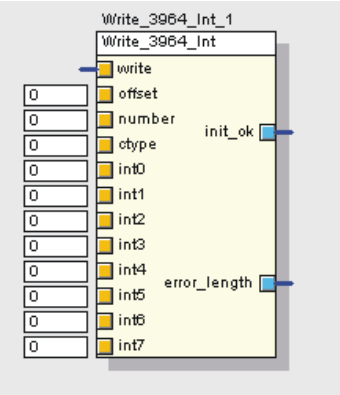
Nr.:	Quelltyp	Analytische Funktionen	Zieltyp	Beschreibung, Beispiele
1	DINT REAL		DINT BOOL REAL	**MovingAverage**: Kumulierter Mittelwert Der Eingangswert "Count" definiert eine Anzahl Werte (=Samples), die für eine Mittelwertberechnung des Wertes "Value" herangezogen werden. Der Ausgangswert "Size" zeigt die Anzahl der zur Mittelwertberechnung verwendeten Werte. Der Ausgang "Full" ist TRUE, wenn die vorgegebene Werteanzahl erreicht ist. Der Ausgangswert "Average" gibt den kumulierten Mittelwert an. Die Mittelwertberechnung wird kontinuierlich durchgeführt. Die Anzahl der Werte kann jederzeit verändert werden.
2	REAL REAL BOOL		REAL	**Integral**: Werteintegral über die Zeit Ausgangswert "out" ist das Integral des Eingangswertes "value", multipliziert mit einem Faktor "factor" über die Zeit. Mit Eingang "reset" = TRUE wird der Ausgang zurückgesetzt.
3	REAL REAL BOOL		REAL	**Derivative**: Ableitung eines Wertes nach Zeit Ausgangswert "out" ist die Ableitung des Eingangswertes "value", multipliziert mit einem Faktor "factor" nach der Zeit. Mit Eingang "reset" = TRUE wird der Ausgang zurückgesetzt.

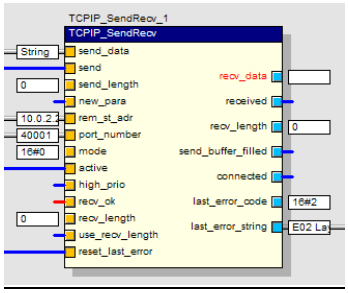
Nr.:	Quelltyp	Analytische Funktionen	Zieltyp	Beschreibung, Beispiele
4	LREAL LREAL LREAL LREAL LREAL LREAL LREAL TIME LREAL TIME BOOL BOOL BOOL BOOL BOOL BOOL BOOL		LREAL LREAL LREAL LREAL LREAL BOOL BOOL	**PIDT1Control**: PIDT1-Regelungsbaustein Universeller PIDT1-Regler, umschaltbar auf die Betriebsarten P-, I-, PI-, PIDT1-Regler. Funktionen: • Anfangswert des Integrators setzen • Momentanwert des Integrators festhalten • Vorsteuerwert wp • Reglerbegrenzung ll und lu • Proportionalbeiwert kp • Nachstellzeit tn • Regelsinn umkehrbar • Anzeige bei Erreichen der eingestellten Grenzen • Anzeige der Regeldifferenz • Anzeige des Reglerausgangswertes Weitere Informationen siehe Kapitel 4.2.9
5	LREAL TIME		LREAL	**PT1**: Verzögerungsglied 1. Ordnung Die Eingangsgröße x wird, dynamisch verzögert um die Glättungszeitkonstante T, auf den Ausgang y gegeben. Implementierung: `t1_t0 := time_to_lreal(t1) / time_to_lreal(EvalDeltaTime);` `y := 1.0 / (1.0 + t1_t0) * (x + t1_t0 * y_old);` `y_old := y;`
6	LREAL LREAL LREAL LREAL LREAL LREAL BOOL BOOL BOOL BOOL		LREAL LREAL BOOL BOOL BOOL	**Ramp**: Rampenbaustein Rampenbaustein mit zwei verschiedenen Rampen, Manuell- und Automatik-Modus. Funktionen: • Sollwertbegrenzung • neuen Sollwert über Rampe anfahren • Sollwert setzen • Anzeige, wenn Grenzwerte überschritten Weitere Informationen siehe Kapitel 4.2.9

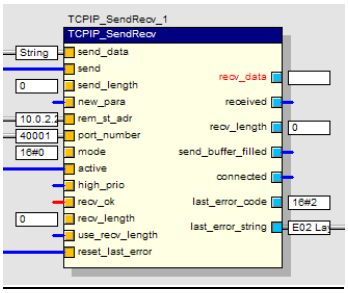
4.2.6. Kommunikationsfunktionen

Diese Gruppe von Funktionsblöcken dient der Kommunikation über die serielle Schnittstelle vom Typ 3964R (DUST) und über TCP/IP.

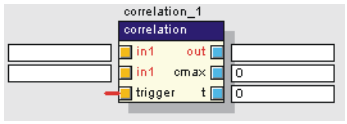
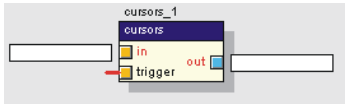
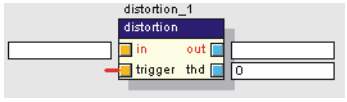
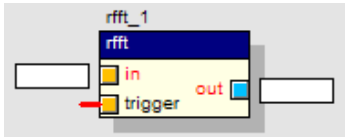
Nr.:	Quelltyp	Kommunikations-funktionen	Zieltyp	Beschreibung, Beispiele
1	BOOL		BOOL BOOL DINT BOOL BOOL BOOL BOOL BOOL	**Recv_3964**: Empfangen eines 3964R-Telegramms (Verwalt.-baustein) Dieser Baustein muss stets einem Lesebaustein vorangestellt sein. Wenn Eingang "receive" = TRUE ist, dann versucht der Baustein ein Telegramm vom 3964R-Treiber zu lesen. Ausgang "init_ok" wird = TRUE, wenn der 3964R-Treiber erfolgreich initialisiert wurde. Ist ein neues Telegramm erfolgreich empfangen worden, wird Ausgang "new_tel" = TRUE gesetzt. "tel_length" gibt die Länge des empfangenen Telegramms in Bytes an. Die Fehlerausgänge "error_..." werden jeweils = TRUE gesetzt, wenn ein entsprechender Fehler erkannt wurde: Timeout, Synchronisation, Telegrammlänge, NAK oder BCC.
2	BOOL DINT		BOOL BOOL BOOL BOOL BOOL BOOL BOOL	**Send_3964**: Senden eines 3964R-Telegramms (Verwaltungsbaustein) Diesem Baustein muss stets ein Write-Baustein vorangestellt werden. Wird der Eingang "send" = TRUE gesetzt, dann versucht der Baustein ein Telegramm der Länge "tel_length" an den 3964R-Treiber zu übergeben. Der Ausgang "init_ok" wird = TRUE, wenn die Initialisierung des 3964R-Treibers erfolgreich war. Ist das Telegramm abgesetzt, dann wird Ausgang "send_done" = TRUE gesetzt. Die Fehlerausgänge "error_..." werden jeweils = TRUE gesetzt, wenn ein entsprechender Fehler erkannt wurde: Puffer voll, Timeout, Synchronisation, Telegrammlänge oder Kollision.

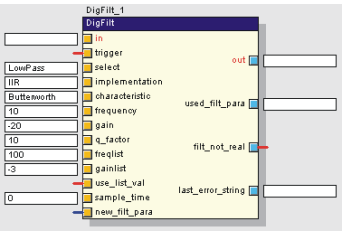
Nr.:	Quelltyp	Kommunikations-funktionen	Zieltyp	Beschreibung, Beispiele
3	BOOL DINT DINT DINT	Example: Read_3964_Int 	BOOL BOOL BOOL DINT DINT DINT DINT 'DINT DINT DINT DINT	**Read_3964_Int**: Lesen eines 3964R-Telegramms und Entpacken des Inhalts von max. acht Integer-Werte ab Offset Diesem Baustein muss stets ein Recv_3964 vorangestellt werden. Wenn Der Eingang "read" = TRUE gesetzt wird, versucht der Baustein Integer-Daten aus einem empfangenen Telegramm zu lesen. Eingang "offset" gibt den Offset für den Adressbeginn der Werte an und "number" die Anzahl der Integerwerte, die gelesen werden sollen (1...8). Am Eingang "ctype" kann der Typ der einlaufenden Daten genauer spezifiziert werden, z.B. falls Swapping erforderlich ist: 0 (default) 4 Bytes 2 2 Bytes 3 2 Bytes und Swapping 4 4 Bytes 5 4 Bytes und Swapping (Read- und Send-Bausteine für INT-, DINT- und WORD-Daten erlauben 2- oder 4-Byte-Größen, Read- und Send-Bausteine für REAL nur 4-Byte) Ausgang "init_ok" wird = TRUE, wenn der 3964R-Treiber erfolgreich initialisiert wurde. Ausgang "error_empty" wird = TRUE, wenn der Empfangspuffer leer ist, so dass kein Telegramm gelesen werden kann. "error_length" wird = TRUE, wenn das Telegramm zu kurz ist. Die Ausgänge "int0" bis "int7" enthalten die entpackten Werte. Für die Bausteine Read_3964_Uint, Read_3964_Word und Read_3964_Float gilt entsprechendes.
4	BOOL DINT DINT DINT DINT DINT DINT DINT DINT DINT DINT DINT	Example: Write_3964_Int 	BOOL BOOL	**Write_3964_Int**: Verpacken von bis zu acht Integer-Werten in ein 3964R-Telegramm Diesem Baustein muss zum Versenden eines Telegramms stets ein Send_3964-Baustein folgen. Wird der Eingang "write" = TRUE gesetzt, dann versucht der Baustein die am Eingang "number" vorgegebene Anzahl von Werten ("int0"..."int7") entsprechend der Typvorgabe am Eingang "ctype" in ein Telegramm an die mit "offset" definierte Stelle zu schreiben. (Erläuterung von ctype-Werten, siehe unter Read_3964_Int.) Der Ausgang "init_ok wird = TRUE, wenn der 3964R-Treiber erfolgreich initialisiert wurde. Der Fehlerausgang "error_length" wird TRUE, wenn das Telegramm zu kurz ist, um die gewünschten Werte noch aufnehmen zu können.

Nr.:	Quelltyp	Kommunikations-funktionen	Zieltyp	Beschreibung, Beispiele
5	untyped BOOL UDINT BOOL STRING UDINT BOOL BOOL BOOL UDINT BOOL BOOL		untyped BOOL UDINT BOOL BOOL DWORD STRING	**TCPIP_SendRecv**: Senden und Empfangen von Daten via TCP/IP Dieser Baustein kann anstelle von auf DLL-Basis erstellter Kommunikationsbausteinen verwendet werden. Eingangsparameter: *Send_data*: Zu sendende Daten (Datentypen String oder Array) *send*: Sendebefehl; in jedem Taskzyklus, in dem der Eingang auf TRUE steht, wird versucht zu senden. *Send_length*: Anzahl zu sendender Bytes. Wenn = 0, dann wird entweder das gesamte Array oder der anliegende String versendet. Falls Wert größer als Array-Gesamtlänge, dann wird auf Arraylänge begrenzt. *New_para*: Übernahme neuer Bausteinparameter, wenn = TRUE *Rem_st_Adr*: Remote Station Address. IP_Adresse des Zielrechners, mit dem kommuniziert wird, im Format nnn.nnn.nnn.nnn. Muss nur angegeben werden, wenn die Instanz des Bausteins innerhalb der TCPIP-Verbindung aktiv ist. *Port_number*: Port-Nummer der Verbindung *Mode*: Bit 0 = 0: Strings werden auf den Wert 0 terminiert. Bit 0 = 1: Strings werden nicht terminiert. Bit 1 = 0: Lesespeicher wird nach dem Lesen nicht gelöscht. Bit 1 = 1: Lesespeicher wird nach dem Lesen gelöscht. *Active*: Diese Instanz des Bausteins ist bei der TCPIP-Verbindung aktiv, wenn = TRUE. *High_prio*: High Priority Mode, für schnelle TCPIP-Kommunikation < 10 ms Zykluszeit *Recv_ok*: Dient zur Flusssteuerung auf Empfangsseite. Wenn auf TRUE gesetzt, dann ist Empfang eines Daten-Telegramms in der Task möglich. *Recv_length*: Empfangsdatenlänge (nur in Verbindung mit Use_recv_length) *Use_recv_length*: Wenn = TRUE, dann werden nur Telegramme der Länge Recv_Length empfangen. *Reset_last_error*: Rücksetzen der Fehler-Ausgänge

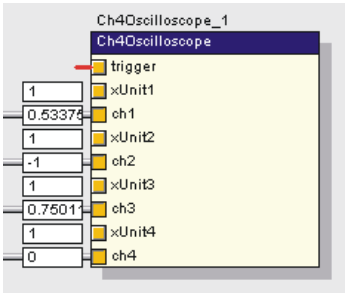
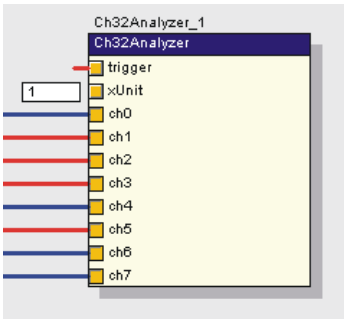
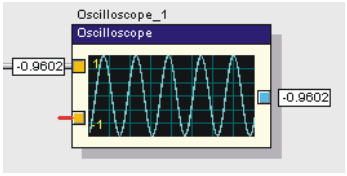
Nr.:	Quelltyp	Kommunikations-funktionen	Zieltyp	Beschreibung, Beispiele
		Fortsetzung TCPIP_SendRecv 		Ausgangsparameter: *Recv_data*: Empfangene Daten (Datentypen String oder Array) *Received*: Status; TRUE = in diesem Taskzyklus wurde ein neues Telegramm empfangen *Recv_length*: Anzahl empfangener Bytes *Send_buffer_filled*: Wenn = TRUE, dann wurde versucht zu senden und der Sendepuffer der Tiefe 1 war noch gefüllt. *Connected*: TRUE = Die Verbindung ist aufgebaut. *Last_error_code*: Fehlercode des letzten aufgetreten Fehlers *Last_error_string*: Fehlertext des letzten aufgetreten Fehlers

4.2.7. Signal Processing (Signalverarbeitung)

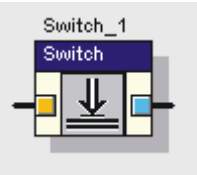
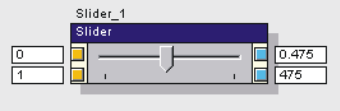
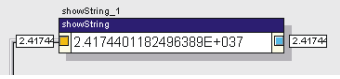
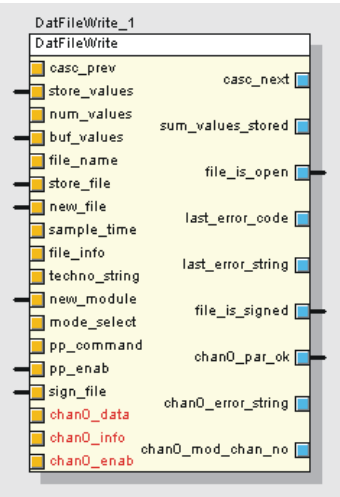
Nr.:	Quelltyp	Signalverarbeitende Funktionen	Zieltyp	Beschreibung, Beispiele
1	ARRAY ARRAY BOOL		ARRAY REAL DINT	**Correlation**: Korrelation von ein / zwei Signalen Dieser Baustein ermittelt die Kreuzkorrelation zwischen zwei Signalen oder, falls der Signalpegel von einem der beiden Eingangssignale zu niedrig ist, die Autokorrelation von einem Signal. Es wird zusätzlich der maximale Korrelationskoeffizient und der Arrayindex ausgegeben. in1, in2, out: Eindimensionale REAL-Arrays mit 2, 4, 8, 16, 32, ...65536 Elementen Startindex 0 Der Baustein wird nur bearbeitet, wenn "trigger" = TRUE ist.
2	ARRAY BOOL		ARRAY	**Cursors**: Grundfrequenz und Harmonische Dieser Baustein ermittelt durch Amplitudenvergleich die Grundfrequenz (Arrayindex 0 von "out") des Eingangssignals "in" und die entsprechenden Indizes der Harmonischen. in, out: Eindimensionale REAL-Arrays mit 2, 4, 8, 16, 32,...65536 Elementen Startindex 0 Der Baustein wird nur bearbeitet, wenn "trigger" = TRUE ist.
3	ARRAY BOOL		ARRAY REAL	**Distortion**: Verzerrungsgrad Dieser Baustein ermittelt den Verzerrungsgrad (Klirrfaktor) des Eingangssignals "in" und die totale harmonische Verzerrung (thd). in1, out: Eindimensionale REAL-Arrays mit 2, 4, 8, 16, 32,...65536 Elementen Startindex 0 Der Baustein wird nur bearbeitet, wenn "trigger" = TRUE ist.
4	ARRAY BOOL		ARRAY	**rfft**: Real Fast Fourier Transformation (schnelle Fourier Transformation) Am Ausgang liefert dieser Funktionsbaustein das einseitige Ergebnis einer FFT (Absolutwert). Am Eingang „in“ muss eine Größe vom Typ ARRAY anliegen, z.B. ein eindimensionales Array von REAL-Variablen mit der Anzahl Werte 2^n. Der Ausgang liefert dann ein Array gleichen Typs (REALs) aber mit der um eine Zweierpotenz reduzierten Dimension $2^{(n-1)}$. Die FFT-Berechnung wird nur aktiviert, wenn der Eingang „trigger“ TRUE ist. Auch nur dann benötigt der Baustein Rechenzeit!

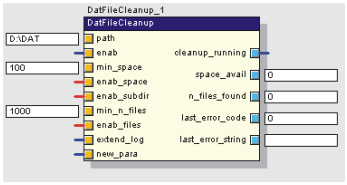
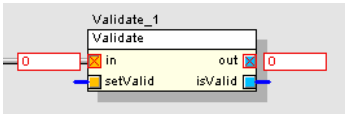
Nr.:	Quelltyp	Signalverarbeitende Funktionen	Zieltyp	Beschreibung, Beispiele
5	UNTYPED BOOL STRING STRING STRING LREAL LREAL LREAL ARRAY ARRAY BOOL LREAL BOOL		UNTYPED STRING BOOL STRING	**DigFilt**: Digitales Filter Digitaler Filterbaustein für kontinuierliche oder gepufferte Signale. Es stehen Tiefpass-, Hochpass-, Bandsperr- und Bandpass-Filtertypen zur Verfügung.Der Filter kann entweder als IIR- (Infinite Impulse Response) oder FIR- (Finite Impulse Response) Filter implementiert werden. Als IIR-Filter sind Butterworth-, Tschebyscheff-, Elliptisch- oder Invers Tschebyscheff- Charakteristiken verfügbar. Als FIR-Filter sind die Fenstertypen Rechteck, Bartlett, Blackman, Hamming, Hanning oder Kaiser verfügbar. Eingestellte Parameter und Fehler werden in Klartext ausgegeben ("last_error_string"). Weitere Informationen siehe Kapitel 4.2.9

4.2.8. Spezielle Bausteine, Hilfsfunktionen

Nr.:	Quelltyp	Spezielle Bausteine, Hilfsfunktionen	Zieltyp	Beschreibung, Beispiele
1	BOOL REAL Jeder Typ REAL Jeder Typ REAL Jeder Typ REAL Jeder Typ			**Ch4Oscilloscope**: Mehrkanal-Oszilloskop Dieser Funktionsbaustein entspricht dem Tastkopf eines herkömmlichen Oszilloskops. Es können ein bis vier Kanäle genutzt werden. Ist der Eingang „trigger“ TRUE, dann läuft die Messung. Die Eingänge „xUnitn“ dienen der Einstellung der X-Achse, die Eingänge „chn“ sind die Messsignaleingänge. Die Anzeige des Oszilloskops wird mit der Symboltaste  oder mit einem rechten Mausklick in den Arbeitsbereich über das Bearbeiten-Menü geöffnet. Siehe dazu auch: 3.13.4 „Mehrkanal-Oszilloskop und Logik-Analysator“
2	BOOL REAL BOOL BOOL BOOL BOOL BOOL BOOL BOOL			**Ch32Analyzer**: Oszilloskop für Binärsignale (Logic Analyzer) Dieser Funktionsbaustein entspricht dem Tastkopf eines herkömmlichen Oszilloskops mit dem Unterschied, dass nur binäre Signale angeschlossen werden können. Es können ein bis 32 Kanäle genutzt werden. Ist der Eingang „trigger“ TRUE, dann läuft die Messung. Der Eingang „xUnitn“ dient der Einstellung der X-Achse, die Eingänge „chn“ sind die Messsignaleingänge. Die Anzeige des Oszilloskops wird mit dem Button  oder mit einem rechten Mausklick in den Arbeitsbereich über das Bearbeiten-Menü (Mehrkanal-Oszilloskop) geöffnet. Siehe dazu auch: 3.13.4 „Mehrkanal-Oszilloskop und Logik-Analysator“
3	Jeder Typ BOOL		REAL	**Oscilloscope**: Einfaches Oszilloskop Dieser Funktionsbaustein dient der einfachen Anzeige eines Signals im Sinne eines Trends. Ist der untere Eingang TRUE, so ist die automatische Skalierung aktiviert. Der Ausgang des Bausteins liefert jeweils den letzten Wert. Achtung: Die Rechenzeit, die der Baustein benötigt ist abhängig von der Größe seiner Darstellung! Je größer das Oszilloskop dargestellt wird, desto mehr Prozessorzeit beansprucht es.

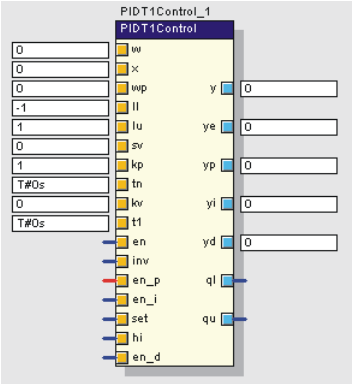
4

Nr.:	Quelltyp	Spezielle Bausteine, Hilfsfunktionen	Zieltyp	Beschreibung, Beispiele
4	BOOL		BOOL	**switch**: Drucktaster und Schalter Dieser Funktionsblock ist ein praktisches Hilfsmittel für manuelle Simulationen von Binärsignalen im Plan. Für die Drucktasterfunktion klickt man mit der linken Maustaste auf das Symbol im FB. Der Ausgang ist so lange TRUE wie die Taste gedrückt ist. Für die Schalterfunktion klickt man mit der rechten Maustaste ein und aus (toggle). Der Eingang des Bausteins dient zum Einschalten des Schalters mit einem Binärsignal (statisch oder Impuls). Die Selbsthaltung muss manuell wieder ausgeschaltet werden.
5	REAL REAL		REAL DINT	**slider**: Schieberegler Abhängig von der Stellung des Schiebers liefert dieser Funktionsbaustein an seinem oberen Ausgang einen Wert, der in den Grenzen der Eingangsvorgaben (Min- und Max-Wert)liegt. Der untere Ausgang liefert den relativen Stellwert des Schiebers in tausendstel Schritten (0...1000). Die Eingänge sind standardmäßig mit 0 bzw. 1 vorbelegt, können aber beliebig verändert werden.
6	Jeder Typ		STRING	**showString**: Wertanzeige Mit diesem Funktionsbaustein können die Inhalte und Werte jeder Variable angezeigt werden, was insbesondere bei sehr großen (langen) Werten oder Texten (STRING) sehr hilfreich ist. Der Baustein interpretiert jeden beliebigen Datentyp am Eingang als String. Der Ausgang ist vom Datentyp STRING.
7	DWORD BOOL DINT BOOL STRING BOOL BOOL LREAL STRING STRING BOOL DWORD STRING BOOL BOOL Jeder Typ Jeder Typ Jeder Typ		DWORD DINT BOOL DWORD STRING BOOL BOOL STRING STRING	**DatFileWrite**: Erzeugen und Füllen eines .dat-Files Mit diesem Funktionsbaustein ist es möglich, direkt aus ibaLogic heraus eine Messdatendatei im iba *.dat-Format zu erzeugen und mit Werten zu füllen. Diese Datei kann dann mit dem Analysewerkzeug ibaAnalyzer weiter ausgewertet werden. Weitere Informationen siehe Kapitel 4.2.9

Nr.:	Quelltyp	Spezielle Bausteine, Hilfsfunktionen	Zieltyp	Beschreibung, Beispiele
8	STRING BOOL UDINT BOOL BOOL UDINT BOOL BOOL BOOL		BOOL UDINT UDINT DINT STRING	**DatFileCleanup**: Aufräumen der Festplatte Mit diesem Funktionsbaustein ist es möglich, eine Aufräumstrategie bzgl. Messdateien (*.dat) für die Festplatte zu organisieren. Entsprechend den Einstellungen, wie sie auch bei ibaPDA vorgenommen werden können, können alte Messdateien nach bestimmten Kriterien gelöscht werden. Weitere Informationen siehe Kapitel 4.2.9
9	Jeder Typ BOOL		Jeder Typ BOOL	**Validate**: Überwachen und setzten von gültigen Signalen Mit diesem Funktionsbaustein ist es möglich, die Gültigkeit eines Eingangssignals zu überwachen. Ausgang *isValid* ist = TRUE, wenn Eingang *in* gültig ist. Wenn Eingang *in* ungültig (invalid) ist, dann ist Ausgang *out* auch ungültig und *isValid* = FALSE. Wenn Eingang *setValid* = TRUE gesetzt wird, dann wird der Ausgang *out* wieder gültig und zwar mit dem letzten gültigen Wert. Mit diesem Baustein kann verhindert werden, dass rekursive Berechnungen (Loops) mit ungültigen Signalen für immer ungültig bleiben. Dazu diesen Baustein in das Netzwerk der Loop einfügen und den Eingang *setValid* gezielt = TRUE setzen.

4.2.9. Komplexe Funktionsbausteine

4.2.9.1. PIDT1Control



4

Funktion und Verwendung

Universeller PIDT1-Regler, umschaltbar auf die Betriebsarten P-, I-, PI- und PIDT1-Regler.

Funktionen:

- ❑ Anfangswert des Integrators setzen
- ❑ Momentanwert des Integrators festhalten
- ❑ Vorsteuerwert wp
- ❑ Reglerbegrenzung ll und lu
- ❑ Proportionalbeiwert kp
- ❑ Nachstellzeit tn
- ❑ Regelsinn umkehrbar
- ❑ Anzeige bei Erreichen der eingestellten Grenzen
- ❑ Anzeige der Regeldifferenz
- ❑ Anzeige der einzelnen P-, I-, DT1-Reglerausgänge

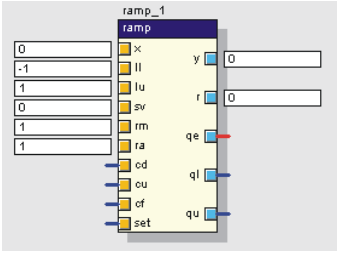
Anschlüsse

Konnektor	Datentyp	Bedeutung / Verwendung
w	LREAL	Sollwert
x	LREAL	Istwert
wp	LREAL	Vorsteuerwert
ll	LREAL	unterer Grenzwert
lu	LREAL	oberer Grenzwert
sv	LREAL	Initialwert
kp	LREAL	P-Verstärkung
tn	TIME	Rücksetzzeit
kv	LREAL	D-Verstärkung
tl	TIME	D-Zeitkonstante
en	BOOL	Reglerfreigabe
inv	BOOL	Vorzeichenumkehr Regelabweichung aktivieren

Forts. PIDT1

Konnektor	Datentyp	Bedeutung / Verwendung
en_p	BOOL	P-Regler aktivieren
en_i	BOOL	I-Regler aktivieren
set	BOOL	Integrator setzen
hi	BOOL	Integrator anhalten
en_d	BOOL	D-Regler aktivieren
y	LREAL	Stellwert
ye	LREAL	Regelabweichung
yp	LREAL	Ausgangswert P-Regler
yi	LREAL	Ausgangswert I-Regler
yd	LREAL	Ausgangswert D-Regler
ql	BOOL	unterer Grenzwert erreicht
qu	BOOL	oberer Grenzwert erreicht

4.2.9.2. Ramp



Funktion und Verwendung

Rampenbaustein mit zwei verschiedenen Rampen, Manuell- und Automatik-Modus.

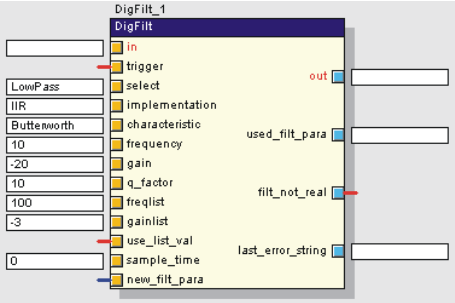
Funktionen:

- ❑ Sollwertbegrenzung
- ❑ neuen Sollwert über Rampe anfahren
- ❑ Sollwert setzen
- ❑ Anzeige, wenn Grenzwerte überschritten

Anschlüsse

Konnektor	Datentyp	Bedeutung / Verwendung
x	LREAL	Eingangswert (Sollwert)
ll	LREAL	unterer Grenzwert
lu	LREAL	oberer Grenzwert
sv	LREAL	Initialwert
rm	LREAL	manuelle Rampe (10/s)
ra	LREAL	automatische Rampe (10/s)
cd	BOOL	Rampe fallend (manuelle Rampensteuerung)
cu	BOOL	Rampe steigend (manuelle Rampensteuerung)
cf	BOOL	Rampe gem. Eingangswert (automatische Rampen-steuerung)
set	BOOL	Ausgangswert setzen
y	LREAL	Ausgangswert; $y_n = y_{n-1} + T_a \bullet r \bullet 10$ T_a = Task-Zykluszeit r = verwendete Rampe
r	LREAL	verwendete Rampe (10/s)
qe	BOOL	Ausgangswert = Eingangswert
ql	BOOL	unterer Grenzwert erreicht
qu	BOOL	oberer Grenzwert erreicht

4.2.9.3. DigFilt - Signale digital filtern



Funktion und Verwendung

Der Baustein dient zum Filtern zeitdiskreter oder gepufferter Signale.

So können Messsignale von störenden Frequenzen befreit (Rauschen oder Brummen) und die Qualität einer nachgeordneten Regelung verbessert werden.

Zusammen mit der Verwendung von rfft-Funktionsbausteien können die im Signal enthaltenen Frequenzen erkannt und herausgefiltert werden.

Anschlüsse

Konnektor	Datentyp	Bedeutung / Verwendung
in	untyped	Das zu filternde Eingangssignal; zulässige Datentypen: REAL und eindimensionale ARRAY of REAL
trigger	BOOL	Der Baustein wird nur bearbeitet, wenn trigger = TRUE ist.
select	STRING	Auswahl des Filtertyps; der angeschlossene String muss exakt die korrekte Schreibweise aufweisen (Groß- und Kleinschreibung beachten!): LowPass.....für Tiefpassfilter HighPass.....für Hochpassfilter BandPass.....für Bandpassfilter BandStop.....für Bandsperrfilter (bei falscher Schreibweise folgt Fehlermeldung Nr. E00)
implementation	STRING	Auswahl der Filter-Implementierung; der angeschlossene String muss exakt die korrekte Schreibweise aufweisen (Groß- und Kleinschreibung beachten!): IIR.....(Infinite Impulse Response) FIR....(Finite Impulse Response) Dieser Eingangsparameter steht in Abhängigkeit zum Eingang "characteristic" (siehe Tabelle weiter unten). (bei falscher Schreibweise folgt Fehlermeldung Nr. E01)
characteristic	STRING	Auswahl der Filter-Charakteristik; der angeschlossene String muss exakt die korrekte Schreibweise aufweisen (Groß- und Kleinschreibung beachten!): Butterworth, Chebyshev, Elliptic oder InvChebyshev (IIR) Rectangular, Bartlett, Blackman, Hamming, Hanning oder Kaiser (FIR) Dieser Eingangsparameter steht in Abhängigkeit zum Eingang "implementation" (siehe Tabelle weiter unten). (bei falscher Schreibweise folgt Fehlermeldung Nr. E01)
frequency	LREAL	Eck- oder Mittenfrequenz des Filters in Hz

Forts. DigFilt

Konnektor	Datentyp	Bedeutung / Verwendung
gain	LREAL	Dämpfung des Filters (pro Dekade oder maximal) in dB
q_factor	LREAL	Qualitätsfaktor, Verhältnis aus Mittenfrequenz und Bandbreite (für Bandpass- und Bandstopfilter)
freqlist	ARRAY[0..3] of LREAL	Liste von Filterfrequenzen Es kann ein Array mit bis zu vier Frequenzwerten angegeben werden, auf die das angeschlossene Signal gefiltert wird. Jede Frequenz kann mit einer eigenen Dämpfung gefiltert werden. Somit lassen sich aus einem Eingangssignal mehrere Frequenzen gleichzeitig herausfiltern.
gainlist	ARRAY[0..3] of LREAL	Liste von Dämpfungswerten , passend zur Liste der Filterfrequenzen.
use_list_val	BOOL	Freigabe (=TRUE) der Verwendung von Frequenz- und Dämpfungswerten aus den Arrays "freqlist" und "gainlist".
sample_time	LREAL	Hier ist die Zeit in ms anzugeben, mit der die Samples des Eingangssignals zu interpretieren sind.
new_filt_para	BOOL	Wenn die (Eingangs-) Filterparameter verändert wurden, muss dieser Eingang für einen Taskzyklus = TRUE gesetzt werden, damit die neuen Parameter wirksam werden.
out	untyped	Filter-Ausgangssignal; der Datentyp wird automatisch vom Eingangssignal übernommen.
used_filt_para	STRING	Ausgabe der tatsächlich verwendeten Filterparameter
filt_not_real	BOOL	Wenn der Filter nicht realisiert werden konnte, weil z.B. mit den angegebenen Parametern das Eingangssignal nicht verrechnet werden konnte, dann wird dieser Ausgang = TRUE gesetzt.
last_error_string	STRING	(letzte) Fehlermeldung

Parameterkombinationen / -abhängigkeiten

wenn "implementation" = ...	....dann "characteristic" = ...
IIR	Butterworth, Chebyshev, Elliptic oder InvChebyshev
FIR	Rectangular, Bartlett, Blackman, Hamming, Hanning oder Kaiser

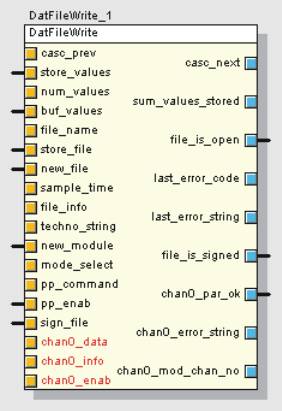
Applikationsbeispiel (Layout) auf CD

sample_layout_digfilt_101.lyt

Diese Beispielanwendung ist geeignet, sich mit der Funktionsweise des Filterbausteins vertraut zu machen. Für die Eingabe der Filterparameter (Art, Implementierung und Charakteristik) stehen Hilfsmittel zur Verfügung.

Das Beispiel zeigt das Filtern sowohl eines gepufferten (Task 0),als auch eines zeitdiskreten Signals (Task 2).

4.2.9.4. DatFileWrite-Baustein – Erzeugen eines iba-Daten-Files (*.dat)



Funktion und Verwendung

Der DatFileWrite-Funktionsbaustein schreibt Daten in dat-Files, die später mit i-baAnalyzer oder jedem anderen Offline-Werkzeug, welches in der Lage ist, die dat-Files zu lesen, untersucht werden können. Die Datentypen der zu speichernden Daten können INTEGER, REAL oder BOOL bzw. ARRAYs dieser Typen sein. Die Daten, die je Kanal in dem dat-File gespeichert werden, könne Einzelwerte oder gepufferte Werte (Pakete) sein. Jeder Kanal kann individuell aktiviert werden, und es ist möglich, anwendungsspezifische Informationen in die dat-Files zu schreiben.

Die Anzahl der Eingangsdaten für den DatFileWrite-Baustein ist skalierbar von mindestens einer bis zu maximal 16 Eingangs- und Ausgangsgruppen. Für jede Gruppe werden stets ein Dateneingang, ein Info-Eingang sowie ein Freigabeeingang zusammen mit einem "par_ok"-Ausgang, einem "error_text"-Ausgang und einem "Mod_chan_no"-Ausgang hinzugefügt.

Anschlüsse

Konnektor	Datentyp	Bedeutung / Verwendung
casc_prev	DWORD	Nicht verwendet, für zukünftige Anwendungen;
store_values	BOOL	Wenn die Datei geöffnet ist, werden in jedem Zyklus, wo dieser Eingang = TRUE ist, Daten gespeichert.
num_values	DINT	Wird nur bei gepufferten Werten verwendet. Die Mindestzahl pro Kanal und Zyklus ist 1. Dieser Wert wird in jedem Zyklus verrechnet, in dem Daten gespeichert werden.
buf_values	BOOL	Wenn dieser Eingang TRUE ist, werden gepufferte Werte verwendet. Verrechnung erfolgt nur einmal zu Beginn jeder neuen Datei.
file_name	STRING	Name der gespeicherten Datei mit kompletter Pfadangabe. Verrechnung erfolgt einmal, wenn eine neue Datei erzeugt wird.
store_file	BOOL	Positive Flanke an diesem Eingang startet die Eingangsüberprüfung, öffnet eine Datei und aktiviert intern die Datenspeicherung. Eine negative Flanke schließt die Datei und initiiert das Post-Processing Kommando, sofern vorgewählt.

Forts. DatFileWrite

Konnektor	Datentyp	Bedeutung / Verwendung
new_file	BOOL	Eine positive Flanke an diesem Eingang schließt die aktuell offene Datei und öffnet eine neue Datei mit dem Namen, der am Eingang "file_name" anliegt. In jedem Fall muss der "store_file"-Eingang TRUE sein. (Entspricht dem kontinuierlichen Speichern in ibaPDA)
sample_ time	LREAL	Dieser Wert wird für den clk-Eintrag im dat-File verwendet. Er beschreibt die Zeit zwischen zwei gespeicherten Werten eines Kanals in Sekunden.
file_info	STRING	optional Dieser String mit anwenderspezifischen Informationen wird beim Schließen der Datei mit eingetragen.
techno_string	STRING	optional Die TechnoString-Information wird beim Schließen der Datei mit eingetragen.
new_module	BOOL	Wenn dieser Eingang TRUE ist, wird ein neuer Kanal an einem neuen Modul hinzugefügt.
mode_select	DWORD	Steuerwort für weitere Funktionen: Die beschriebenen Funktionen werden ausgeführt, wenn die entsprechenden Bits in dem DWORD = TRUE sind. Bit0: Flush Buffers (Puffer leeren) Die Inhalte der internen Datenpuffer für die Online-Komprimierung werden in die Dat-Datei geschrieben. Damit ist es möglich, mit dem ibaAnalyzer bereits Daten aus der noch offenen Messdatei auszulesen und zu analysieren. Bit1: Asynch Access (asynchroner Zugriff) Alle Dateisystem- oder Systemaufrufe werden in einem separaten Thread durchgeführt (asynchron zum Evaluierungs Thread). In diesem Modus gibt es folgende Einschränkungen: 1. Nur eine einzige Dat-Datei kann von einem Funktionsbaustein zu einem Zeitpunkt geöffnet sein, die aktuelle Dat-Datei muss vollständig in der Datei gespeichert sein, bevor die nächste Datei geöffnet werden kann. 2. Der Datenpuffer zwischen der Task-Evaluierung , die die Daten erzeugt, und dem asynchronen Thread, das die Daten in die Datei schreibt,ist auf 1 MB begrenzt. Bit2...32: nicht verwendet,reserviert für zukünftige Anwendungen;
pp_command	STRING	Dieses Post-Processing Kommando wird ausgeführt, wenn die Datei geschlossen wurde, mindestens ein Wert gespeichert ist und die Funktion aktiviert ist.
pp_enab	BOOL	Aktiviert die Post-Processing-Funktion, wenn TRUE
sign_file	BOOL	Wenn TRUE, dann wird die Datei signiert, um erweiterte Funktionen im ibaAnalyzer nutzen zu können.
chanx_data	untyped	Messwerteingang je Kanal (x = 0...15)
chanx_info	untyped	Zusatzinfo je Kanal (x = 0...15)
chanx_enab	untyped	Aktivierung je Kanal (x = 0...15)
casc_next	DWORD	Nicht verwendet, reserviert für zukünftige Anwendungen;

Forts. DatFileWrite

Konnektor	Datentyp	Bedeutung / Verwendung
sum_values_stored	DINT	Summe der in der aktuellen Datei gespeicherten Werte eines Kanals. Mit jeder neuen Datei wird der Wert auf 0 zurückgesetzt.
file_is_open	BOOL	Ist TRUE, wenn die Datei geöffnet ist. Nur dann können Daten gespeichert werden.
last_error_code	DWORD	Fehleranzeige (Code)
last_error_string	STRING	Fehlermeldung (Text)
file_is_signed	BOOL	Dieser Ausgang wird TRUE, wenn eine Datei signiert und geschlossen wurde. Er wird FALSE, wenn eine neue Datei geöffnet wird.
chanx_par_ok	BOOL	Eingangsparameterstatus für Messkanäle (x = 0...15) Wenn die Datei geöffnet ist, werden die Eingangsparameter (Daten, Datei-Info und Aktivierungsbit) auf Datentypen und Anzahl der Einträge kontrolliert. Wenn dabei kein Fehler festgestellt wird und die Kanäle für das Speichern freigegeben sind, dann wird dieser Ausgang TRUE.
chanx_error_string	STRING	Wenn die vorgenannte Überprüfung einen Fehler feststellt, wird hier die Fehlermeldung je Messkanal (x = 0...15) ausgegeben.
chanx_mod_chan_no	STRING	Dieser Ausgang gibt für jeden Kanal (x = 0...15) an, unter welcher Modul- und Kanalnummer das Signal im dat-File zu finden ist.

Forts. DatFileWrite

Verwendung des Bausteins

Nachdem der DatFileWrite-Funktionsbaustein im Plan platziert wurde, ist zunächst die Erfassungszeit (sampling time) einzutragen und zu entscheiden, ob Einzelwerte oder gepufferte Werte gespeichert werden sollen. Bei gepufferten Werten nicht vergessen, die Anzahl der Eingangswerte ("num_values"-Eingang) anzugeben.

Anschließend müssen die zu speichernden Signale mit den Eingängen "chanx_data" verbunden werden. Für jeden Kanal, der gespeichert werden soll, muss der Eingang "enable" auf TRUE gesetzt werden, entweder mit einem einzelnen boolschen Signal oder einem entsprechenden ARRAY.

Im nächsten Schritt ist der Dateiname des dat-Files zu spezifieren.

4

Um die Daten zu speichern, wird zunächst die Datei geöffnet und anschließend eine Überprüfung der Eingangskanäle vorgenommen. Um dies zu veranlassen, muss der Eingang "store_file" auf TRUE gesetzt werden. Wenn die Überprüfung bei einem oder mehreren Kanälen nicht erfolgreich ist, dann wird jeweils der entsprechende "par_ok"-Ausgang nicht TRUE gesetzt, und eine Fehlermeldung wird generiert.

Mit Hilfe des "showString"-Bausteins kann die Fehlermeldung angezeigt werden, um den Grund zu erkennen und das Problem zu beseitigen, damit der "file_open"-Ausgang schließlich auf TRUE gesetzt wird.

Sind Eingang "store_file" und Eingang "store_values" ständig TRUE, dann werden Daten gespeichert.

Wenn alle Daten in der Datei gespeichert sind, bzw. wenn die Speicherung beendet werden soll, dann ist der Eingang "store_file" auf FALSE zu setzen.

Schließlich wird die Datei geschlossen und signiert, sofern vorgewählt.

Wenn die Postprocessing-Option gewählt wurde, dann wird das entsprechende Kommando ausgeführt.

Regeln für überladbare Eingangsanschlüsse je Kanal

- ***Chanx_data -***
  - Skalare Datentypen INT, REAL, oder BOOL, wenn Einzelwerte verwendet werden.
  - Eindimensionales ARRAY der Datentypen INT, REAL or BOOL. Wenn es sich um Einzelwerte handelt, dann entspricht jeder Index der ersten Dimension des Arrays einem Signal (=Kanal). Wenn gepufferte Werte verwendet werden, dann entspricht jeder Index des Arrays einem Sample desselben Signals (resp. Kanals).
  - Zweidimensionales ARRAY der Datentypen INT, REAL or BOOL, nur wenn es sich um gepufferte Werte handelt. Dabei entspricht jeder Index der ersten Dimension einem Signal und jeder Index der zweiten Dimension einem Sample des jeweiligen Signals.
- ***Chanx_info - optional***
  - STRING; dieser String wird für jedes Signal (=Kanal) verwendet, um die Info-Einträge in das dat-File zu schreiben. Es handelt sich dabei um ein Array der gleichen Dimension wie das Datenarray eine beliebigen Typs. (Strings können dort verborgen sein, da ein ARRAY of STRINGS nicht möglich ist. Die Anzahl der Einträge in der ersten Dimension muss der des Datenarrays entsprechen.

Forts. DatFileWrite

- ***Chanx_enab -***
  - BOOL; dieser Merker wird für jedes Signal benötigt, um das Speichern zu ermöglichen.
  - Eindimensionales ARRAY of BOOL; kann sowohl mit Einzelwerten als auch mit gepufferten Werten verwendet werden. Die Anzahl der Signale, die mit diesem Array aktiviert werden kann, muss der Anzahl der Eingangssignale entsprechen.

Besondere Anmerkungen

- Kaskadierte Ein- und Ausgaben werden noch nicht unterstützt.
- Die rechenzeit-intensiven Funktionsaufrufe zum Speichern der Daten in eine Datei belasten die Berechnung des Layouts und können im ungünstigen Fall die Layout-Berechnung gänzlich blockieren! Diese Gefahr kann durch die Aktivierung des asynchronen Zugriffs vermieden werden (Eingangskonnektor „mode_select“, Bit1 = TRUE).
- Die Sortierung der Signalkanäle im ibaAnalyzer erfolgt entsprechend 32 Analog- plus 32 Binärkanälen pro Modul. Wenn mehr als 32 Signale gespeichert werden sollen oder wenn ein Mix aus Analog- und Binärsignalen verwendet wird, dann ist unbedingt darauf zu achten, dass die Signale in der ibaAnalyzer-gerechten Art (32 Analog- + 32 Binärsignale pro Modul und in dieser Reihenfolge) angeordnet werden.
- Um den Funktionsbaustein verwenden zu können, muss das Layout online geschaltet sein und die iba Hardwarekomponenten müssen installiert sein, damit der ibaLogic-Treiber arbeitet. Der Baustein arbeitet auch im Demo-Modus und sowohl mit als auch ohne Dongle. Wenn der Baustein ohne Dongle verwendet wird, erzeugt er Messdateien ohne Signatur, d.h., dass diese Dateien dann nur mit ibaAnalyzer betrachtet aber nicht analysiert werden können.

Wird ibaLogic mit Dongle benutzt, dann erzeugt der Baustein vollwertige dat-Dateien mit Signatur.

Wird ibaLogic ohne Dongle im eCon-Modus benutzt, dann werden *.dat-Dateien ohne Signatur erzeugt, und die Daten können mit ibaAnalyzer nur betrachtet aber nicht in vollem Umfang analysiert werden.

Regeln für Texteinträge in das dat-File

Jeder Texteintrag in ein dat-File folgt der Regel <Eintragsname>:<beliebiger Text>. Texteinträge können für die Datei oder für jedes einzelne Signal gemacht werden. Die Texte werden im ibaAnalyzer verarbeitet und angezeigt. Der Funktionsbaustein erlaubt es, mehrere Einträge, die mit einem Komma getrennt sind, einzugeben. Einige Einträge haben bereits festgelegte Bedeutungen in dem dat-File. Das Beschreiben bestimmter, vitaler Einträge wird von dem Funktionsblock unterbunden. Andere Einträge werden vom Funktionsbaustein selbst nur beschrieben, wenn keine Auswahl vom Anwender erfolgt ist.

Forts.
DatFileWrite

Liste der Globalen Header Text-Einträge (unvollständig)

Eintragsname	Bedeutung	Klasse	von Logic	von Anwender
beginheader	Beginn des Headers	vital	Ja	Nein
starttime	Startzeit der Datei	vital	Ja	Nein
clk	Erfassungsraster	vital	Ja	Nein
frames	Anzahl der Werte	vital	Ja	Nein
typ	Dateiart	vital	Ja	Nein
ibalogic	ID des Erzeugers	optional	Ja	Nein
technostring	Technostring-Information	optional	Ja	Nein
endheader	Ende des Headers	vital	Ja	Nein
module_name_x	Name des Moduls	optional	Nein	optional

4

Liste der Channel Header Text-Einträge (unvollständig)

Eintragsname	Bedeutung	Klasse	von Logic	von Anwender
beginchannel	Beginn des Headers	vital	Ja	Nein
channel_offset	Offset des Kanals	vital	Ja	Nein
digchannel	Digitale Kanalinfo	vital	Ja	Nein
name	Name des Kanals	vital	Ja	optional
minscale	Oberer Skalenendwert	vital	Ja	optional
maxscale	Unterer Skalenendwert	vital	Ja	optional
endchannel	Ende des Headers	vital	Ja	Nein

Zum Speichern von zusätzlichen, anwendungsspezifischen Informationen kann man wie folgt verfahren:

Hinzufügen eines weiteren Strings im Eingangskonnektor-String, z.B. <my-entry>:<mytext>. Mehr als ein Eintrag müssen mit ',' (Komma) getrennt werden.

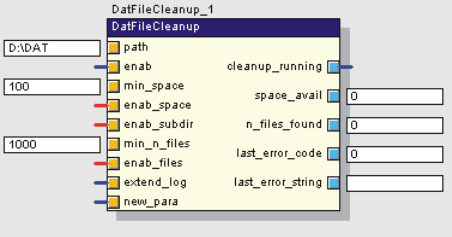
Applikationsbeispiel (Layout) auf CD

sample_layout_DatFileWrite_301.lyt

Diese Beispielanwendung ist geeignet, sich mit der Funktionsweise des DatFileWrite-Funktionsbausteins vertraut zu machen. Für die Beschaltung des Bausteins stehen Hilfsmittel zur Verfügung.

Das Beispiel zeigt das Erzeugen einer Dat-Datei sowohl mit Einzelwerten (Task Sample_1) als auch mit gepufferten Werten (Task Sample_2).

4.2.9.5. DatFileCleanup-Baustein – Aufräumen der Festplatte



Funktion und Verwendung

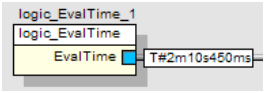
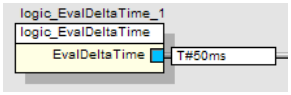
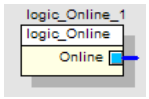
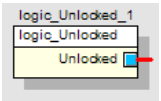
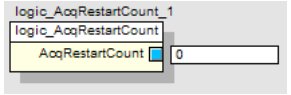
Mit diesem Funktionsbaustein ist es möglich, eine Aufräumstrategie bzgl. der Messdateien (*.dat) für die Festplatte zu organisieren. Entsprechend den Einstellungen, wie sie auch bei ibaPDA unter Triggereinstellungen / Optionen vorgenommen werden können, können alte Messdateien nach bestimmten Kriterien endgültig gelöscht werden.

Anschlüsse

Konnektor	Datentyp	Bedeutung / Verwendung
path	STRING	Speicherort für die Messdateien (= Bereich auf den sich die Aufräummaßnahmen beziehen); es müssen Laufwerksname und vollständiger Pfad angegeben werden.
enab	BOOL	Bausteinfreigabe; der Baustein wird jeweils bei einer positiven Flanke an diesem Eingang bearbeitet; wenn dieser Eingang statisch = TRUE ist, wird der Baustein alle 15 min bearbeitet.
min_space	UDINT	Angabe des Festplatten-Speicherplatzes (MB), der mindestens frei bleiben soll. Erkennt der Baustein, dass dieser Wert unterschritten und "enab_space" = TRUE ist, werden die Löschmaßnahmen eingeleitet.
enab_space	BOOL	Freigabe (= TRUE) der Überwachungsfunktion für minimalen freien Speicher s.o.
enab_subdir	BOOL	Freigabe für das Aufräumen und Löschen leerer Unterverzeichnisse. Wenn dieser Eingang =TRUE ist, dann werden nicht nur die Dateien, sondern auch die (leeren) Unterverzeichnisse nach 48 Std. gelöscht.
min_n_files	UDINT	Diese Zahl bestimmt, wieviele Dateien nach dem Löschen noch mindestens erhalten bleiben sollen. Mit dieser Einstellung kann verhindert werden, dass sämtliche Messdateien gelöscht werden. Diese Situation kann z.B. dadurch entstehen, dass andere Prozesse Daten auf die Festplatte legen und der minimal zulässige freie Speicherplatz dadurch ständig unterschritten wird.
enab_files	BOOL	Freigabe (= TRUE) der Überwachungsfunktion für minimale Anzahl Dateien s.o.
extend_log	BOOL	Freigabe (= TRUE) für die Erzeugung einer Protokolldatei für Ereignisse während der Aufräummaßnahmen.
new_para	BOOL	Freigabe (= TRUE) zur Übernahme neuer Parameter.
cleanup_running	BOOL	Aufräummaßnahmen sind gerade im Gange.
space_avail	UDINT	Freier Speicher (MB) beim letzten Aufräumen
n_files_found	UDINT	Anzahl der gefundenen Dateien
last_error_code	DINT	Fehlercode des letzten Fehlers
last_error_string	STRING	letzte Fehlermeldung

4.3 Globale Variablen

Das Konzept von ibaLogic geht von gekapselten Datenstrukturen aus. Globale Variablen sind daher im Gegensatz zu sonstigen Steuerungs- und Regelungsanwendungen hier die Ausnahme. Es existieren jedoch einige projektübergreifende Systemvariablen, die innerhalb des Funktionsplanes bzw. von Structured Text und/oder C++ Konstrukten benutzt werden können.

Nr.	Variablenname Layout-Symbol	Ziel-Typ	Beschreibung
1	**logic_EvalTime** 	TIME	= Zeit, die seit dem Start der Anwendung vergangen ist;
2	**logic_EvalDeltaTime** 	TIME	= Zeit, die seit dem letzten Taskstart vergangen ist (Abtastzeit); bei Verwendung dieser Variable werden ± Abweichungen der Abtastzeit korrigiert.
3	**logic_Online** 	BOOL	= Status, ob Layout Online ist; damit können bestimmte Funktionen oder das exklusive Verwenden von Ressourcen in Zusammenhang mit Hot Swap verriegelt werden. TRUE: Layout ist online, Ausgänge sind aktiv. FALSE: Layout ist offline, Ausgänge sind gesperrt, Eingänge sind weiter aktiv.
4	**logic_Unlocked** 	BOOL	= Status, ob Layout gesperrt wurde; dann dürfen im Layout auch keine Default-Werte verändert werden. Einsatz sinnvoll, z.B. bei der Programmierung von DLLs, die beispielsweise ihre Default-Werte von 'innen' nach 'aussen' setzen und damit das Layout verändern könnten. TRUE: Layout ist nicht gesperrt, Veränderungen können erfolgen FALSE: Layout ist gesperrt, Veränderungen sind nicht möglich
5	**logic_AcqRestartCount** 	UDINT	= Zählwert mit Anzahl der Treiberstarts seit Beginn der Berechnung. Wird dazu benutzt, dem Layout mitzuteilen, dass der Treiber bzw. die Hardware zurückgesetzt wurde, um ihm zu ermöglichen bei bestimmten Messaufbauten die Hardwareparameter wieder in den gewünschten Zustand zu bringen.

4.4 Global FBs and Macros

Globale FBs und Makros sind dann sinnvoll zu verwenden, wenn mehrere ibaLogic-Systeme auf diese übergeordneten Funktionen zugreifen sollen, weil in den verschiedenen Anlagen die gleichen Funktionen benötigt werden.

Wenn projektübergreifende Bausteine dieser Art vom Anwender zunächst als lokale FBs/Makros erstellt und exportiert wurden, dann müssen sie anschl. im Windows-Explorer vom Verzeichnis „...configuration\FBs_Macros“ in das Verzeichnis „...configuration\globalRessource\FBs_Macros“ kopiert, bzw. verschoben werden werden.

- *Die gleichen Bausteine sollten auf KEINEN Fall im lokalen Verzeichnis „...configuration\FBs_Macros“ vorhanden sein, da sie in diesem Fall nur einmal als globale Bausteine angezeigt werden.*
- *Nach dem Löschen oder Hineinkopieren von Bausteinen in das Verzeichnis „...configuration\globalRessource\FBs_Macros“ muss ibaLogic neu gestartet werden, um die Anzeige in ibaLogic zu aktualisieren!*
- *Löschen von FBs/MBs ist NUR im Windows-Explorer möglich!*
- *Wenn der Inhalt eines Bausteins nachträglich verändert wird, muss dieser auch wieder zunächst als local FB/MB exportiert und anschl. mit dem Windows-Explorer in das globale Verzeichnis kopiert werden!*

4.5 Global DLLs

Projektübergreifende DLLs, die vom Anwender z.B. mit C erstellt wurden, sind dann sinnvoll, wenn die gleiche Funktionalität einer DLL in verschiedenen Anlagen gebraucht wird.

Die globale DLL wird ibaLogic zur Verfügung gestellt, indem sie mit dem Windows-Explorer in das Verzeichnis „...configuration\globalRessource\DLLs“ kopiert wird.

- *Die gleichen DLLs sollten auf KEINEN Fall im lokalen Verzeichnis „...configuration\DLLs“ vorhanden sein, da sie nur einmal als globale DLL angezeigt werden.*
- *Nach dem Löschen oder Hineinkopieren von DLLs in das Verzeichnis „...configuration\globalRessource\DLLs“ muss ibaLogic neu gestartet werden, um die Anzeige in ibaLogic zu aktualisieren!*
- *Löschen von DLLs ist NUR im Windows-Explorer möglich!*

4.6 Local FBs and Macros

Lokale FBs und Makros kommen dann zur Anwendung, wenn die gleiche Funktionalität eines Bausteins in der selben Anlage mehrmals gebraucht wird.

Wenn vom Anwender projektspezifische Bausteine dieser Art erstellt wurden, dann müssen sie anschließend exportiert werden. Dazu mit der Maus auf den betreffenden Baustein zeigen, rechte Maustaste ↪*Ändern* ↪*Makroblock* / *Funktionsblock* ↪*Exportieren*. Der FB oder Makroblock (MB) liegt dann als Datei (*.fbm) in dem Verzeichnis „...configuration\FBs_Macros“.

Wenn bereits bei anderen Anwendungen FBs oder MBs erstellt wurden und als Datei vorliegen, können diese einfach mit dem Windows-Explorer in das Verzeichnis „...configuration\FBs_Macros“ kopiert werden.

4

- *Die gleichen MBs/FBs sollten auf KEINEN Fall im globalen Verzeichnis „...configuration\GlobalResource\FBs_Macros“ vorhanden sein, da sie sonst nur einmal im globalen Verzeichnis angezeigt werden.*
- *Nach dem Löschen oder Hineinkopieren von FBs/MBs in das Verzeichnis „...configuration\FBs_Macros“ mittels Windows-Explorer muss ibaLogic neu gestartet werden, um die Anzeige in ibaLogic zu aktualisieren!*
- *Löschen von FBs/MBs ist NUR im Windows-Explorer möglich!*
- *Wenn der Inhalt eines Bausteins nachträglich verändert wird, muss dieser nochmals als local FB/MB exportiert werden!*

4.7 Local DLLs

Lokale DLLs kommen dann zur Anwendung, wenn die gleiche Funktionalität einer DLL in der selben Anlage mehrmals gebraucht wird.

Wenn vom Anwender projektspezifische DLLs z.B. mit C erstellt wurden, dann müssen sie ibaLogic zur Verfügung gestellt werden.

Bei laufendem ibaLogic erfolgt dies über das Menü ↪*Datei* ↪*DLL öffnen...*, wo mit Hilfe des Datei-Browsers die DLL-Datei gesucht und geöffnet werden kann. Mit dem Öffnen wird die DLL auch in das Verzeichnis „...configuration\DLLs“ kopiert.

Die DLL kann auch mit dem Windows-Explorer in das Verzeichnis „...configuration\DLLs“ kopiert werden. Dann steht steht Sie ibaLogic aber erst nach einem Neustart zur Verfügung.

- *Die gleichen DLLs sollten auf KEINEN Fall im globalen Verzeichnis „...configuration\GlobalResource\DLLs“ vorhanden sein, da sie sonst nur einmal im globalen Verzeichnis angezeigt werden.*
- *Nach dem Löschen oder Hineinkopieren von DLLs in das Verzeichnis „...configuration\DLLs“ mittels Windows-Explorer muss ibaLogic neu gestartet werden, um die Anzeige in ibaLogic zu aktualisieren!*

Löschen von DLLs ist NUR im Windows-Explorer möglich!

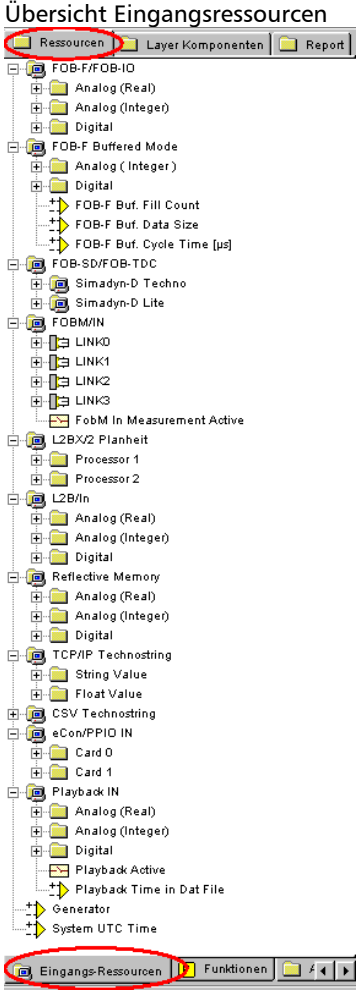
5 Prozessankopplung

Die I/O-Prozessankopplung und offene Kommunikation zur Außenwelt erfolgt bei ibaLogic mittels vordefinierter, einfach verschaltbarer Eingangs- und Ausgangsressourcen. Im Ressourcenbereich werden die verfügbaren Ressourcen grafisch dargestellt. Über die Ressourcenauswahl kann zwischen der Darstellung der Eingangs- und Ausgangs-Ressourcen umgeschaltet werden.

5.1 Eingangsressourcen

Die Eingangsressourcen wurden in die folgenden Bereiche unterteilt:

Übersicht Eingangsressourcen



- FOB-F oder FOB-IO-Karte

Standardisierte analoge und digitale Eingänge, gruppiert in 32 Module mit je 32 Eingängen (1024 max). Einkopplung über LWL von
1) PADU- (Parallel Analog Digital Units)
2) ibaNet750 (WAGO) Remote-I/O-Klemmen oder
3) SM64 / SM128V-Karten.
Bei Verwendung einer PCMCIA-F-Karte werden die ersten beiden Module benutzt.

- FOB-F Buffered Mode

Diese Eingaben beziehen sich auf die ersten acht Module einer FOB-F-Karte.
Ein vordefinierter Satz von Eingangsvariablen, die gepufferte Messwerte von FOB-F-Karten verarbeiten (z.B. für FFT-Anwendungen). Maximale Puffertiefe: 256 Werte für bis zu acht Module mit je 32 Kanälen (8*32=256 Kanäle)

- FOB-SD / FOB-TDC-Karte

Vollautomatischer Koppelpartner zu SIMADYN-D Automatisierungsgeräten (CS12/13/14), bzw. Simtaic-TDC (GDM). Unterstützt passiven- und Anforderungs-Modus.
1) SIMADYN-D Techno; vordefinierter TechnoString.
2) SIMADYN-D Lite; vordef. Satz von Eingangsvariablen FOB-M/IN
Vordefinierter Satz von Eingangsvariablen für FOB-M Anschaltungen und Padu8 ICP (25KHz) Messsystem. (Vibrationsmessungen)

- L2BX/2 Planheit

Vordefinierter Satz von Eingangsvariablen für Planheitsmessung. Anschluss über Profibus L2Bx-F, L2B x/8 PCI.

- L2B/In

Standardisierte analoge und digitale Eingaben; 32 Gruppen (Module) mit je 32 Eingaben (max. 1024). Anschluss über Profibus von
1) S7 (nur 28 Real-Werte je Modul aufgrund von S7-Beschränkungen)
2) jedem anderen Profibus-Master

- Reflective Memory

Vordefinierter Satz von Eingangsvariablen für eine Reflective Memory-Verbindung. Analoge (Integer oder Real) und digitale Eingänge, gruppiert in je 32 Module mit je 32 Eingängen (1024 max). Für die Kopplung ist eine spezielle Hardware erforderlich (PC-Steckkarte von VMIC).

Forsetzung nächste Seite

Forsetzung Eingangsressourcen

- TCP/IP TechnoString

TCP/IP-Eingangsvariablen, gruppiert in 16 String- und 96 Float Variable, zuzuordnen über Dialogfenster ↪*TechnoString* ↪*TCP/IP*.

- CSV TechnoString

TCP/IP-Eingangsvariablen, gruppiert in 128 String-Variablen beliebigen Inhalts. Die einzelnen Variablen des CSV-Strings (**C**omma **S**eparated **V**alue) werden mitttels Komma voneinander getrennt.

- eCon/PPIO IN

Vordefinierter Satz von 32 Eingangsvariablen von einer parallelen Druckerschnittstelle des PCs (LPTx).

- PlaybackIn

Vordefinierter Satz von 32 Modulen mit je 32 Analogeingaben (Integer oder Real) und 32 Digitaleingaben zur Verarbeitung von Signalen aus Messdateien (*.dat) im Playback-Modus.

- Generator

Einfach parametrierbarer Signalgenerator für Sinus, Rechteck, Triangel und mit frei definierbarem Signalausgang.

- System UTC Time

Systemuhrzeit verschaltbar an Zeitfunktionen zur Anzeige und Auswertung

5

5.1.1. FOB-F, FOB-IO oder FOB 4i-Eingangsressourcen

Die FOB-F / FOB-IO Eingangsressourcen sind unterteilt in:

- ❑ Analog (Real) Module 1..32 oder alternativ
- ❑ Analog (Integer) Module 1..32 und
- ❑ Digital Module 1..32

Jedes Modul verfügt über 32 Eingänge, d.h. maximal sind 32 * 32 = 1024 analoge und 1024 binäre Eingangswerte verschaltbar.
Jeder optische Anschluss einer FOB-F,- FOB-IO- oder FOB-4i-Karte ist verschaltet zu zwei Modulen mit jeweils 32 Eingängen, d.h. insgesamt 64 analogen und 64 binären Eingängen.

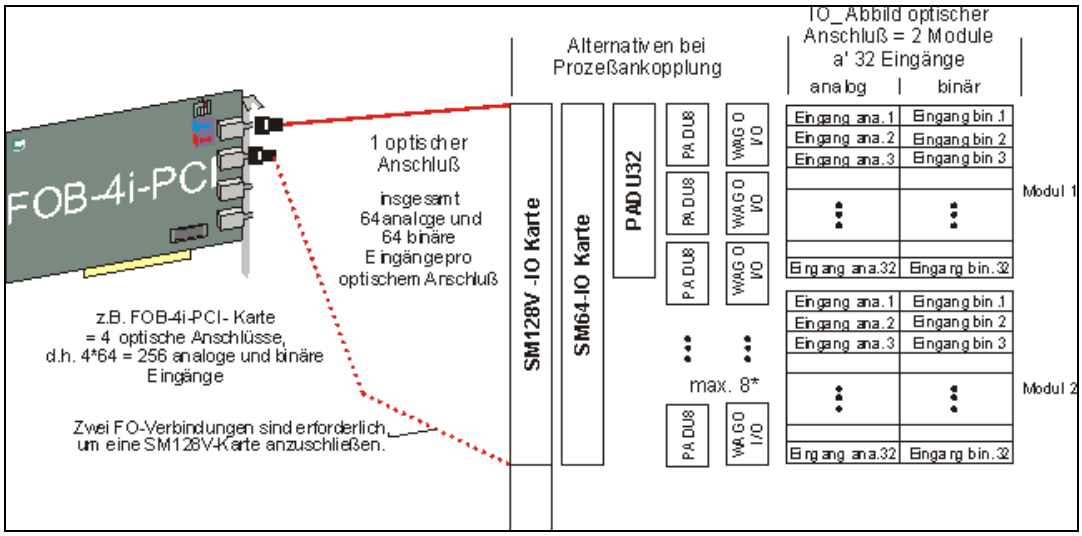


Bild 70 FOB-IO PCI-Karte und Lichtleiterverbindungen

Ein optischer Anschluss kann alternativ verschaltet werden mit:

- ❑ einer SM 64-IO-Baugruppe (64 analoge und 64 binäre Signale)
- ❑ zwei PADU 32 Einheiten (2*32 = 64 analoge und 64 binäre Signale)
- ❑ acht PADU8-Einheiten (8*8 = 64 analoge und 64 binäre Signale)
- ❑ acht WAGO-Köpfen (8*8 = 64 analoge und 64 binäre Signale)

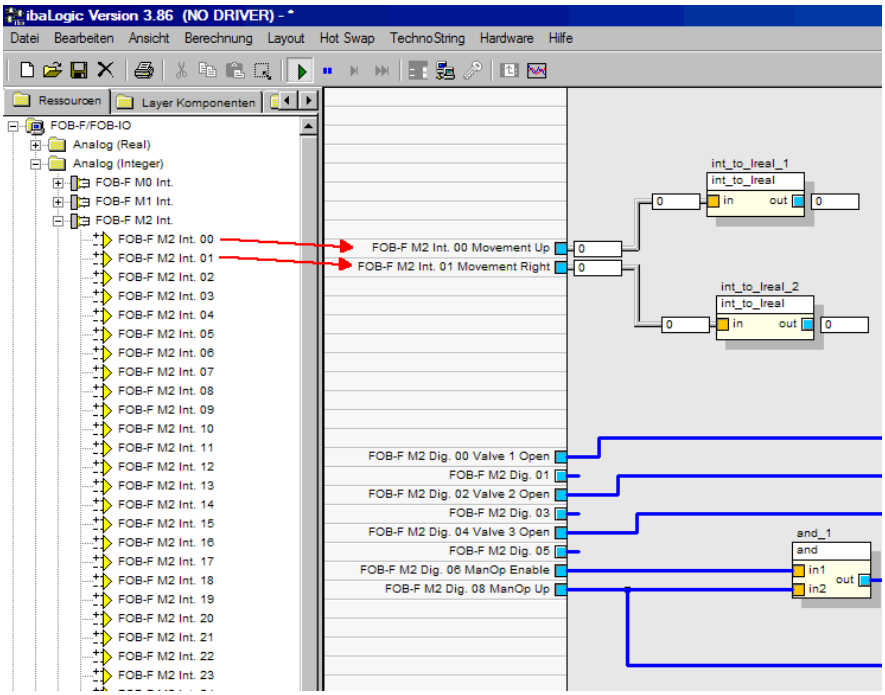


Bild 71 Platzierung von FOB-F / FOB-IO Eingangsressourcen im Layout

Das Beispiel zeigt die Verschaltung von analogen und binären FOB-F / FOB-IO Eingangsressourcen mit ibaLogic.

Es müssen nicht alle Ressourcen eines Moduls in einer ibaLogic-Task verschaltet werden, sondern jedes benötigte Signal kann einzeln selektiert und auf die Ein- oder Ausgangsrandleiste verschoben werden.

Bei Bedarf können aber auch sämtliche 32 Ein- bzw. Ausgänge eines Moduls zusammenhängend auf die Ein-/Ausgangsleiste plaziert werden. Dazu das entsprechende Modul (z.B. Module 2) selektieren, auf die Ein-/ Ausgangsleiste verschieben und die anschl. Frage "Splitten des Arrays in Einzelsignale" mit "Ja" beantworten.

5.1.2. FOB-F Buffered Mode

Die Gruppe der Eingangsressourcen für FOB-F Buffered Mode wurde geschaffen, um auch Signale verarbeiten zu können, deren Datenerfassungsrate in der FOB-F-Karte wesentlich höher ist, als die Verarbeitungsgeschwindigkeit durch eine Task.

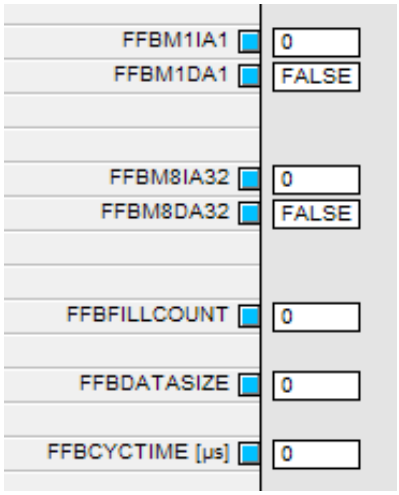
Bei einer Erfassungsrate von 1 ms, einer Tasklaufzeit von 50 ms und einer erforderlichen Anzahl von Messwerten für die FFT von 128 pro Signal können so trotzdem alle Messwerte verabeitet werden.

Ermöglicht wird dies durch einen speziellen Messmodus, in dem das ibaLogic-Laufzeitsystem Daten puffert und die gepufferten Messwerte als Arrays mit der maximalen Tiefe von 256 Werten zur Verfügung stellt. Damit kein Messwert verloren geht, muss die Leserate der Task, bzw. des ibaLogic-Layouts höher sein als die Füllrate der Arrays.

Sinnvoll anzuwenden ist dieser Messmodus nur in der ibaLogic Betriebsart "SignalManager-Modus".

5

Es gibt aber auch Anwendungen, die weniger als 256 Messwerte benötigen, oder bei denen nicht immer oder nicht alle gepufferte Werte benötigen werden. Dafür wurde ein spezielles Kommunikationsinterface zwischen der Task und der ibaLogic-Laufzeitumgebung geschaffen, das der Task folgende Eingaben zur Verfügung stellt:



8 Module mit je 32 Analogeingaben (Integer)

8 Module mit je 32 Digitaleingaben (Bool)

Fillcount ist ein Zählwert, der um 1 erhöht wird, wenn der Puffer gefüllt wurde und neue gepufferte Daten an die Task übergeben wurden.

Datasize ist die Anzahl der Messwerte die tatsächlich in den Puffer eingetragen wurden.

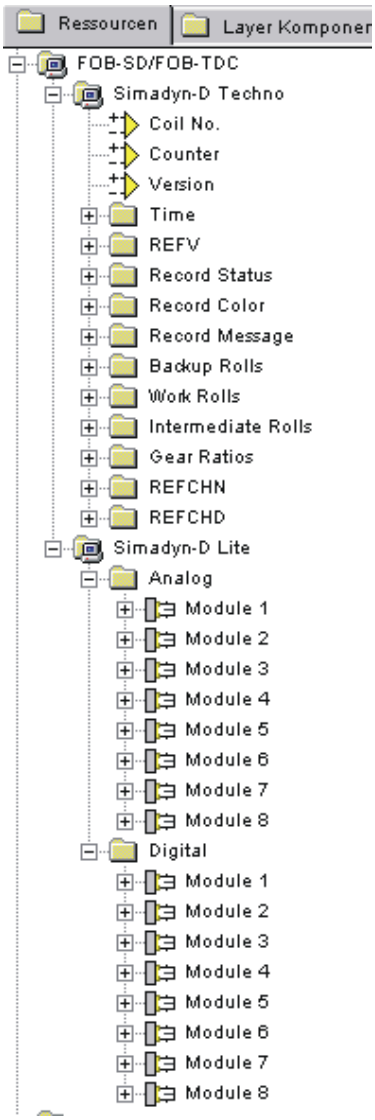
Cyctime ist die benutzte Zykluszeit am Lichtwellenleiter-Anschluss. Diese Eingabe ist für den so genannten Asynchronmodus relevant.

Bild 72 Eingangsressourcen für FOB-F Buffered Mode

5.1.3. Signale von Simadyn-D und TDC (FOB-SD / FOB-TDC)

Bei der SIMADYN-D Prozessanbindung wird zwischen folgenden Signalarten unterschieden:

- ❑ SIMADYN-D Techno (kurz für Technostring)
- ❑ SIMADYN-D Lite (16 Module mit jeweils 32 analogen (real) und 32 binären Signalen)



SIMADYN-D TechnoString (für FOB-SD und FOB-TDC)

Der Simadyn-D TechnoString beinhaltet alle Daten, die zur Konfiguration und Versorgung eines QDA-Systems für eine 7-gerüstige Tandemstraße benötigt werden (mit FFT-Einstellungen, Gerüstparameter, Rollendurchmesser usw.). Die Datenstruktur ist fest und kann nicht verändert werden. Die Daten werden über FOB-SD oder FOB-TDC-Kopplung ausgetauscht.

Das angeschlossene Simadyn-D-Gerät muss den Kanalnamen Q1DAT mit der Länge 512 Byte (Typ Refresh) eingerichtet haben. (siehe ergänzende SIMADYN-D Dokumentation).

Achtung: Der Q2DAT-Kanal wird nicht länger benötigt, er wurde durch den Kanal MxPDADAT ersetzt (siehe nächstes Kapitel)
Q1DAT_AcqLength = 512 // TechnoString Kanal mit 512 Bytes
Q2DAT_AcqLength = 0 // nicht mehr benötigter Kanal

SIMADYN-D Lite (für FOB-SD / FOB-TDC)

Diese Prozessankopplung ist ähnlich der FOB/ FOB-IO Eingangsressourcen strukturiert, d.h. für die (Real) Analog- und Binärwerte sind jeweils 8 Module mit 32 Kanälen vorgesehen. Jedes Modul kann (muss aber nicht) von einer SIMADYN-D CPU versorgt werden.

Achtung:
In den Simadyn-D Automatisierungsgeräten / Simatic-TDC Automatisierungsgeräten müssen Refresh-Kanäle "M0PDADAT" bis "M7PDADAT" mit einer Länge von jeweils 132 Bytes definiert werden. Jeder Kanal repräsentiert ein Modul
Einige zusätzliche Informationen, insbesondere bezüglich der Kanalbezeichner in Simadyn-D sind für eine einwandfreie Kommunikation zwischen Simadyn-D und ibaLogic notwendig (siehe ergänzende SIMADYN-D Dokumenta-tion).

Für die Einstellungen auf den FOB-SD bzw. FOB-TDC gibt es einen speziellen Dialog im Menü *↪Ansicht ↪PCI-Konfiguration ↪FOB-SD/TDC Link Einstellungen*

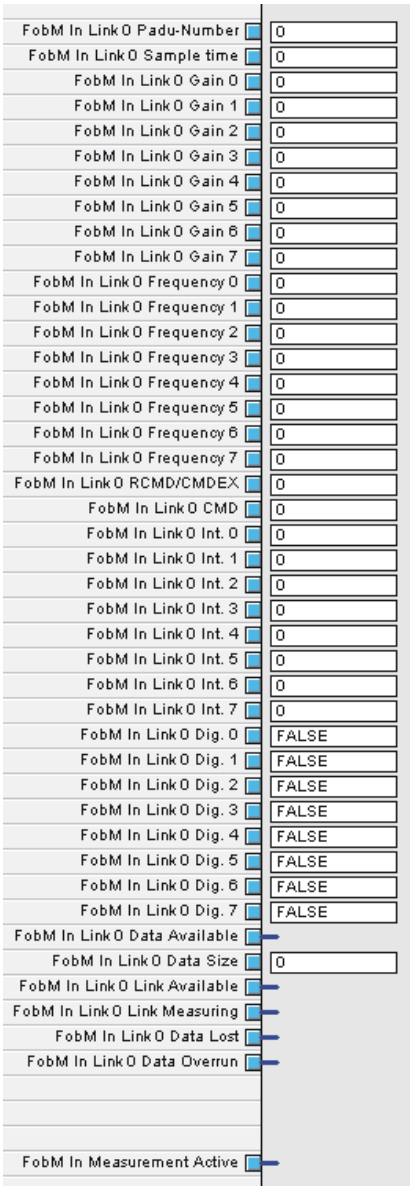
Bitte überprüfen Sie die ibaLogic Konfigurations-Datei bezüglich der korrekten Parametereinstellung:

```
FOBSX_AcqAddress = 0xE0000          // FOB Simadyn-D-Basis Adresse
CS22_BgtName = PDA001               // Name Simadyn-D-BGT, korrekte ID in STRUC Plänen ablesen
CS22_AcqAddress = 0xD0000           // immer !!
Simadyn_Sync_Timeout = 15           // Timeout, hier auf 15 sec. eingestellt
Simadyn_Proc_Timeout = 15           //
CS22_0_OwnName = DPDA1A             // Frei einstellbarer Name für den "PC"
CS22_0_Partner = D1700B             // Adresse für den Koppelpartner, Dxx00B, wobei xx angibt,
CS22_0_SoftwareVersion = V420       // wo die CS1x Baugruppe gesteckt ist, hier Slot 17
CS22_1_OwnName = DPDA2A             // CS22 mit HW-id 01 und dem Namen DPDA2A, gesteckt
CS22_1_Partner = D0900B             // in Slot 09 und Baugruppenträger PDA001
CS22_1_SoftwareVersion = V430
CS22_2_OwnName = DPDA3A
```

```
CS22_2_Partner = D1200B
CS22_2_SoftwareVersion = V430       // Version der Projektierungs-Software STRUC,
CS22_3_OwnName = DPDA4A             // die angeschlossene CPU wurde mit V 4.30 projektiert
CS22_3_Partner = D1500B             // V4.25 muss parametriert werden mitV4.20
CS22_3_SoftwareVersion = V430
CS22_NBoards = 1
Q1DAT_AcqLength = 512               // Anzahl aktiver CS22 Baugruppen (nicht FOB-SD´s!)
Q2DAT_AcqLength = 0                 // immer angeben bei Einsatz des "TechnoString"
M0DAT_AcqLength = 132               // immer !!
M1DAT_AcqLength = 0                 // Achtung: Alle Kanäle haben eine feste Länge und Struktur
M2DAT_AcqLength = 0                 // Kürzere Kanäle müssen mit 0 aufgefüllt werden
M3DAT_AcqLength = 0                 // Für Module mit 32 analogen- und 32 binären Eingängen
M4DAT_AcqLength =0                  // wird eine Gesamtlänge von 132 Bytes benötigt
M5DAT_AcqLength =0                  // M0DAT kommuniziert mit Modul1, M7DAT mit Module8
M6DAT_AcqLength =0
M7DAT_AcqLength =0
```

5.1.4. Eingangsressourcen FOBM/IN

Zur Maschinenüberwachung mittels Schwingungsmessung und -analyse werden die PADU8-M bzw. PADU8-ICP Analog-/Digitalwandler mit 40µs Wandlungszeit (25kHz) zusammen mit den FOB-M Anschaltungen eingesetzt. In der folgenden Tabelle ist die Projektierung des Kanal 1 (link1) des ersten FOB-M Moduls beschrieben. Bis zu vier Kanäle sind projektierbar.



Nummer der aktiven PADU8-ICP Einheit (00..96)
Abtastzeit in µs für diese PADU8-ICP Einheit

Aktuelle Verstärkungseinstellung in dB für Kanäle 0…7;
Der Umsetzer gibt die aktuelle Verstärkungseinstellung pro Kanal an.

Aktuelle Eckfrequenz für Tiefpass der Kanäle 0...7 in Hz
Der Umsetzer gibt die aktuelle Eckfrequenzen pro Kanal an.

Aktueller Zustand des "Reset"-Kommandos
Aktuelles Kommando in Bearbeitung

Analog Eingangs-Kanäle 0...7; (Integer mit Vorzeichen)

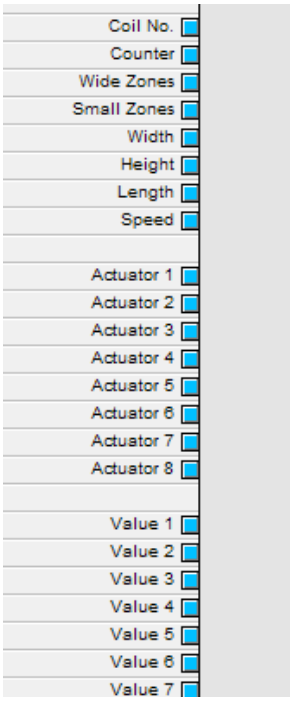
Binär-Eingangs-Kanäle 0...7

- Status Eingangspuffer; TRUE, Anzahl der Werte >= Datenbereich.
- Größe der Daten, sobald Daten verfügbar.
- Verbindungsstatus (TRUE, wenn FOB und Verbindung ok)
- TRUE, wenn Verbindungsstatus OK und PADU aktiviert zur Messung (FOMEASUREMENTSTART = TRUE)
- TRUE, wenn Daten langsamer gelesen als geschrieben werden.
- TRUE, wenn Puffer-Überlauf und Messung abgebrochen

TRUE, wenn FOB-M Messung in Bearbeitung

5.1.5. L2Bx/2 Planheit

Diese spezielle Eingangsressource wurde zur Einkopplung der Siemens-Planheitsregelung entwickelt. Die Kopplung zwischen Siemens Planheits-PC und ibaLogic erfolgt über Profibus L2-DP, wobei der Planheits-PC als Profibus-Master und ibaLogic (via L2B-Anschaltung) als Slave agiert. Zum Verbindungsaufbau müssen sowohl die Master- als auch die Slave Adressen bekannt sein. Die L2B-Anschaltung muss für einen der Planheits-Modi parametriert werden. Egal, welcher Modus gewählt wird, die entsprechenden Daten werden stets den gleichen Eingangsressourcen in ibaLogic zugeordnet.



Es stehen zwei Eingangsressourcen (Prozessor 1 und 2) zur Einkopplung von bis zu zwei Planheits-PCs zur Verfügung.

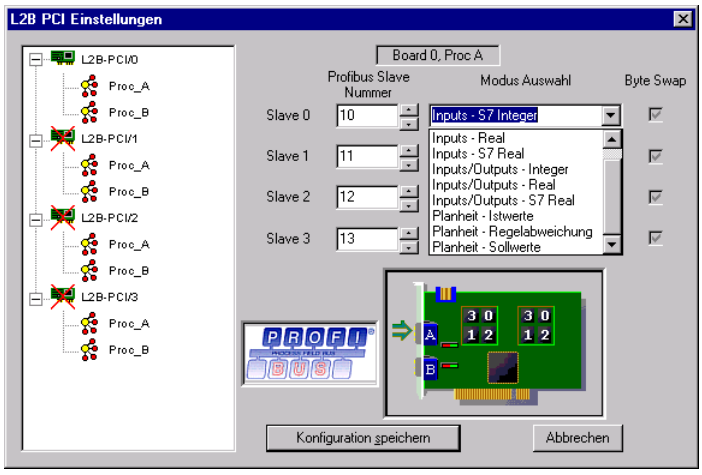
Der übertragene Datensatz beinhaltet eine Header-Struktur (Coil-Nr, Counter, Zonenbreite usw.) zur Steuerung der QDA-Anzeige.

Neben 8 Aktoren wurden bis zu 80 Zonenwerte (Values) realisiert.

ibaLogic überwacht die Profibus-Verbindung. Eine unterbrochene Verbindung wird erkannt und automatisch wieder neu aufgebaut. Im "Offline"-Zustand, d.h. bei einer Unterbrechung der DP-Verbindung, friert ibaLogic die letzten empfangenen Daten ein. Das QDA-Planheitsprofil zeigt somit keine Änderungen mehr an.

Bemerkung: Eine unterbrochene Profibus-Verbindung hat keinen Einfluss auf das Zeitverhalten von ibaLogic.

L2B - Kartenkonfiguration



Beim Aufbau der Verbindung zwischen ibaLogic und dem Zielsystem werden genau die Daten angefordert, die dem gewählten Modus entsprechen. Das Zielsystem stellt sich entsprechend darauf ein. Eine Umschaltung der Modi während des Betriebes (online) ist nicht möglich.

↗ *Siehe dazu auch Kapitel 2.6.3*

5.1.6. Reflective Memory (RM)

Die Verknüpfung der RM-Ressourcen mit der RM-Schnittstelle ist Teil der PCI-Konfiguration, wie in Kapitel 2.6.5 beschrieben.

Zu jedem der 32 RM-Eingangsmodule gehören 32 RM-Eingangssignale, deren Signalnamen dem Modul fest und eindeutig zugeordnet sind. Zusätzlich erhält jedes Signal eine Beschreibung (Klartext), die sich zugunsten eines besseren technischen Verständnisses editieren läßt.

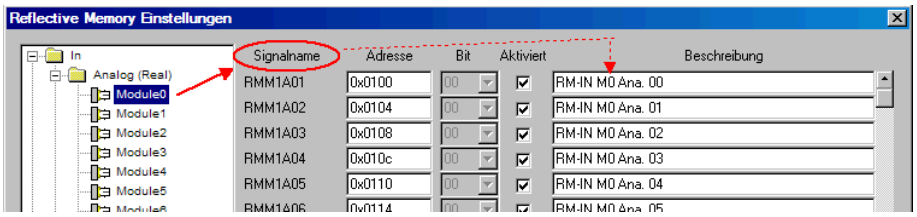


Bild 73 Reflective Memory Eingangsressourcen, Beziehung zw. Modul, Signalname und Beschreibung

Diese Beschreibungen der Eingangssignale finden sich im Signalbaum bei den Eingangsressourcen wieder und werden bei der Verwendung der Eingangssignale im Funktionsplan benutzt. Sie erscheinen auch im Tooltipp.

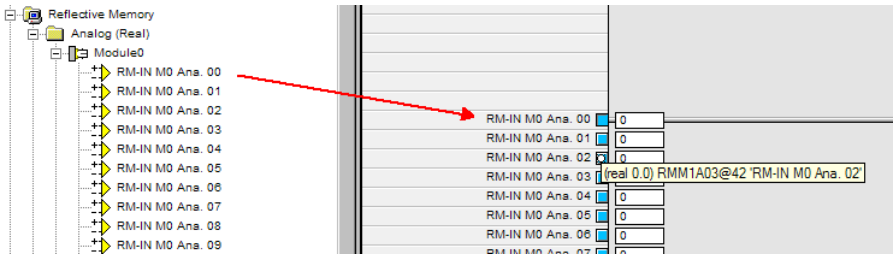


Bild 74 Reflective Memory Eingangsressourcen, Anzeige der (Signal-)Beschreibung

5.1.7. TCP/IP-Technostring

Bei der TCP/IP Technostring-Kommunikation wird eine feste Datenstruktur (mit Anzahl der Bytes und Strukturinhalt) zwischen zwei Partnern definiert. Jede Art von Daten kann übertragen werden (Gleitpunktwerte, Zeichenketten etc.). Mit Hilfe des Kommandos ↪*TechnoString* ↪*TCP/IP* der ibaLogic-Menüleiste kann ein beliebiger Teil einer empfangenen TechnoString-Zeichenkette selektiert und einer TCP/IP-String-Eingangsvariable (0...15) zugeordnet werden.

Voraussetzung für diese Funktion ist allerdings, dass die TCP/IP-Kommunikation grundsätzlich im Menü ↪*Datei* ↪*Systemeinstellungen* ↪*Sonstige* aktiviert wurde. Dort muss im Feld *TCP/IP Aktivieren* ein Häkchen sein.

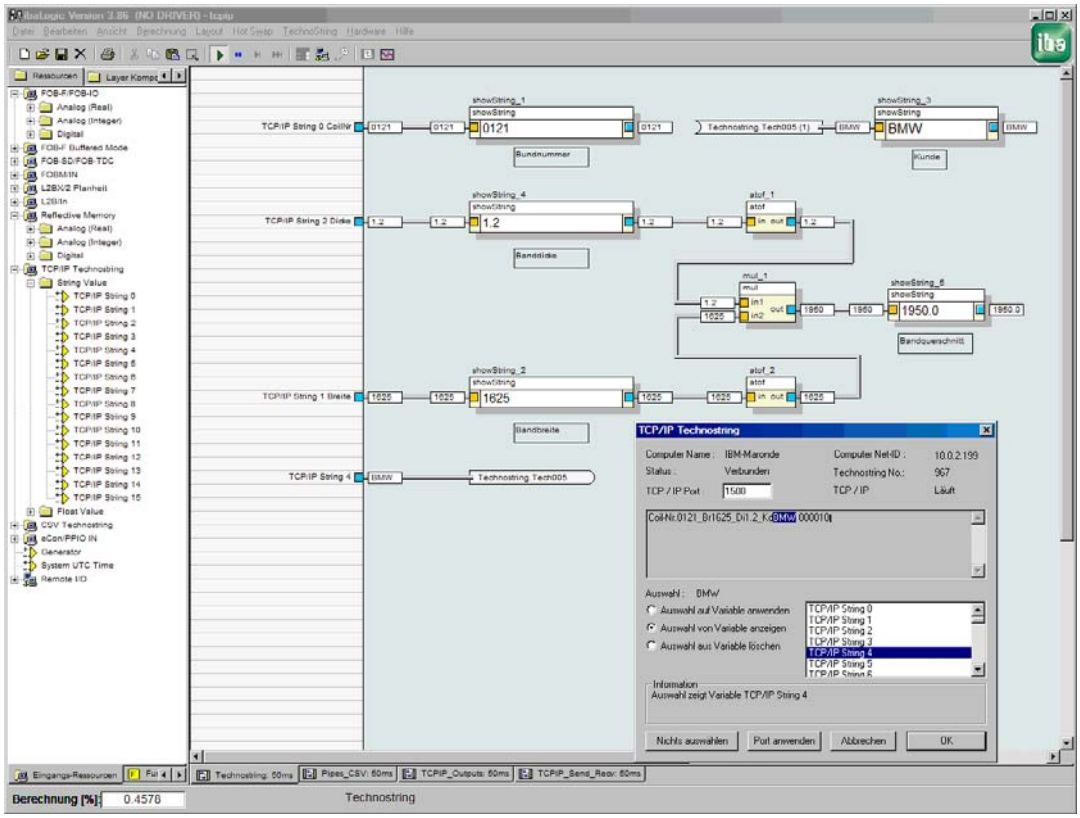


Bild 75 Beispiel: Zuordnung der Variable TCP/IP-String 4 zu Teilen des empfangenen TechnoStrings

Das obere Beispiel zeigt, wie die Variable *TCP/IP String 4* einem bestimmten Bereich innerhalb des TechnoStrings zugewiesen wird (hier die Zeichenkette "BMW").

1. In der Menüzeile ↪*TechnoString* ↪*TCP/IP* wählen.
2. In dem Feld TCP/IP Port die gleiche Port-Nummer einstellen, die auch das sendende System verwendet.
3. Von dem Quellsystem, das den Technostring erzeugt, einen Musterstring senden lassen oder mit dem iba-Hilfsprogramm *TcpIpTest...exe* einen Musterstring erstellen und an ibaLogic schicken. Der String erscheint dann in dem Dialogfenster TCP/IP Technostring.
4. Bei den Auswahloptionen *Auswahl auf Variable anwenden* markieren.
5. Mit der Maus die Zeichenfolge im Technostring markieren, die einer TCP/IP String Variable zugewiesen werden soll. (Falls das Markieren nicht gelingen sollte, bitte sicherstellen, dass in diesem Moment kein Technostring gesendet wird.)

6 In dem Feld mit der Variablenliste die gewünschte Variable (hier: TCP/IP String4) suchen und anklicken. Fertig!

In dieser Vorgehensweise können alle TCP/IP-String-Variablen den verschiedenen TechnoString-Inhalten (Teilen der Zeichenkette) zugewiesen werden.

Es ist wichtig, dass der TechnoString eine festgelegte Struktur besitzt, damit gleiche Daten stets an der gleichen Stelle stehen. Wenn im o.g. Beispiel "Coil 121" anstelle von "Coil 0121" gesendet würde, dann würden alle nachfolgenden Zeichen um eine Stelle nach vorn rutschen und TCP/IP 1 bis 4 hätten falsche Werte. Daraus folgt z.B., dass führende Nullen verwendet werden sollten!

Für den Empfang von Technostrings sind nur die o.g. Einstellungen vorzunehmen. Die Einstellungen zum Thema TCP/IP und Technostring, die im Menü ↪Datei ↪PCI-Konfiguration ↪TCP/IP Out Einstellungen *zu finden sind, haben nichts mit dem Empfang von Technostrings zu tun.*

Sie beziehen sich ausschließlich auf die Ausgabe- bzw. Senderichtung! (vergl. Kapitel 5.2.5)

5.1.8. CSV-Technostring

CSV-Technostring ist eine andere Möglichkeit, Daten an ibaLogic zu übermitteln. Die Werte werden durch Kommas getrennt, wie der Name CSV (**C**omma **S**eparated **V**alues) bereits ausdrückt. Diese Funktion ist einfacher auszuführen als die TCP/IP-TechnoString-Funktion, weil keine feste Formatierung erforderlich ist. Zeichenketten können mittels MS-EXCEL oder anderen Programmen erstellt werden, die in der Lage sind, Dateien mit dem "Komma"-Trennzeichen zu generieren.

ibaLogic empfängt die Daten als Zeichenketten (Felder) und weist diesen automatisch den CSV-String 1...128 zu. Die Datenzuweisung erfolgt in der Reihenfolge, in der die Felder innerhalb der Quelle definiert wurden.

Die Quelle muss das folgende Format haben:

< field1>,< field2>,.....,.....,< field128> < cr > < lf >

Beispiel:

Erstellen Sie eine Textdatei mit dem Namen " pipetest.txt " mit folgendem Inhalt (4 Felder):

```
CSV-Test,1234,5678,hallo < cr, lf >
```

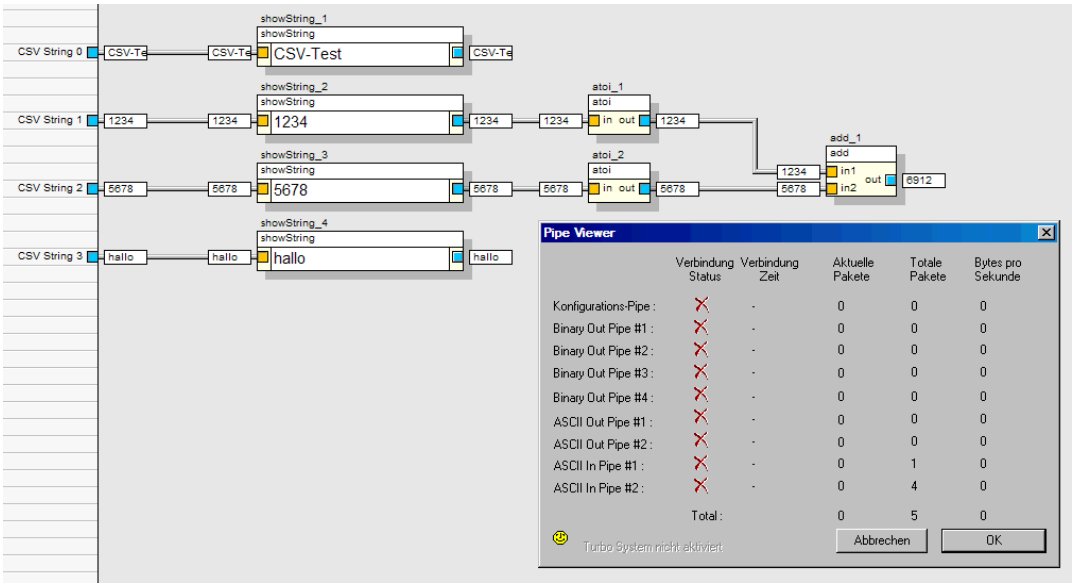
Vergessen Sie nicht, das "Carrige Return", "Line Feed" am Ende der Datei als Begrenzung hinzuzufügen!

Leiten Sie die Datei mit folgendem DOS-Befehl zum Empfänger-PC mit dem Namen "PDA":

```
copy pipetest.txt \\PDA\pipe\qda_asciiin
```

"qda_asciiin" ist das Schlüsselwort für die ibaLogic-Pipe (beachten Sie die 3 i!)

ibaLogic empfängt die Daten als Zeichenkette und stellt sie in den Eingangsvariablen "CSV String 1...128" zur Verfügung. Die Umwandlung in andere Datenformate erfolgt mit Hilfe der Konvertierungsbausteine (hier die Funktion ASCII zu Integer).



Mit dem Menükommando ↪*Ansicht* ↪*Pipes..* kann der Status der "ASCII In Pipe #1 und #2" abgefragt werden.

5.1.9. eCon/PPIO IN – Eingaben von eCon / eCon32

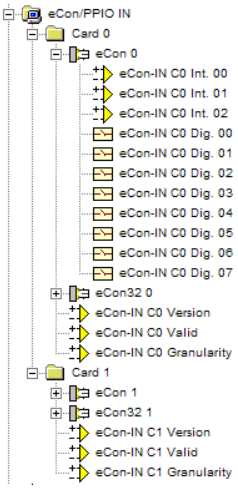
Diese Eingaberessourcen sind speziell für die Geräte eCon und eCon32 konzipiert.

Bei den eCon-Geräten handelt es sich um kleine Ein-/Ausgabegeräte, die an die Parallelschnittstelle des PCs angeschlossen werden. Es gibt sie in zwei Ausführungen:

eCon: Dieser Typ verfügt über 3 Analogeingänge, 2 Analogausgänge; 8 Digitaleingänge und 8 Digitalausgänge.

eCon32: Dieses Gerät verfügt über 32 Digitaleingänge und 32 Digitalausgänge.

Bis zu zwei Geräte können miteinander kombiniert werden. Maximal zwei solcher Geräte lassen sich an einem PC-Parallelport betreiben.



Die Zuordnung von eCon-Geräten und Eingangsressourcen ist wie folgt:

Card 0	erstes eCon an der Parallelschnittstelle
	wenn eCon, dann 3 AE und 8 DE wenn eCon32, dann 32 DE
Card 1	zweites eCon, steckt am ersten eCon
	wenn eCon, dann 3 AE und 8 DE wenn eCon32, dann 32 DE

Mit den Eingangssignalen ...Version, ...Valid und ...Granularity werden Informationen über das angeschlossene Gerät angezeigt:

Version:	Firmwareversion des Gerätes,
Valid:	Zustandsanzeige, ob Eingabewerte gültig,
Granularity:	Schrittweite in Abhängigkeit von der Auflösung der AD-Wandler. Bei 10 Bit-Auflösung beträgt die Schrittweite 64.

Weitere Informationen zu den eCon-Geräten finden Sie in der entsprechenden Hardwaredokumentation, die auch auf die ibaLogic-Programmierung eingeht.

hw_man_eCon_de.pdf

5.1.10. PlaybackIN – Eingaben für den Playback-Betrieb

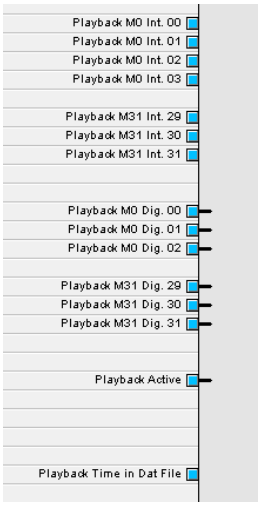
Die Eingangsressourcen PlaybackIn wurden eigens für den Playback-Betrieb mit Signalen aus Messdateien (*.dat) geschaffen. Sie können nur über den Dialog der Modulrangierung (*Menü ↪Datei ↪Programmeinstellungen ↪Playback ↪Modul Rangierung*) konfiguriert werden. (vergl. Kapitel 2.4.4 und 3.6.4)

Je nachdem, in welchem Format die Analogwerte in der Datei vorliegen, werden automatisch die Integer- oder die Real-Signale mit Werten versorgt.

Die Signalnamen werden nicht aus der Messdatei übernommen. Sie müssen ggf. von Hand angepasst werden.

Mit den 32 * 32 Kanälen können Messdateien eines voll ausgebauten PDA-Systems mit 1024 analogen und 1024 digitalen Signalen gelesen werden.

Im Mischbetrieb, d.h. mit Beteiligung der Hardware Eingaben, kann so der Playback-Betrieb gleichzeitig mit der Verarbeitung von realen Signalen realisiert werden.



32 Module mit je 32 Analogwerten (Integer oder Real)

32 Module mit je 32 Digitalwerten

Playback Active ist = TRUE, wenn in den Systemeinstellungen (Menü ↪Datei ↪Systemeinstellungen ↪Allgemein) der Playback-Modus aktiviert wurde.

Playback Time in Dat File liefert die aktuelle zeitliche Position des "Cursors" in der Messdatei. Der Wert versteht sich als relativer Zeitpunkt zum Startzeitpunkt in der Datei, Angabe in Sekunden.

5.1.11. Generator

Die Eingangsressource Generator ist ein ausgesprochen praktisches Hilfsmittel. Lassen sich doch mit ihm auf einfache Weise Testsignale verschiedener zeitlicher Verlaufsformen erzeugen.

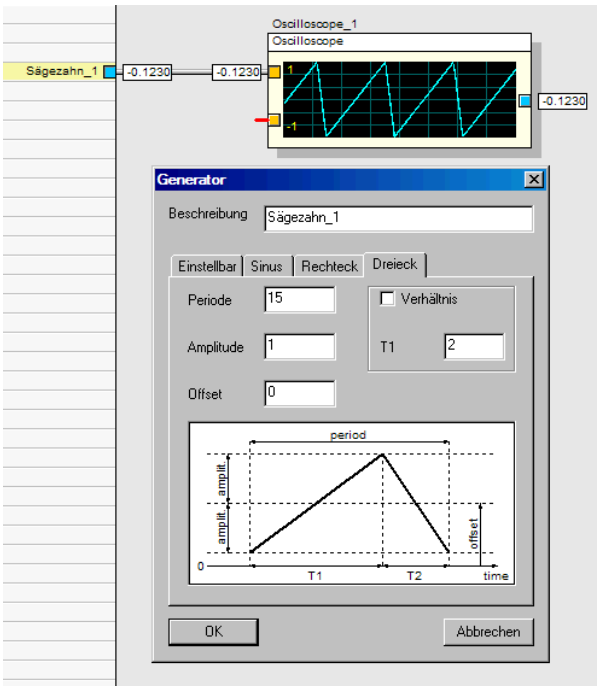


Bild 76 Eingangsressourcen, Generator

Um einen Generator zu verwenden muss einfach nur die Eingangsressource "Generator" markiert und mit der Maus auf die Eingangsrandleiste des Arbeitsbereiches gezogen werden. Es können beliebig viele Instanzen des Generators mit unterschiedlichen Signalformen gestartet werden.

Es öffnet sich das oben dargestellte Dialogfenster, in dem folgende Einstellungen vorgenommen werden können.

❑ ***Beschreibung***

Hier kann ein Klartext eingegeben werden, der den Generator, bzw. das von ihm erzeugte Signal eindeutig identifiziert. Das ist besonders bei Verwendung mehrerer Generatoren hilfreich.

❑ ***Register mit den Generatorarten***

Auf jeder Registerkarte findet sich eine grafische Darstellung mit den charakteristischen Größen der entsprechenden Signalform.

Die folgenden Parameter können bei allen Generatorarten eingestellt werden:

- *Periode*: Angabe der Periodendauer in Sekunden
- *Amplitude*: Amplitudenwert; es gibt nur einen Wert, der symmetrisch zur X-Achse berechnet wird, d.h. er gilt sowohl positiv als auch negativ.
- *Offset*: Angabe des Offsets (Lage der Nulllinie); ist ein Signalverlauf gewünscht, der nicht negativ wird, dann muss der Offset mindestens so groß wie die Amplitude gewählt werden.

Darüber hinaus gibt es für jeden Generatortyp noch weitere Parameter:

❑ ***Register Einstellbar***

Diese Generatorart erlaubt die Definition eines beliebigen periodischen Signalverlaufs. Die angegebene Periodendauer wird stets in 20 gleiche Abschnitte unter-

teilt (Index). Für jeden Index (1...20) kann ein einzelner Wert manuell eingegeben werden. Um die Arbeit etwas zu erleichtern kann zunächst ein anderes Register (Sinus, Rechteck, Dreieck) gewählt werden. Wird dann wieder auf das Register *Einstellbar* zurückgeschaltet, dann ist die zuletzt angewählte Signalform übernommen worden, und die Werte müssen u.U. nur noch leicht verändert werden. Die Werteanpassung kann nach Auswahl des Index mittels Werteingabe in das entsprechende Feld des Dialogs oder mittels Mausbedienung in der Kurve erfolgen.

❑ ***Register Sinus***

Für das Sinussignal sind keine weiteren Einstellungen vorgesehen.

❑ ***Register Rechteck***

Das Rechtecksignal kann zeitlich unsymmetrisch verlaufen. Die beiden "Halbwellen" lassen sich über die Zeit T1 einstellen (Angabe in Sekunden). Wird die Option Verhältnis angeklickt, dann beschreibt der Wert im Feld T1 das Verhältnis T1/T2.

❑ ***Register Dreieck***

Hier gelten die gleichen Hinweise wie unter Rechteck.

5

5.1.12. System UTC Time

ibaLogic verfügt über eine so genannte Echtzeitbasis. Echtzeit bedeutet dabei, dass Schaltvorgänge anhand von Datum und Uhrzeit innerhalb von ibaLogic vorgenommen werden können.

Dazu werden in ibaLogic eine Ressource *System UTC Time* sowie Bausteine wie z.B. SplitUtcTIME zur Verfügung gestellt.

Probleme können bei der Umstellung von Sommer auf Winterzeit und umgekehrt auftreten, da es darauf ankommt, wie (und zu welchem Zeitpunkt) das System konfiguriert worden ist.

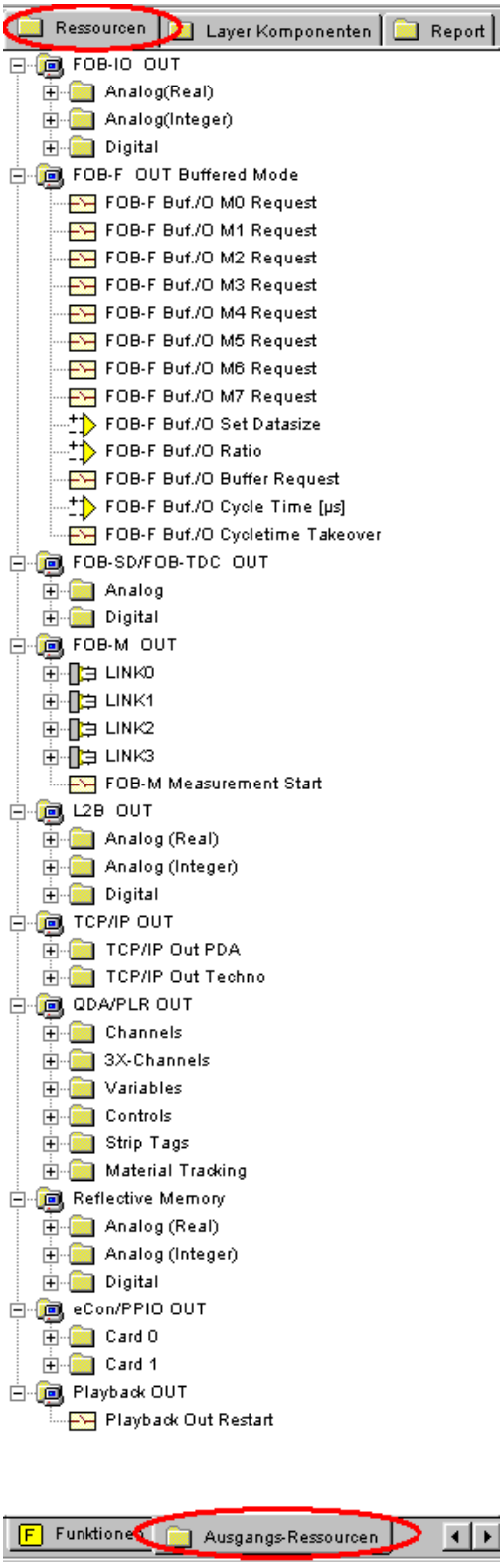
Achtung: Sommer-Winterzeitumstellung:

In der Eigenschaftenmaske "Eigenschaften von Datum / Uhrzeit", Register "Zeitzone", von Windows darf die Checkbox "Uhr automatisch auf Sommer- / Winterzeit umstellen" ***nicht*** *angewählt sein!*

Die Umstellung muss vor einem Wechsel von Winter- auf Sommerzeit deaktiviert werden; danach nützt es nichts mehr!

5.2 Ausgangsressourcen

Die zur Verfügung stehenden Ausgangsressourcen wurden in folgende Bereiche unterteilt:



- FOB-IO/ OUT

Standardisierte analoge und digitale Ausgänge, gruppiert in 32 Module mit je 32 Kanälen (256 max). Auskopplung über LWL an PADU-(Parallel Analog Digital Units), WAGO-Remote-I/O-Klemmen oder SM64-Karten.

- FOB-F/ OUT Buffered Mode

Vordefinierter Satz von Ausgangsvariablen zur Steuerung von Messsystemen, die gepufferte Daten der FOB-F-Karten nutzen (z.B. FFT-Anwendungen). Datenabfrage für bis zu acht Module.

- FOB-SD/FOB-TDC OUT

Vollautomatischer Koppelpartner zu SIMADYN-D Automatisierungsgeräten (CS12/13/14) gruppiert in analoge und digitale Ausgänge mit 8 Modulen zu je 32 Kanälen (256 max).

- FOB-M/ OUT

Vordefinierter Satz von Ausgangsvariablen für FOB-M bzw. Padu8 ICP (25KHz) Messsystem. (Vibrationsmessungen)

- L2B/ OUT

Standardisierte analoge und digitale Eingaben, 32 Gruppen (Module) mit je 32 Ausgaben (max. 1024). Anschluss über Profibus an

1. Profibus-Slave (z.B. SIEMENS S7)

- TCP/IP-OUT

TCP/IP-Ausgangsvariablen, unterteilt in:
1.) TCP/IP-Ausgänge an das PDA-System mit analogen und digitalen Ausgängen in 16 Modulen zu je 32 Kanälen (512 max.).
2.) TCP/IP-Ausgänge als TechnoString beliebigen Inhalts, siehe Dialogfenster "*Ansicht & TCP/IP Ausgang...*".

- QDA/PLR OUT

Vordefinierter Satz von Ausgangsvariablen an das QDA- oder PLR-System.

- Reflective Memory

Vordefinierter Satz von Ausgangsvariablen für eine Reflective Memory-Verbindung. Analoge (Integer oder Real) und digitale Eingänge, gruppiert in je 32 Module mit je 32 Ausgängen (1024 max). Für die Kopplung ist eine spezielle Hardware erforderlich (PC-Karte von VMIC).

- eCon/PPIO OUT

Vordefinierter Satz von 32 Ausgangsvariablen an eine parallele Druckerschnittstelle des PCs.

- Playback OUT

Ein digitaler "Ausgang" für den Neustart des Playbacks.

5.2.1. FOB-IO oder FOB 4o-Ausgangsressourcen

FOB-IO Ausgangsressourcen sind unterteilt in:

- ❑ Analog (Real) Module 0...31,
- ❑ Analog (Integer) Module 0...31 und
- ❑ Digital Module 0...31

Jedes Modul verfügt über 32 Ausgänge, d.h. maximal sind 32 * 32 = 1024 analoge und 1024 binäre Ausgangswerte verschaltbar.

Jeder optische Anschluss einer FOB IO- oder FOB 4o-Karte ist verschaltet zu zwei Modulen mit jeweils 32 Ausgängen, d.h. insgesamt 64 analogen und 64 binären Ausgängen.

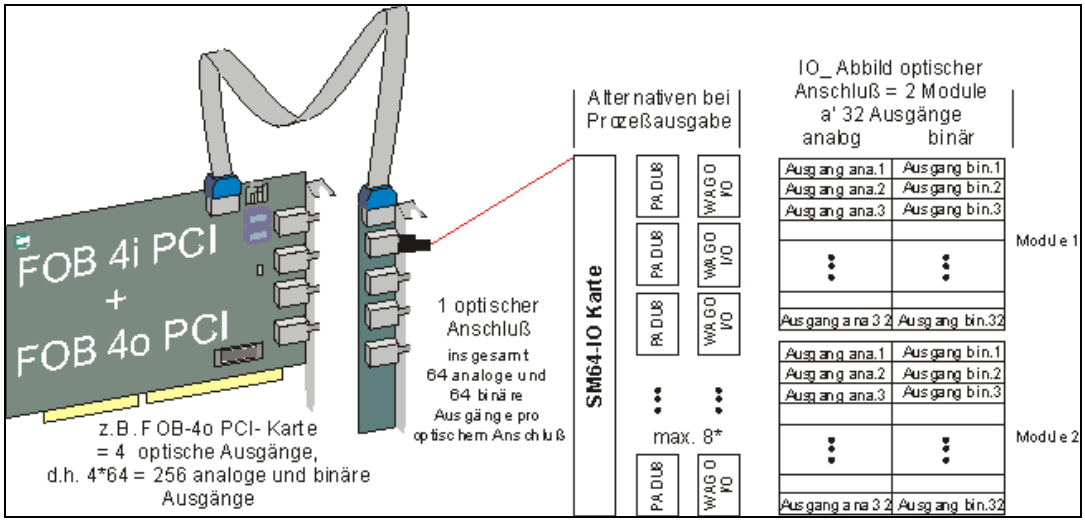


Bild 77 FOB 4o Ausgangsverbindungen

Ein optischer Anschluss kann alternativ verschaltet werden mit:

- ❑ einer SM 64-IO-Baugruppe (64 analoge und 64 binäre Ausgänge)
- ❑ acht PADU8-Output-Einheiten (8*8 = 64 analoge und 64 binäre Ausgänge)
- ❑ acht ibaNet750-Köpfen (8*8 = 64 analoge und 64 binäre Ausgänge)

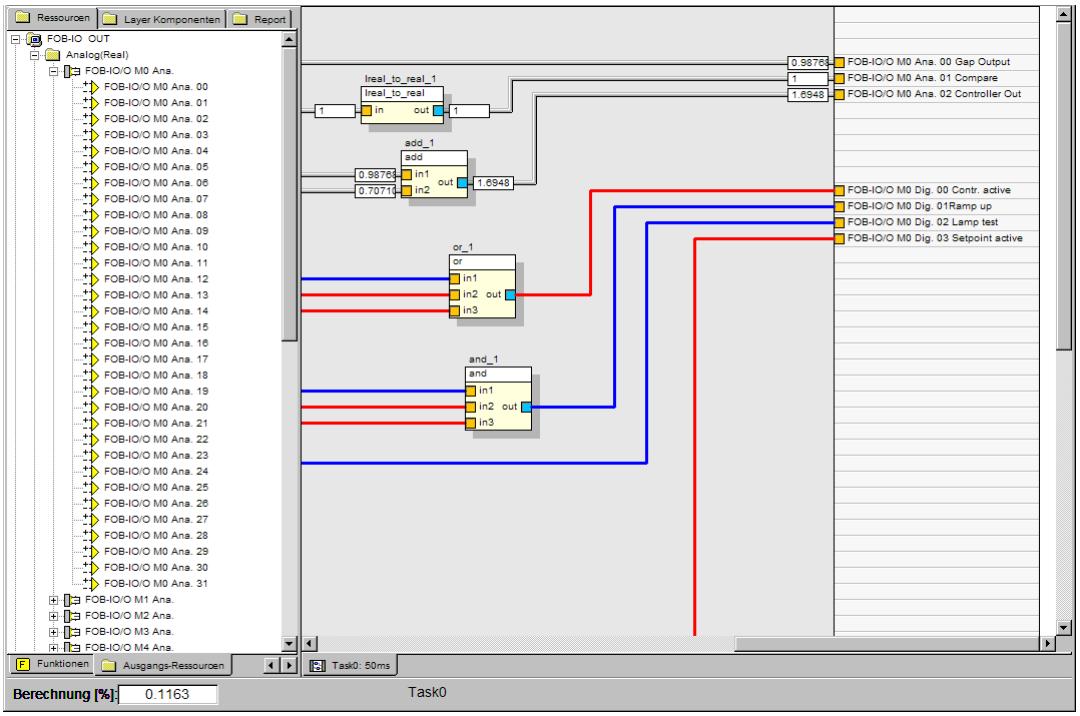


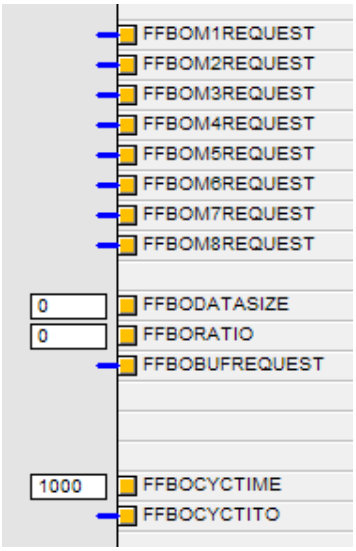
Bild 78 FOB-IO Ausgangsressourcen (Beispiel)

Das Beispiel zeigt die Verschaltung von analogen und binären FOB-IO Ausgangsressourcen.

Bei Bedarf können sämtliche 32 Ausgänge eines Moduls zusammenhängend auf die Ausgangsleiste plaziert werden. Dazu das entsprechende Modul selektieren, auf die Ausgangsleiste verschieben und die anschl. Frage "Aufsplitten des Arrays in Einzelsignale?" mit "Ja" beantworten.

5.2.2. FOB-F OUT Buffered Mode

Bei den Ausgaberessourcen für den FOB-F Buffered Mode handelt es sich nicht um Datenausgaben an den Prozess sondern um Steuerungsausgänge für das Lesen der gepufferten Eingänge (vergl. 5.1.2)



8 digitale Ausgaben zur gezielten, modulspezifischen Anforderung gepufferter Daten vom FOB-F-Interface. (Optimierung der Prozessorlast, Reduzierung des Verwaltungsaufwands)

Datasize ist die Anzahl der Messwerte, die auf einmal von der ibaLogic-Laufzeitumgebung zur Verfügung gestellt werden sollen (max 256).

Ratio ist ein ganzzahliges Vielfaches der in der Erfassungszeit eingelesenen Messwerte. Ratio = 2 bedeutet z.B., dass nur jeder zweite Messwert in den Puffer eingetragen wird.

Bufrequest ist der Steuerausgang an die ibaLogic-Laufzeitumgebung. Bufrequest = TRUE bedeutet, dass Daten der Anzahl Datasize unter Berücksichtigung von Ratio in den Puffer einzutragen sind. Der Puffer soll dann an die Task übertragen und der Eingang FillCount um 1 erhöht werden.

Cyctime ist die auszugebende Zykluszeit am Lichtwellenleiter-Anschluss (1 – 10μs) für den Asynchronmodus.

Cyctito ist das Übernahmesignal für die auszugebende Zykluszeit im Asynchronmodus.

5.2.3. FOB-SD/FOB-TDC OUT-Ausgangsressourcen

Hierbei handelt es sich um einen vollautomatischen Koppelpartner zu SIMADYN-D Automatisierungsgeräten. Die Ausgangsressourcen sind entsprechend der FOB-IO Anschaltungen unterteilt in:

- ❑ Analog (Real) Module 0...7 und
- ❑ Digital Module 0...7

Jedes Modul verfügt über 32 Ausgänge, d.h. maximal sind 8 * 32 = 256 analoge und 256 binäre Ausgangswerte verschaltbar.

5.2.4. FOB-M OUT-Ausgangsressourcen

Die FOB-M Output-Ressourcen dienen der Aktivierung und Parametrierung der PADU-ICP Analog-Digitalwandler (25 kHz). Es werden insgesamt vier Verbindungen mit jeweils einer PADU-ICP-Einheit (8 Kanäle) von ibaLogic unterstützt (zwei FOB-Ms mit je zwei Kanälen).

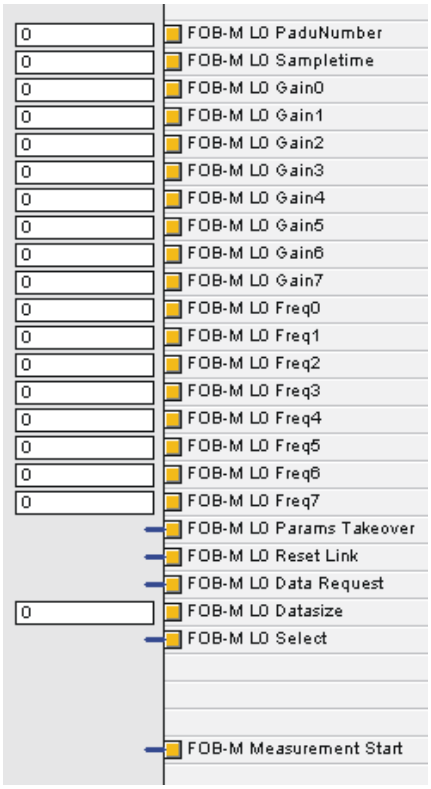
Nummer der gewünschten Padu-ICP Einheit
Gewünschte Abtastzeit (in µs)

Gewünschter Verstärkungsfaktor für Kanäle 0...7 (0..63dB)

Gewünschte Eckfrequenz für Tiefpass Kanäle 0...7 in Hz.

Trigger-Anstoß zur Übertragung der Parameter an PADU-ICP
Rücksetzen der Verbindung
Daten-Anforderungs-Trigger von PADU-ICP
Gewünschte Datenblockgröße (wird gerundet zu vollen Vielfachen von 10 (Maximal 2050 Werte)
Freigabe dieser Verbindung zur Messung

Start der Messung



Zur Parameteränderung muss die aktuelle Messung zunächst gestoppt werden. Anschließend können die neuen Parameter zum PADU-ICP übertragen werden.

Die PADU-ICP-Einheit benötigt eine interne Bearbeitungszeit von bis zu 10 s um Änderungen des Verstärkungsfaktors zu übernehmen. Nach der Umparametrierung sendet das Gerät die neuen Daten fortlaufend an ibaLogic. Dieser Vorgang hat auch Auswirkungen auf andere I/O-Anschaltungen (z.B. FOB-IO), da der ibaLogic I/O-Treiber für ca. zwei Zyklen gestoppt werden muss.

Datenpuffer:

Um eine einwandfreie Funktion der Datenübertragung mit fortlaufenden Datenblöcken sicherzustellen, wurden verschiedene Datenpuffer mit festen Größen hintereinandergeschaltet:

- ❑ FOB-M-Anschaltung, Größe Datenpuffer: 1.024 Werte pro Kanal
- ❑ I/O-Treiber, Größe des Datenpuffers: 25.000 Werte pro Kanal
- ❑ ibaLogic, Größe des Datenpuffers: 50.000 Werte pro Kanal

Daraus ergeben sich folgende Abtast- bzw. Task-Zykluszeiten:

- ❑ PADU-ICP-Abtast-Zyklus z.B. 40 µs
- ❑ Größe der übertragenen Datenblöcke: z.B. 2050 Werte
- ❑ ibaLogic-Task-Abtastzeit z.B. 25 ms

1 / 25ms x 2050 = 82.000 Werte/sec/Kanal: Data read rate (DRR)

1 / 40µs = 25.000 Werte/sec/Kanal: Data generation rate (DGR)

Die Dateneinleserate (data read rate) sollte mindestens 3 mal größer als die Datenerzeugungsrate (data generation rate) sein !

Es ist wichtig sicherzustellen, dass ein verlorengegangener Abtastzyklus nicht zu einem Datenverlust in ibaLogic führt.

5.2.5. TCP/IP OUT-Ausgangsressourcen

Für Ausgaben über TCP/IP stehen zwei verschiedene Ausgangsressourcen zur Verfügung:

- TCP/IP Out PDA, zur Ausgabe von Messwerten an ein ibaPDA-System
- TCP/IP Out Techno, zur Ausgabe von Technostrings, z.B. an ein ibaPDA-System

TCP/IP-Out PDA-Ausgänge

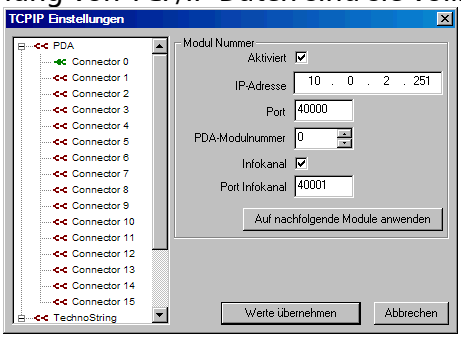
❑	Analog (Real)	Module 0...15,
❑	Digital	Module 0...15 und
❑	Control	Steuerausgänge je Modul 0...15

Jedes Modul verfügt über 32 Ausgänge, d.h. maximal können 16 * 32 = 512 analoge und 512 binäre Ausgangswerte von ibaLogic an ein PDA-System über TCP/IP gesendet werden.

Zur Steuerung der TCP/IP-Übertragung ist eine Control-Ausgangsleiste vorhanden. Jeder der Sendekanäle 0...15 (entsprechend den Modulen 0...15) kann hierüber einzeln gestartet und gestoppt werden. Zur Übertragung der Werte muss der jeweilige Control-Ausgang *TOUTPDA Send xx* = TRUE sein.

Einstellungen für die Ausgabe von Werten an ein PDA-System

1. Im Menü ↪*Datei* ↪*Systemeinstellungen* ↪*Register Sonstige* ↪*Bereich TCP/IP* muss die TCP/IP-Kommunikation grundsätzlich aktiviert sein (Häkchen).
2. Über die Schaltfläche Konfiguration im selben Dialog bzw. über das Menü ↪*Datei* ↪*PCI-Konfiguration* ↪*TCP/IP Out Einstellungen* nun das Dialogfenster für die TCPIP-Einstellungen öffnen. Die Einstellungen in diesem Dialogfenster beziehen sich ausschließlich auf die TCP/IP-Ausgaben. Für den Empfang von TCP/IP-Daten sind sie völlig irrelevant.



3. Mit der Maus den ersten "Connector" (0) in der Baumstruktur im Zweig PDA anklicken. Jeder Connector entspricht genau einem Modul in den TCP/IP Out PDA-Ausgangsressourcen.
4. Diese Verbindung nun aktivieren (Häkchen machen), die IP-Adresse des Zielrechners (PDA-Rechner) und die vereinbarte Portnummer eintragen. Durch die individuelle Adressierung der einzelnen Verbindungen können auch verschiedene PDA-Systeme mit Werten versorgt werden.
5. Bei Bedarf können die Werte dieser Verbindung auf eine andere Modulnummer im PDA-System rangiert werden, wenn z.B. die Module 0...15 im PDA-System bereits von anderen Datenquellen belegt sind (Padus o.ä.).
6. Optional kann die Übertragung des Infokanals unterdrückt oder freigegeben werden. Mit dem Infokanal werden Zusatzinformationen übertragen, die später in der von ibaPDA erzeugten Messdatei wieder zu finden sind.

7 Sollen mehrere Module (Connectors) an dasselbe PDA-System übertragen werden, dann auf die Schaltfläche "Auf nachfolgende Module anwenden" klicken, und die Einstellungen werden für die Module (Connectors) unterhalb des aktuellen übernommen.

8 Abschließen die Schaltfläche "Werte übernehmen" bzw. "Konfiguration speichern" betätigen.

Eine aktivierte Verbindung wird mit einem grünen Symbol gekennzeichnet.

TCP/IP Out Techno-Ausgänge

- ❑ Data (String) bis zu vier Technostrings 0...3 und
- ❑ Control Steuerausgänge für jeden String

Jede Technostring-Ausgabe kann ASCII-Strings mit bis zu 1024 Zeichen, incl. Endezeichen (0 hex) verschicken.

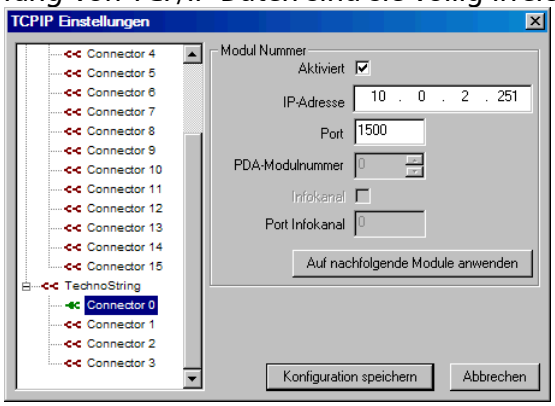
Zur Steuerung der TCP/IP-Übertragung ist eine Control-Ausgangsleiste vorhanden. Jeder der Sendekanäle 0...3 (entsprechend den Technostrings 0...3) kann hierüber einzeln gestartet und gestoppt werden. Zur Übertragung der Strings muss der jeweilige Control-Ausgang *TOUTTECHNO Send x* = TRUE sein.

5

Einstellungen für die Ausgabe von Technostrings

1 Im Menü ↪*Datei* ↪*Systemeinstellungen* ↪*Register Sonstige* ↪*Bereich TCP/IP* muss die TCP/IP-Kommunikation grundsätzlich aktiviert sein (Häkchen).

2 Über die Schaltfläche Konfiguration im selben Dialog bzw. über das Menü ↪*Datei* ↪*PCI-Konfiguration* ↪*TCP/IP Out Einstellungen* nun das Dialogfenster für die TCPIP-Einstellungen öffnen. Die Einstellungen in diesem Dialogfenster beziehen sich ausschließlich auf die TCP/IP-Ausgaben. Für den *Empfang* von TCP/IP-Daten sind sie völlig irrelevant.



3 Mit der Maus den ersten "Connector" (0) in der Baumstruktur im Zweig TechnoString anklicken. Jeder Connector entspricht genau einem Technostring, d.h. einer TCP/IP Out Techno-Ausgangsressource.

4 Diese Verbindung nun aktivieren (Häkchen machen), die IP-Adresse des Zielrechners (z.B. PDA-Rechner) und die vereinbarte Portnummer eintragen. Die Portnummer sollte sich von der für die Übertragung von Messwerten unterscheiden! Durch die individuelle Adressierung der einzelnen Verbindungen können auch verschiedene Zielstationen mit Technostrings versorgt werden.

5 Sollen mehrere Technostrings (Connectors) an dasselbe Zielsystem übertragen werden, dann auf die Schaltfläche "Auf nachfolgende Module anwenden" klicken, und die Einstellungen werden für die Module (Connectors) unterhalb des aktuellen übernommen.

6 Abschließend die Schaltfläche "Werte übernehmen" bzw. "Konfiguration speichern" betätigen.

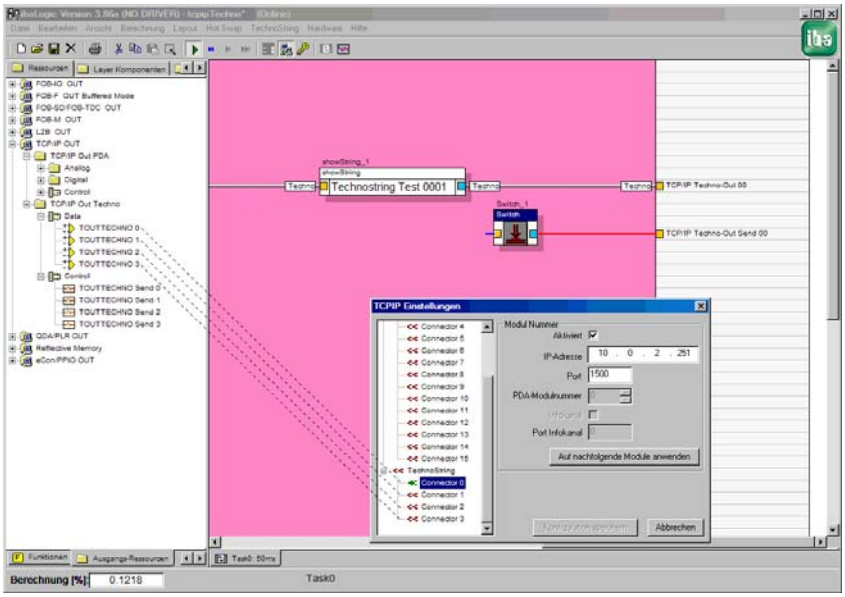


Bild 79 TCPIP TECHNO Out, Beziehung zw. Ausgangssignal und TCP/IP Einstellungen

ibaLogic verschickt seine Technostrings stets mit einem leeren Terminierungszeichen (0 hex). Darum muss in den Technostring-Einstellungen (im Menü ↪PDA Einstellungen) in ibaPDA als Terminierungszeichen nicht "Carriage Return" sondern "Andere" gewählt und in das zugehörige Feld eine 0 (Null) eingetragen werden.

Außerdem ist darauf zu achten, dass keine anderen Teilnehmer in dem Netzwerk die Portnummern verwenden, die für die Technostring-Kommunikation vereinbart wurden. Die Systeme können sich sonst gegenseitig stören.

5

Einstellungen bei älteren ibaLogic-Versionen:

Bei älteren ibaLogic-Versionen wird die TCP/IP-Kommunikation in dem Dialog für die ISA-Konfiguration vorgenommen.

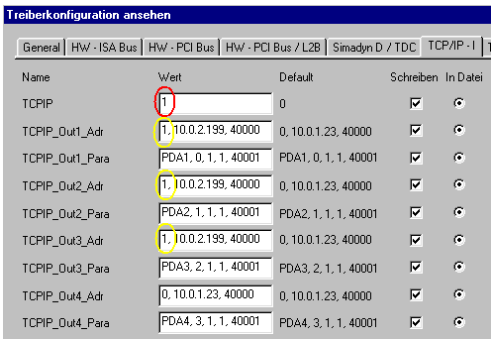


Bild 80 TCP/IP einstellungen in früheren Versionen von ibaLogic

5

Zur Freigabe der TCP/IP-PDA Out-Kanäle muss erst der notwendige Treiber freigegeben werden (TCPIP auf "1" setzen). Für jede Ausgangsressource muss die Freigabe zusätzlich individuell erfolgen, und es muss die IP-Adresse des PDA-Rechners (im Beispiel oben 10.0.2.199) sowie die Portnummer (z.B. 40000) eingegeben werden.

Die TCP/IP-**PDA** Out-Kanäle werden mit den Parametern TCPIP_Out**1**_Adr / ..._Para bis TCPIP_Out**16**_Adr / ..._Para konfiguriert.

Die TCP/IP **Techno** Out-Kanäle werden mit den Parametern TCPIP_Out**17**_Adr / ..._Para bis TCPIP_Out**20**_Adr / ..._Para konfiguriert.

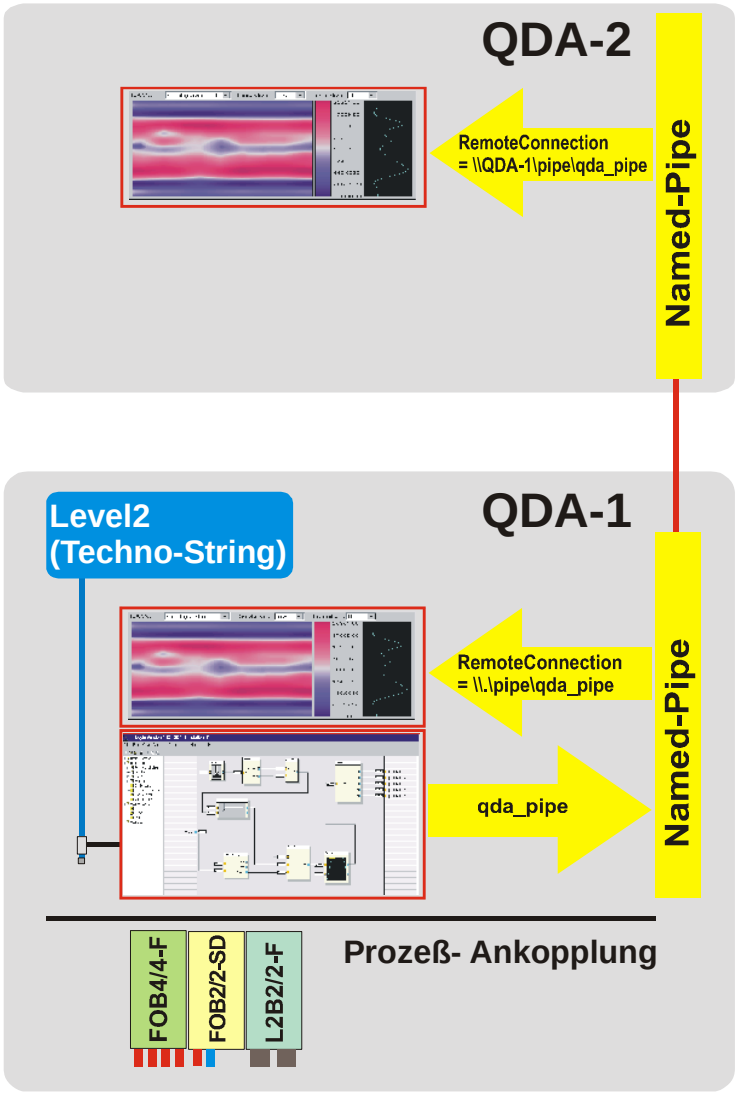
Dazu das Kommando "*Ansicht & Treiber-Konfiguration* " anwählen, und in den Registern "*TCP/IP I*" und "*TCP/IP II*" die entsprechenden Änderungen vornehmen. Den Vorgang mit "In iba_drv.cfg speichern" abschließen. Die Änderungen werden erst nach einem Neustart der Treiber wirksam, was bei älteren Versionen einen Neustart von ibaLogic erfordert.

Dieser Dialog steht auch in der aktuellen ibaLogic-Version (386) noch zur Verfügung, allerdings im Menü ↪*Datei*.

5.2.6. QDA OUT-Ausgangsressourcen

Um den Kommunikationsmechanismus zwischen QDA und ibaLogic zu verstehen, folgt eine kurze Einführung in die "Named Pipes". Dieser Mechanismus wird benutzt, um QDA mit ibaLogic über TCP/IP-Netze zu verbinden.

Merkmale der ibaLogic Kommunikation über Named Pipes:



Das iba "Named Pipe"-Konzept verfügt über die Möglichkeit, mehrfach synchronisierte PC-Arbeitsplätze zu einzusetzen. Folglich können die Arbeitsplätze überall dort plaziert werden, wo sie erforderlich sind. In den meisten Fällen wird der erste PC im Schalthaus stationiert. Er verfügt über alle notwendigen Hardwarevoraussetzungen zur Einkopplung der Messsignale. Weitere PC' s werden z.B. auf Steuerbühnen oder überall dort stationiert, wo es sinnvoll ist.

ibaLogic verwendet das "Named Pipe"-Konzept " um mit einer Vielzahl von Anwendungen - sogar mit sich selbst - zu kommunizieren. Die "Named Pipes" sind eine TCP/IP Anwendungsfunktionalität, die auf allen Windows-Arbeitsplätzen vorhanden ist.

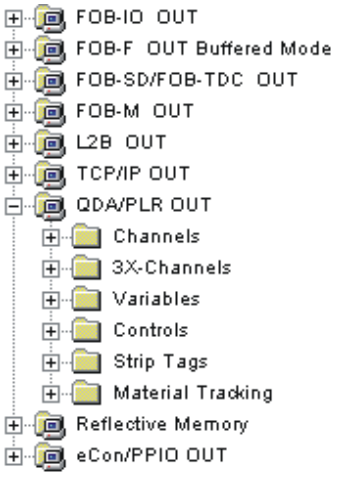
Das Beispiel zeigt zwei miteinander synchronisierte QDA-PCs, die über Named Pipes zu einer einzelnen ibaLogic-Quelle verbunden sind. Der komplette Datensatz wird zum lokalen QDA auf PC "QDA-1" und gleichermaßen zum PC "QDA-2" geschickt.

Achtung: Die Referenz "QDA-1" auf PC QDA-2 ist ein Verweis auf den Namen des PC, nicht auf die ibaLogic-Anwendung, während "qda_pipe" der Adressierungshinweis für die Anwendung ist.

Sobald ein Signal mit einem Namen definiert wurde, sind alle anderen Anwendungen in Vorwärtsrichtung (Signalfluss) sofort in der Lage, diesen Namen zu verwenden. So muss jedes Signal nur einmal genannt werden.

5.2.7. QDA / PLR OUT

Alle Betriebsmittel, die auf QDA einwirken, werden innerhalb des QDA OUT-Abschnitts der ibaLogic-Ausgangsressourcen organisiert.



Dies sind:

- Channels (Kanäle)
- 3X-channels (3D-Kanäle)
- Variables (Variablen)
- Controls (Steuerelemente)
- Strip Tags (Beschriftung Aufzeichnungsstreifen)
- Material Tracking (Materialverfolgung)

Im folgenden Abschnitt werden die Ressourcen erklärt:

5

5.2.7.1. Channels

ibaLogic unterstützt bis zu 96 Kanäle, die wie folgt strukturiert sind:

- ❑ Value CH # (float) // Signal Wert (Gleitpunkt)
- ❑ Reference CH # (float) // Referenz Wert (Gleitpunkt)
- ❑ Low. Limit CH # (float) // QDA Wert für untere Signalbegrenzung
- ❑ Up. Limit CH # (float) // QDA Wert für obere Signalbegrenzung

Diese Kanäle können vorgewählt werden, um auf einem oder mehreren der QDA-Recorder 1..6 beobachtet zu werden.

Wie alle Ausgangsressourcen zu QDA, können die Namen der Ausgänge (z.B. Value CH # 1) individuell verändert werden (z.B. in "Tension1") mittels Doppelklick auf den Namen mit der linken Maustaste. Dieser Name wird dann an QDA über die "Named Pipe" übermittelt.

Anmerkung: Wenn Sie einen anderen Logikplan (Layout) laden, werden die Ressourcen nicht mit den individuellen Variablennamen aktualisiert, die im Plan vorhanden sind. Aber der Logikplan selbst kennt die vorgegebenen Namen. So hat QDA immer die korrekte Zuordnung zum Signal, obwohl die Variable innerhalb der Ausgangsrandleiste mit dem Default-Namen "Value CH1" bezeichnet wird.

Weil die Kanal-Informationsdaten nur 1mal pro Minute gesendet werden, kann ein Wiederanlauf von QDA helfen, den Updateprozess zu beschleunigen

5.2.7.2. 3X-Channels für QDA und ibaVision3X

Die zwei 3D-Kanäle "Planheit 1" und "Planheit 2" sind fest zum QDA 3X-Window verdrahtet. Daher müssen nur die Daten, die von der Planheitsregelung (z.B. Siemens-Planheits-PC) kommen, zum entsprechenden 3X-Kanal verschaltet werden.

- ❑ Bis zu 2* 128 3D-Kanäle werden von ibaLogic unterstützt.
- ❑ QDA unterstützt ein 3D-Window.
- ❑ ibaVision3X unterstützt eine unbegrenzte Anzahl von Fenstern, die auf die gleiche ibaLogic Pipe zugreifen.

Im folgenden Kapitel werden die zusätzlichen Variablen zum Steuern der QDA 3X-Anzeige beschrieben.

5.2.7.3. Variable

Der Variablensatz wird zur Fernparametrierung der QDA-Einstellungen benutzt. In der Regel sind sie an einen Input-TechnoString angeschlossen, der über TCP/IP oder direkt von der SPS kommt. Es gibt drei unterschiedliche QDA-Teile, die sich auf diese Signale beziehen.

- QDA Datei Speicherung und Material-Informationen
- QDA 3X-Window Skalierung
- QDA FFT Window Gerüst Symbole

Variablenname / Ressource	Bedeutung in ibaLogic	Aktion in QDA, sofern angeschlossen	Kommentar
counter	Anzahl empf.Telegramme [float]	keine	
Data version number	Version des Datensatzes [string]	keine	
Time stamp	Aktuelle Zeit des Datensatzes [string]	Zeitstempel im Rekorder-Streifen	
Strip id	"Name" der Coil-ID [string]	Zu speichernder Name der Datei, sobald im QDA-Setup Menü vorgewählt, (Generiere Dateinamen anhand Coil-ID)	Mit der Fähigkeit, String-Variablen zu verarbeiten ist ibaLogic in der Lage, die Coil-ID mit einem anderen Ergeignis zu einem neuen Dateinamen zu verbinden.
Strip length	Erwartete Bandlänge (konstant für ein Band!) [float] [m]	Skaliert die x-Achse von längenbasierten QDA-Streifen sowie die 3D-Darstellung (Statische Freigabe in QDA-Eigenschaften notwendig)	Bei ständiger Änderung der Bandlänge während der Aufzeichnung, können bei QDA Performance-Probleme auftreten, durch Anpassung der x-Achsen-Skalierung. Minimale Bandlänge: 200m!
Head length	Länge Bandkopf [float] [m]	keine	
Tail length	Länge Bandfuß [float] [m]	keine	
S1: Diameter BUR top : **S5: Diameter BUR top**	Durchmesser obere Stützwalze für Gerüst 1 bis 5 [float] [mm]	Alle diese geometrischen Maße beeinflussen das Verhalten der Gerüst-Symbole im QDA FFT-Window. Mit bekannten Gerüstgeschwindigkeiten und Getriebeverhältnissen können mögliche Exzentrizitäten der Walzen ermittelt werden.	
S1: Diameter BUR bottom	Durchmesser untere Stützwalze für Gerüst 1 bis 5 [float] [mm]		
S4: Diameter IMR top, **S5: Diameter IMR top**	Durchm. obere Zwischenwalzen Gerüst 4 und 5 [float] [mm]		
S4: Diameter IMR bottom, **S5: Diameter IMR bottom**	Durchm. untere Zwischenwalzen Gerüst 4 und 5 [float] [mm]		
S1: Diameter WR top : **S5: Diameter WR top:**	Durchm. obere Arbeitswalzen Gerüst 1 bis 5 [float] [mm]		
S1: Diameter WR bottom : **S5: Diameter WR bottom**	*Durchm. untere Arbeitswalzen Gerüst 1 bis 5 [float] [mm]*		
S1: Thickness set point : **S5: Thickness set point**	Erwartete Banddicke nach Gerüst 1 bis 5 [float] [mm]	keine	
S1:Reduction (out of rolling directive) : **S5: Reduction (out of rolling directive)**	Reduktionsfaktor zwischen ein- und auslaufendem Band pro Gerüst in Prozent [float] [%]	Steuert die Geschwindigkeitsvorsteuerung der QDA-Streifen.	Diese Faktoren kommen üblicherweise vom Level 2 Computer
Small zones **Wide zones** **First zone** **Last zone**	Definiert die Anzahl der schmalen und breiten Zonen der Planheitsmessung [float] [-] Die erste und letzte Zone steuern die Anzeige der Bandbreite. Nicht belegte Zonen werden ausgeblendet	Anzahl der breiten Zonen in der Mitte des Messrolle und schmalen Zonen am Rand zur Steuerung der 3D-Darstellung. Die Summe breite + schmale Zonen muss mit der Anzahl 3D-Signale identisch sein!	
Message Rec. 1 : **Message Rec. 6**	Name des Rekorder-Streifens [string]	Anzeige in der zugeordneten "Recorder strip message box"	
Variables 90 to 97	Rekorder Status 1 bis 8	Verschaltet zur Rekorder Window 1 to 8. Mit log. "0 " wird die Darstellung gesperrt, mit "1" freigegeben.	In einigen QDA-Versionen müssen diese Ressourcen zur zeitbezogenen Aufzeichnung verschaltet und freigegeben werden ! **UseRecStat = 1**
Variable xyz	Reservierte Signale	Nicht verschaltet zu QDA, Bitte nicht benutzen !	

5.2.7.4. Controls

Die QDA-"Controls" Steuervariablen unterstützen vier unterschiedliche Funktionen zur Steuerung des QDA-Recorders.

- ❑ Start Acquisition: Startet den QDA Rekorder bei Übergang von log. “0” auf log. “1”
- ❑ Stop Acquisition: Stoppt den QDA Rekorder bei Übergang von log “0” auf log. “1”
- ❑ Pause Acquisition: Anhalten des QDA-Rekorders solange das Signal auf log “1” gesetzt ist
- ❑ Print: Hardcopy-Ausdruck der aktuellen Bildschirmanzeige
- ❑ Save CAM: Speichert den CAM Inhalt
- ❑ Length Trigger: Meterpulse für QDA Trend Window (Längenbasierte Statistik)
- ❑ Head: Definiert die Phase, während der Bandkopf gewalzt wird
- ❑ Steady state: Phase "Steady state" (konstante Bedingungen)
- ❑ Tail: Coil-Bandfuß Phase zur Freigabe bestimmter Berechnungen.

Die Pausenfunktion ermöglicht die Einsparung von Aufnahmekapazität, z.B. während das Gerüst für eine Reparatur gestoppt wurde, ohne die Relation zwischen dem Band im Gerüst und der zugeordneten Datei zu verlieren. Der Rekorders wartet, bis die Pause aufgehoben wird, um die Aufnahme in der gleichen Datei fortzusetzen. Wegen dieses Verhaltens ist es nützlich, die Pause für die spätere Analysephase anzugeben.

5.2.7.5. Material tracking (QDA Recorder #6 controls)

Zur Steuerung der komplexen Funktion einer längenbasierten Online-Aufzeichnung wurde ein umfangreicher Satz von Steuerelementen integriert. Das Gegenstück zu dieser Funktionalität ist der Recorder#6 in QDA.

Die Steuerfunktionen wurden in 2 Abschnitte unterteilt: Feeds und Triggers:

- ❑ ***Feeds (Vorschub) 1..8:***

Überwachen den Materialfluss zu jedem der 8 Recorderstreifen im Recorder # 6. Jeder Vorschub (Feed) entspricht genau einem Streifen in diesem Recorder !

- ❑ ***Triggers (Auslöser) 1..8:***

Zeigt an, dass das Material genau diese Position erreicht hat.

Beispiel:

Wenn "Feed1 " den Fluss des Materials, das Gerüst 1 verläßt steuern soll (Anzeige in Recorder#6, Streifen 1 –Zählung beginnt mit 0), muss " Trigger1 " erkennen, dass das Material gerade diese Position erreicht hat.

Die Einrichtung dieser Funktionalität mit ibaLogic + QDA setzt ein gutes Wissen über den Prozess und das Prozessleitsystems voraus.

5.2.7.6. Strip Tags

Dieser Ressourcen-Satz wird benutzt, um die QDA-Streifen-Bezeichnung zu steuern. Für jeden Recorder (6) und jeden Streifen innerhalb eines Recorders (Maximum 8) ist ein Datenstring vorhanden. So ist ibaLogic in der Lage, die Bezeichnung der QDA Online- und Offline-Anzeige zu steuern. Die Bezeichnungen werden von QDA gespeichert. Bei jedem Start-Trigger oder, wenn sich die Streifenbezeichnung geändert hat, wird der Datenstring an QDA übermittelt.

Jeder ASCII String (max. 10 Zeichen) kann an QDR gesendet werden.

Ein Maximum von 20 Bezeichnern pro Streifen und Anzeige wird von QDA zur Verfügung gestellt (d.h. , wenn Sie einen Bezeichner definieren, der sich jede Sekunde ändert, und die QDA-Anzeige wird auf 60 Sekunden eingestellt, sehen Sie 20 Bezeichner, die sich von links nach rechts über den Schirm bewegen).

5.2.8. Reflective Memory (RM)

Die Verknüpfung der RM-Ressourcen mit der RM-Schnittstelle ist Teil der PCI-Konfiguration, wie in Kapitel 2.6.5 beschrieben.

Zu jedem der 32 RM-Ausgangsmodule gehören 32 RM-Ausgangssignale, deren Signalnamen dem Modul fest und eindeutig zugeordnet sind. Zusätzlich erhält jedes Signal eine Beschreibung (Klartext), die sich zugunsten eines besseren technischen Verständnisses editieren läßt.

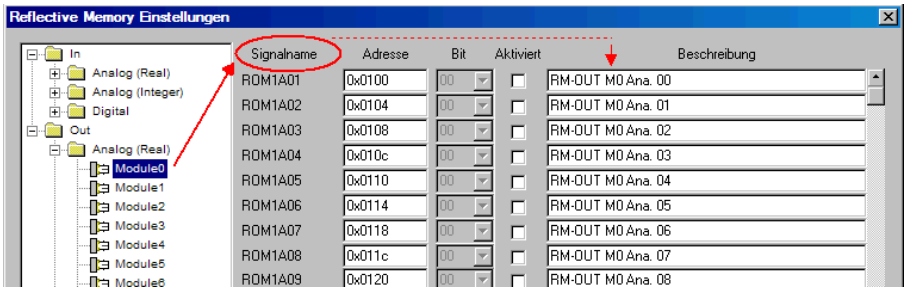


Bild 81 Reflective Memory Ausgangsressourcen, Beziehung zw. Modul, Signalname und Beschreibung

Diese Beschreibungen der Ausgangssignale finden sich im Signalbaum bei den Ausgangsressourcen wieder und werden bei der Verwendung der Ausgangssignale im Funktionsplan benutzt. Sie erscheinen auch im Tooltipp.

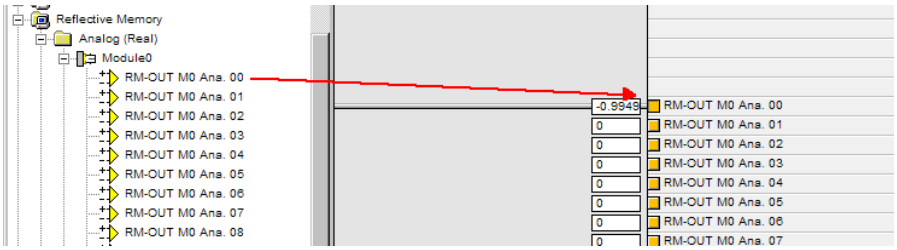


Bild 82 Reflective Memory Ausgangsressourcen, Anzeige der (Signal-)Beschreibung

5.2.9. eCon/PPIO OUT

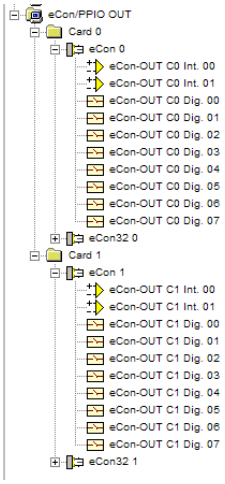
Diese Ausgaberessourcen sind speziell für die Geräte **eCon** und **eCon32** konzipiert.

Bei den eCon-Geräten handelt es sich um kleine Ein-/Ausgabegeräte, die an die Parallelschnittstelle des PCs angeschlossen werden. Es gibt sie in zwei Ausführungen:

eCon: Dieser Typ verfügt über 2 Analog- und 8 Digitalausgänge.

eCon32: Dieses Gerät verfügt über 32 Digitalausgänge.

Bis zu zwei Geräte können miteinander kombiniert werden. Maximal zwei solcher Geräte lassen sich an einem PC-Parallelport betreiben.



Die Zuordnung von eCon-Geräten und Ausgangsressourcen ist wie folgt:

Card 0	erstes eCon an der Parallelschnittstelle
	wenn eCon, dann 2 AA und 8 DA wenn eCon32, dann 32 DA
Card 1	zweites eCon, steckt am ersten eCon
	wenn eCon, dann 2 AA und 8 DA wenn eCon32, dann 32 DA

Die Berechnung der Analogausgaben basiert auf einer 10 Bit-Auflösung (Schrittweite auf der digitalen Seite = 64).

Da das System nicht erkennen kann, ob ein eCon oder ein eCon32 angeschlossen ist, muss außerdem in den Systemeinstellungen der eCon-Typ eingestellt werden.

↗ *Siehe dazu auch Kapitel* 2.5.3

Verbunden damit ist zwangsläufig die Wahl einer Nullmaske, die dafür sorgt, dass beim Offline-Schalten des Layouts alle Ausgänge des eCons auf Null gesetzt werden.

Bei den Analogausgängen ist in diesem Zusammenhang zu beachten, dass der Ausgangswert 0 (Null) der hexadezimalen Angabe 0x8000 (High+Lowbyte) entspricht. Die eCon-Geräte haben einen Analogausgabebereich von –10 V bis +10 V. Die Maskierung 0x0000 würde also in diesem Fall den Ausgangswert –10 V hervorrufen.

Zur Verdeutlichung der Zusammenhänge von Hex-Codierung und Zuordnung zu den Ausgaben soll die folgende Grafik dienen.

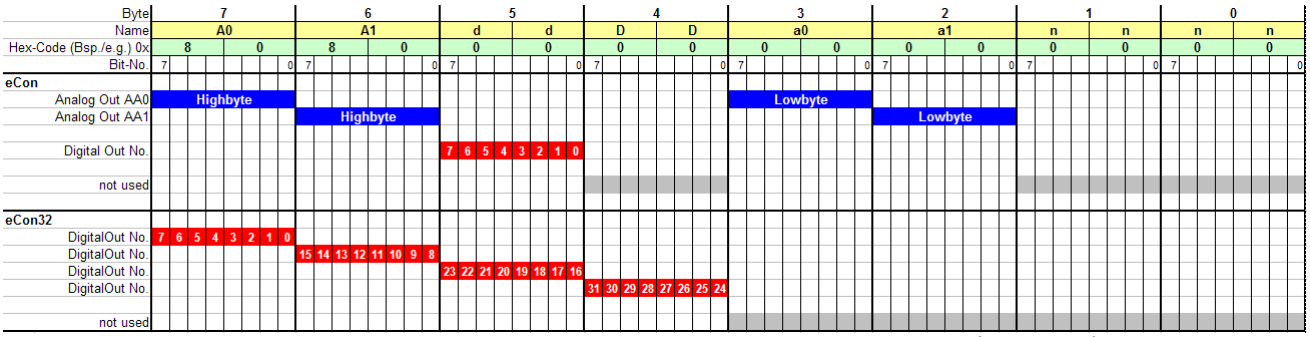


Bild 83 Hex-Adressierung von Analog- und Digitalausgaben bei eCon und eCon32 (Nullmaske)

Weitere Informationen zu den eCon-Geräten finden Sie in der entsprechenden Hardwaredokumentation, die auch auf die ibaLogic-Programmierung eingeht.

eCon_Doku_V1.7d.pdf oder aktueller

5

5.2.10. Playback OUT

Für den Playback-Betrieb gibt es einen digitalen Ausgang, der jedoch intern verwendet wird. Dieser Ausgang kann im ibaLogic-Programm gesetzt oder zurückgesetzt werden. Damit ist es möglich, den Playback-Betrieb, also das Abspielen einer Messdatei, aus dem Layout heraus zu steuern, bzw. neu zu starten.

Playback Out Restart, wenn = TRUE, dann wird der "Cursor" für das Abspielen der Messdatei auf den Dateianfang zurückgesetzt und der Abspielvorgang wir neu gestartet.

5.3 OPC-Kommunikation

Ziel der OPC (**O**LE for **P**rocess **C**ontrol) Standard-Schnittstelle ist es, die Interaktion zwischen Automatisierung / Steueranwendungen, Feldsystemen / Geräten und Geschäfts- / Büroanwendungen in der Automatisierungstechnik zu fördern.

Die von der "OPC-Foundation" spezifizierte Schnittstelle hat schnell eine breite Unterstützung bei Anwendern und Herstellern gefunden und sich inzwischen als leistungsfähige Standardschnittstelle im Windows-Umfeld durchgesetzt. OPC basiert auf der Microsoft OLE/COM-Technologie. Die OPC Spezifikation beinhaltet zwei Schnittstellen-Definitionen; das "Custom-Interface" und das "Automation Interface".

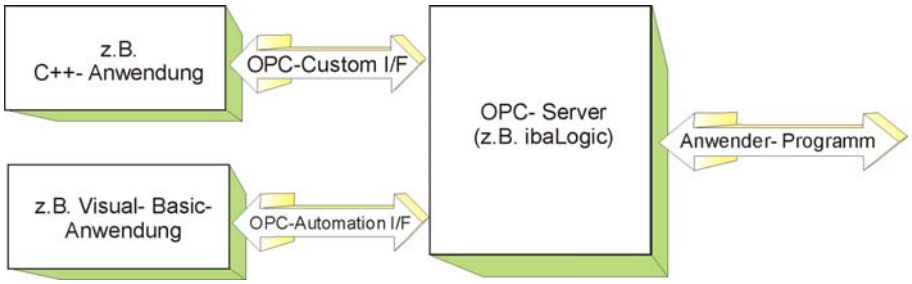


Bild 84 OPC-Schnittstellen

Ein OPC Client (z.B. eine Visual-Basic Anwendung) kommuniziert zum OPC-Server über das "Automation-" Interface. (siehe Literatur- und Quellenangaben, Anhang B, [4] und [5]). Im folgenden Beispiel soll die OPC-Kommunikation zwischen ibaLogic und einer Visual Basic Applikation erläutert werden.

5.3.1. OPC Automation Server Object Modell

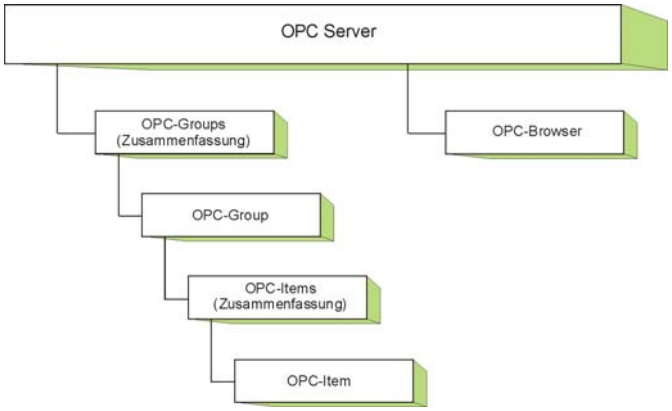


Bild 85 OPC Automation Server Object Modell

Objekt	Beschreibung
OPCServer	Ein Client muss zunächst eine Instanz des OPCServer Objektes einrichten. Anschließend muss der Client es zu einem OPC Data Automation Interface verbinden (Methode "Connect"). Das OPC-Server Objekt kann nun genutzt werden, um generelle Informationen vom Server zu erhalten, und um OPCGroups Objekte einzurichten und zu verwalten.
OPCGroups	Eine Sammlung aller "OPCGroup" Objekte, die ein Client innerhalb eines OPCServers Bereichs eingerichtet hat und die Methoden zu deren Erzeugung, Löschung und Verwaltung. Beinhaltet auch die Default-Eigenschaften der OPCGroup Objekte bei deren Erstellung.
OPCGroup	Ein "OPCGroup" Objekt ist eine Möglichkeit zur Organisation von Daten. Ein Beispiel für eine OPCGroup repräsentiert ein Bedienerbildschirm oder Report. Vom Client werden nur die für das Bild/Report benötigten Daten vom Server mit spezifizierter Übertragungsrate angefordert.
OPCItems	Eine Sammlung aller "OPCItem" Objekte, die ein Client innerhalb eines OPCServers Bereichs eingerichtet hat und die Methoden zu deren Erzeugung, Löschung und Verwaltung. Beinhaltet auch die Default-Eigenschaften der OPCItem Objekte bei deren Erstellung.
OPCItem	Ein "OPCItem" repräsentiert eine Verbindung zu einer Datenquelle im Server. Zu jedem Item gehören: ein Wert (Type Variant), Status-Informationen und Zeitstempel.
OPCBrowser	Ein "OPCBrowser" Objekt zeigt die Hierarchie, d.h. die Zweige (Branches) und Items, die im Server eingerichtet wurden. Die Funktion des Browsers kann optional genutzt werden.

5.3.2. Installation der OPC-Treiber-DLLs

Zur OPC-Kommunikation zwischen ibaLogic und einer Visual Basic Applikation muss zunächst sichergestellt werden, dass auf allen beteiligten PCs die gleichen OPC-DLLs installiert sind (DLL = Dynamic Link Libraries). Die folgenden DLLs werden benötigt:

Opcproxy.dll	76kB	27/11/02
Opccomn_ps.dll	60kB	27/11/02
Opcdaauto.dll	156kB	13/11/00

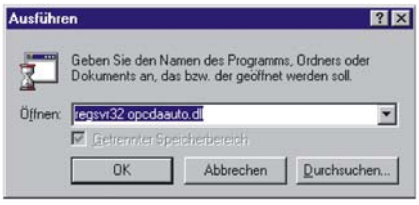
Die drei Dateien befinden sich auf der ibaLogic-CD im Verzeichnis \sample_OPC_VB_V103\OPC_Install\OPC_DLL's.

Für die Registrierung der DLLs gehen Sie wie folgt vor:

1. Wenn Sie eine aktuelle ibaLogic-CD vorliegen haben, so gibt es dort in dem Verzeichnis *sample_OPC_VB_V103\OPC_Install* das Hilfsprogramm *Install.exe*, das die Registrierung der DLLs automatisch vornimmt.
2. Kopieren Sie das gesamte Verzeichnis *sample_OPC_VB_V103* von CD in ein beliebiges Verzeichnis auf der Festplatte des ibaLogic-PCs.
3. Suchen Sie im Windows-Explorer das Programm ...\ *sample_OPC_VB_V103 \OPC_Install\Install.exe* und starten Sie das Programm mit einem Doppelklick. Die erfolgreiche Registrierung der DLLs wird angezeigt.

Wenn Sie keinen Zugriff auf dieses Hilfsprogramm haben (z.B. bei älteren Installationen), dann verfahren Sie wie folgt:

1. Kopieren Sie die o.g. DLLs in das Verzeichnis "c:\Winnt\System32" (Windows NT), bzw. c:\windows\system32 (Windows XP) auf den Festplatten aller beteiligten PCs
2. Mit dem Kommando "regsvr32" werden anschließend alle drei DLLs nacheinander registriert. Betätigen Sie dazu den "Start" Button der Windows-Taskleiste und den Befehl "Ausführen".



3. Stellen Sie sicher, dass alle drei Dateien sowohl auf dem OPC-Server (ibaLogic) als auch auf dem OPC Client (Visual Basic) installiert sind. Bei einem Einzelplatzsystem müssen die DLLs nur einmal installiert werden.

5.3.3. OPC-Beispielapplikation mit Visual Basic

OPC-VB Beispiel (sample_OPC_VB_V103)

Zum besseren Verständnis der OPC-Funktionen von ibaLogic befindet sich auf der CD eine einfache Musterapplikation in dem Pfad \sample_OPC_VB_V103*.*

In diesem Verzeichnis sind alle Programme und Dateien enthalten, die für das Starten der ibaLogic-Applikation erfoderlich sind. Visual Basic (VB) muss dazu nicht auf dem PC installiert sein.

Für Interessierte, die die Visual Basic-Applikation (Projekt) genauer untersuchen und ggf. Teile daraus für eigene Projekte verwenden wollen, sind auch die dafür relevanten Programme und Dateien auf der CD enthalten. Um das VB-Projekt öffnen zu können, muss auf dem PC Visual Basic (Visual Studio) installiert sein.

Um die Musterapplikation zu verwenden, gehen Sie bitte wie folgt vor:

1. Sofern noch nicht geschehen, kopieren Sie das gesamte Verzeichnis *sample_OPC_VB_V103* von CD in ein beliebiges Verzeichnis auf der Festplatte des ibaLogic-PCs.
2. Starten Sie ibaLogic.
3. Öffnen Sie das Beispielprogramm ... *\sample_OPC_VB_V103\LYT-File\sample_layout_OPC_VB_V103.lyt*
4. Schalten Sie ibaLogic mit  in den "Online"-Modus (rosa Hintergrund).
5. Starten Sie im Windows-Explorer das VB-Projekt *\sample_OPC_VB_V103\VB_application\sample_application_OPC_VB_V103.exe* mittels Doppelklick.

Auf dem Monitor erscheint ein neues Fenster mit den Werten der im ibaLogic projektierten Off-Task-Konnektoren.

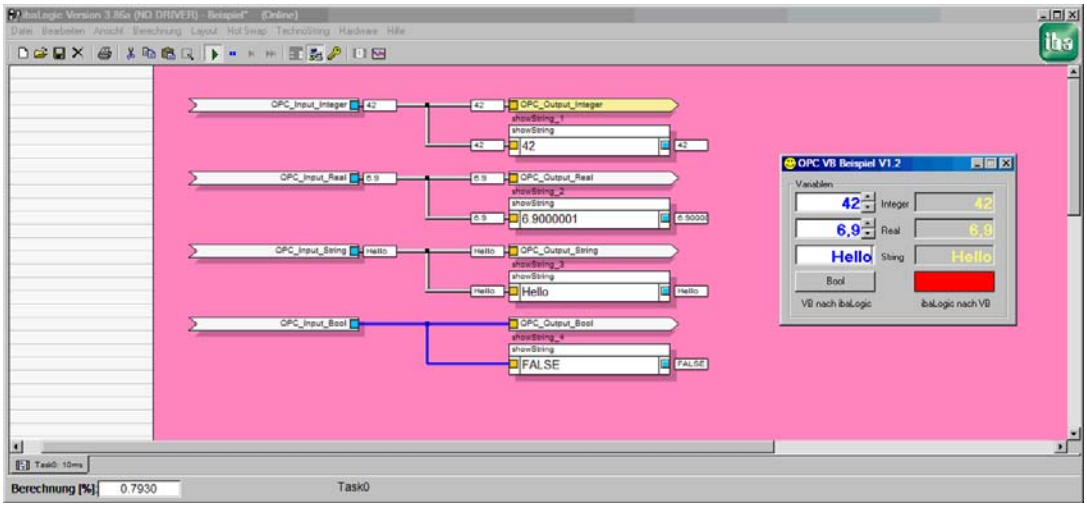


Bild 86 OPC Beispielapplikation, Kontrollfenster

Zur Ausgabe in Visual Basic sind dort folgende "Off-Task-Konnektoren" definiert:

OPC_Output_Integer	Ausgabe INT als Anzeigewert
OPC_Output_Real	Ausgabe Real als Anzeigewert
OPC_Output_String	Ausgabe eines Textfeldes
OPC_Output_Bool	Ausgabe als Binärvariable (hier rot/grün)

In dem Visual Basic – Bedien- und Anzeigefenster können folgende Variable eingegeben und in ibaLogic abgefragt werden:

OPC_Input_Integer	Feld zur Eingabe und Anzeige eines Integerwertes in ibaLogic (Eingabe über Klicks auf Pfeiltasten)
OPC_Input_Real	Feld zur Eingabe und Anzeige eines Realwertes in ibaLogic (Eingabe über Klicks auf Pfeiltasten)
OPC_Input_String	Feld zur Eingabe und Anzeige eines Textes (ASCII-String)
OPC_Input_Bool	Schaltfläche zur binären Umschaltung in ibaLogic

Die folgende Grafik verdeutlicht die unterschiedlichen Einstellungen der verschiedenen Off-Task-Konnektoren:

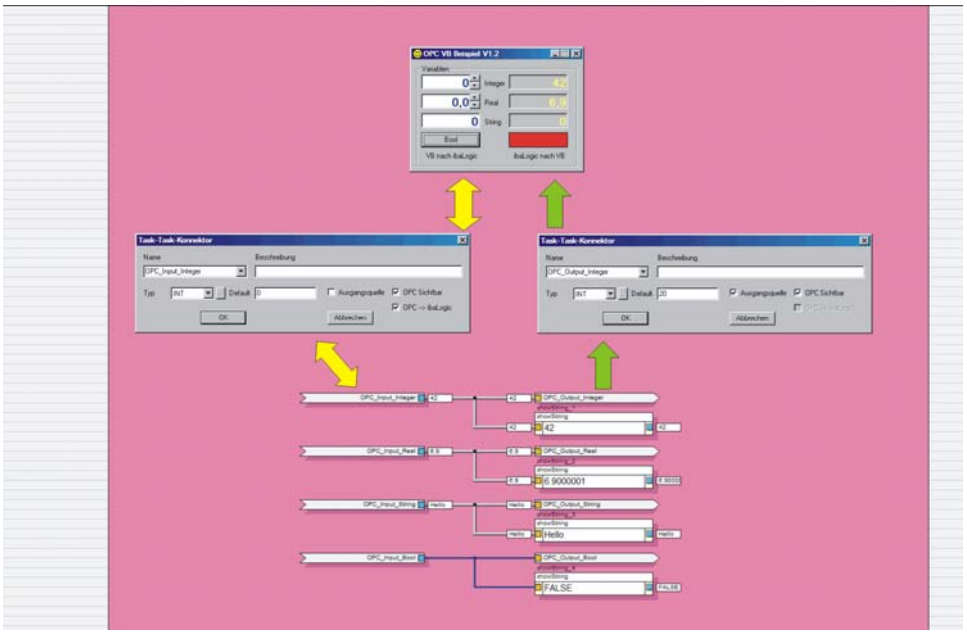
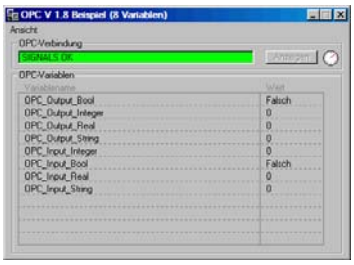
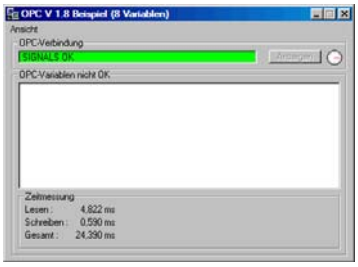
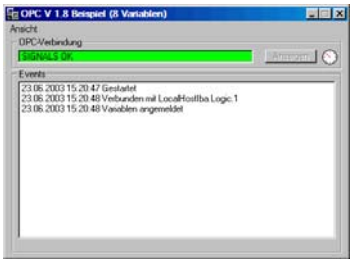


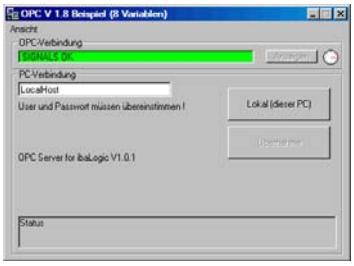
Bild 87 OffTask-Konnektor, Einstellungen für die Kommunikation zwischen ibaLogic und Visual Basic

OPC-Diagnose









Als kleine Hilfe zur OPC-VB-Kommunikation, die zwar nicht Bestandteil von ibaLogic, aber Teil des VB-Beispielprojektes ist, wurde der OPC-Verbindungsmonitor geschaffen.
In verschiedenen Ansichten (wählbar über Menü) erhält man Informationen über
- Anzahl, Namen und Werte der OPC-Variablen,
- Status OK / gestört
- Dauer der Schreib- und Lesezyklen,
- Ereignishistorie und
- PC-Verbindung.

Wenn der OPC-Client (ibaLogic) und der OPC-Master (z.B. HMI) auf unterschiedlichen Rechnern laufen, dann beachten Sie bitte die Sicherheitseinstellungen unter Windows XP. Siehe auch Kapitel 6.2.3.

6 Installation

6.1 Installation von ibaLogic

6.1.1. Installation mit dem Assistenten (nur mit eCon)

1. Legen Sie die Installations-CD ein. Das Installationsprogramm startet automatisch. Sollte dies nicht der Fall sein, dann starten Sie bitte die Datei "Setup.exe" auf der CD.
2. Wählen Sie die gewünschte Sprache. Die Sprachwahl bezieht sich sowohl auf die Installationsdialoge, als auch auf die Dokumentation und die Beispielapplikationen, die auf Ihre Festplatte kopiert werden.
3. Folgen Sie den Anweisungen des Installationsprogramms.
4. Nachdem die Dateien auf die Festplatte kopiert wurden, erscheint ein Dialog, in dem Sie mit Hilfe des Auswahlfeldes (links) den Parallelport bestimmen müssen, an dem ggf. ein **eCon** angeschlossen werden soll. Unter Windows XP können Sie über die Schaltfläche "GeräteManager" feststellen, welche Parallelport-Schnittstelle in Ihrem System zur Verfügung steht.
   Wenn Sie nochmals weitere Hinweise zur Einstellung des Parallelports benötigen, klicken Sie auf die entsprechende Schaltfläche. Verlassen Sie den Dialog mit "Weiter".
5. Im nächsten Dialog wählen Sie die gff. zu verwendenden **eCons** aus. Bei Verwendung von einem **eCon** markieren Sie nur das erste (links), bei Verwendung von zwei Geräten können Sie beide **eCons** alternativ oder gleichzeitig markieren, je nach dem, welches genutzt werden soll. Klicken Sie auf "Weiter".
   Sie können diese Einstellung später jederzeit ändern.
6. Klicken Sie auf "Fertigstellen".

6.1.2. Standard-Installation von CD

1. Erstellen Sie ein Verzeichnis Ihrer Wahl auf der Festplatte (z.B. ibaLogic).
2. Kopieren Sie den gesamten Inhalt des CD-Verzeichnisses *ibaLogic*\ in das entsprechende Verzeichnis auf der Festplatte.
3. Wenn Sie Windows NT verwenden, entfernen Sie bitte das Schreibschutzattribut bei allen Dateien, die kopiert wurden. Bei Windows XP ist dies nicht mehr erforderlich.
4. Wenn Sie ein Update in gezippter Form erhalten oder aus dem Internet geladen haben, dann entpacken Sie alle Dateien aus der *.zip-Datei in dieses Verzeichnis.
5. Starten Sie im Windows-Explorer ibaLogic mit Doppelklick auf die Datei *..\ibaLogic\ibaLogic*Version*.exe* oder nutzen Sie das "Ausführen"-Kommando von Windows. ibaLogic wird automatisch alle erforderlichen Dateien und Unterverzeichnisse erstellen, sofern diese noch nicht vorhanden sein sollten.

Das Verzeichnis \ibaLogic\configuration\schematics ist das Standardverzeichnis für die mit ibaLogic erstellten Programme in der Layout-Form (*.lyt) und in der Structured Text-Form (*.txt).

Das Verzeichnis DLLs beinhaltet später alle DLLs und FBs_Macros alle Funktionsbausteine und Makros, die während der Projektierung erstellt wurden.

Es bleibt Ihnen als Anwender freigestellt, ob Sie sich eine Verknüpfung zum Windows-Desktop einrichten oder ibaLogic in der Programm-Startleiste von Windows unterbringen.

Wenn es sich um einen Projektierungs-Rechner handelt, mag dies Sinn machen, da u.U. auch andere Programme verwendet werden. Bei dem echten Steuerungsrechner sollten ohnehin keine anderen Applikationen neben ibaLogic verwendet werden, so dass hier lediglich der ibaLogic-Aufruf im "Autostart"-Verzeichnis vorgenommen werden sollte.

6.2 USB-Dongle

Aufgrund der stärkeren Verbreitung von USB-Schnittstellen bei den PC-Systemen bietet iba auch einen Kopierschutz für die Software (Dongle) für die USB-Schnittstelle an. Somit kann die serielle Schnittstellen für andere Anwendungen genutzt werden, z.b. für Kommunikation oder USV-Anlage.

Die Unterstützung von USB-Dongles ist unter Windows XP weitgehend automatisiert. Unter Windows NT ist dies meist nicht der Fall, weswegen eine manuelle Installation erforderlich ist.

6.2.1. USB-Dongle unter Windows XP

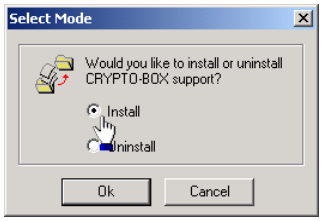
Die von iba verwendeten USB-Dongles werden unter Windows XP automatisch von den iba-Softwareprodukten (wie ibaPDA, ibaLogic, dongleupgrade usw.) installiert.

Eine manuelle Installation ist daher nicht erforderlich.

6.2.2. USB-Dongle unter Windows NT

Wenn ein bestehendes Windows NT-System mit einem USB-Dongle ausgestattet werden soll, dann sind folgende Installtionsschritte auszuführen:

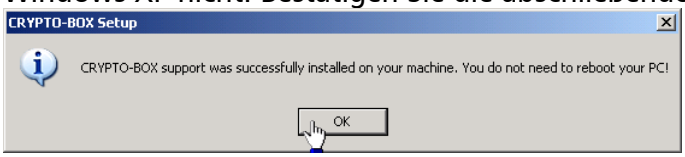
1. Starten Sie das mit dem USB-Dongle gelieferte Programm CBSETUP.exe
2. Wählen Sie Install und anschließend Ok.



3. Wählen Sie Yes / Ja

4 Wählen Sie CRYPTO-BOX USB und anschließend Ok.

5 Je nach Betriebssystem informiert Sie das System, ob ein Reboot des PCs erforderlich ist oder nicht. Bei Windows NT wird ein Reboot verlangt, bei Windows XP nicht. Bestätigen Sie die abschließende Meldung mit Ok



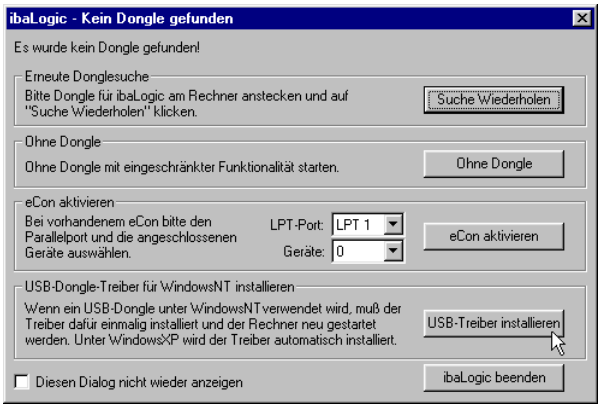
Die Installtionsroutine für die USB-Unterstützung kann auch als Batch-Programm ablaufen. Dazu starten Sie das Installationsprogramm wie folgt:

CbSetup.exe /q /CRYPTOKEN

6

Stellen Sie sicher, dass die USB-Schnittstelle im BIOS des PCs freigegeben bzw. aktiviert ist.

Wenn Sie ibaLogic starten und das Programm keinen Dongle findet, z.B. weil die USB-Unterstützung noch nicht installiert ist, dann erscheint das folgende Dialogfenster:



Klicken Sie einfach auf die Schaltfläche USB-Treiber installieren.

6.2.3. Sicherheitseinstellungen bei Windows XP

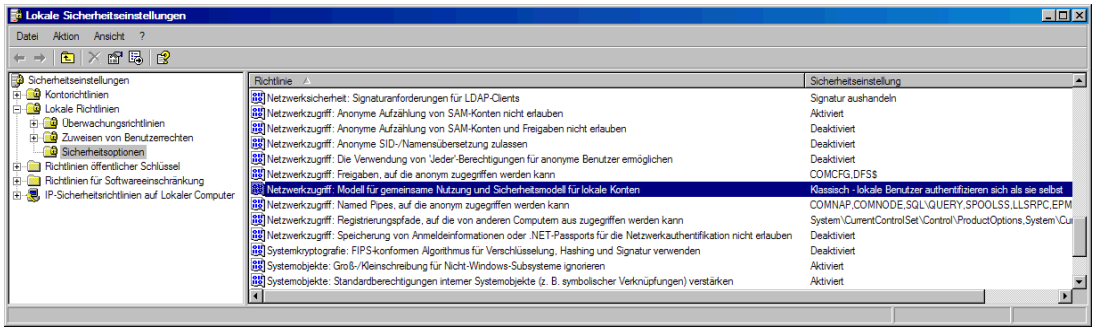
Windows XP verhält sich im Vergleich zu NT noch etwas restriktiver hinsichtlich der Kommunikation zwischen mehreren Rechnern im Netzwerk.

Einige Leistungen der Softwareprodukte von iba nutzen die Vorteile, die eine Vernetzung mehrerer Rechner bietet. Dazu gehören beispielsweise:

- ibaLogic-Verbindungen zwischen OPC-Client und OPC-Server
- Starten von ibaAnalyzer auf einem entfernten Rechner mit Hilfe des PostProcessing-Kommandos im DatFileWrite-Baustein
- Nutzen des PostProcessing-Kommandos in ibaPDA zu entfernten Rechnern
- Ferndiagnose mit ibaDiag

Um die Funktion dieser Dienste zu gewährleisten auch wenn die beteiligten Rechner alle oder teilweise mit Windows XP laufen, sind folgende Einstellungen vorzunehmen:

1. Wenn möglich, sollten Benutzername (Login), Passwort und Berechtigung auf den beteiligten Geräten gleich sein, mindestens jedoch die Berechtigung.
2. Bei unterschiedlichen Logins müssen die Benutzer gegenseitig registriert sein.
3. In den lokalen Sicherheitseinstellungen auf allen beteiligten PCs ist der Parameter *Netzwerkzugriff: Modell für gemeinsame Nutzung und Sicherheitsmodell für lokale Konten* auf **Klassisch** einzustellen. Sie finden die Einstellung in der Windows XP-Systemsteuerung unter ↪Verwaltung ↪Lokale Sicherheitsrichtlinie ↪Sicherheitseinstellungen ↪Lokale Richtlinien ↪Sicherheitsoptionen.



6.3 Systemkonfiguration für ISA-Karten

Definieren Sie nun die Treiberkonfiguration mit dem folgenden Befehl:

Menü ↪*Datei* ↪*ISA-Konfiguration*
(Dieser Menüpunkt ist unter Windows XP nicht verfügbar.)

ibaLogic öffnet das folgende Fenster:

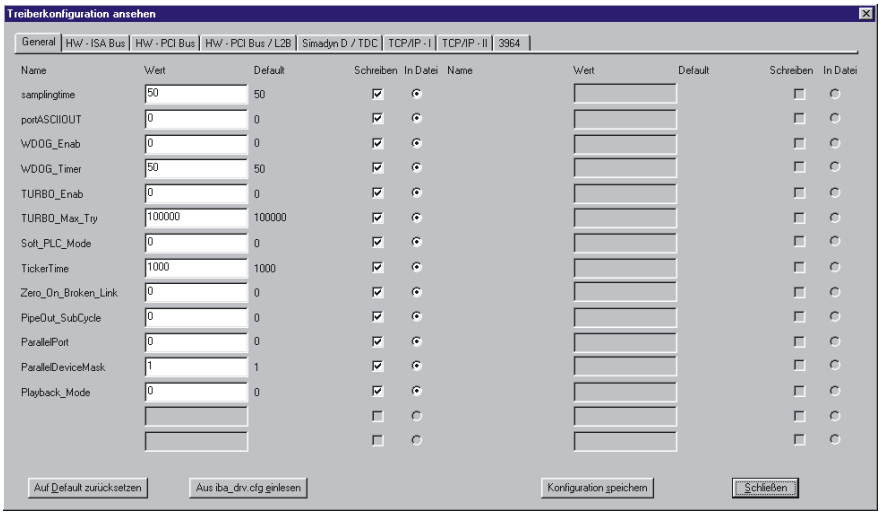


Bild 88 Konfiguration von ISA-Karten

Tragen Sie jetzt die Hardwareparameter ein, die den Brückeneinstellungen der Karte entsprechen. Auch die Grund-Abtastzeit (samplingtime) muss in diesem Register eingestellt werden (Werkseinstellung: 50ms). In unserem Beispiel wird eine FOB-F verwendet. Setzen Sie portFOBF = 1 und stellen Sie FOBF_AcqAddress auf D8000ein. Achten Sie auf die Einstellung des Interrupt Vektors (Int_Vector = 5) !

Achtung: Diese Einstellungen sind nur gültig in Verbindung mit den Brückeneinstellungen auf der Karte. Für den ersten Start von ibaLogic sind nur die Eintragungen im Register 1 von Bedeutung.

Betätigen Sie den Button: "*In iba_drv.cfg speichern*" und ibaLogic erzeugt die Datei **iba_drv.cfg**. Beenden Sie ibaLogic und starten Sie erneut. Die Einstellungen sind erst jetzt wirksam.

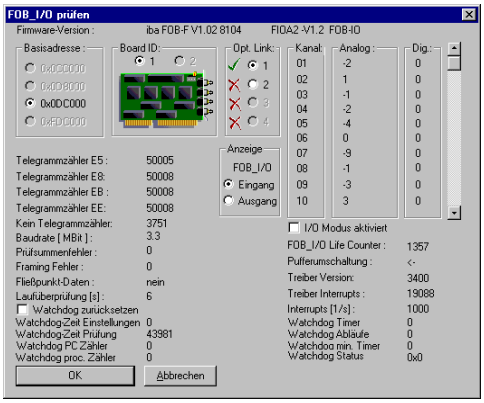


Bild 89 ISA-Karte, Überprüfung der Einstellungen

Überprüfen Sie die Einstellungen mit dem "HARDWARE"-Menü.

Die Hardware arbeitet korrekt, sobald der Interrupt-Zähler einen Wert zwischen 999 und 1001 hat (dies bedeutet 999 bis 1001 Interrupts / s). Wenn ein Padu angeschlossen wird, wechseln die roten Kreuze bei "Opt. Link" zu einem grünem √.

6.3.1. Empfohlene Hardware-Einstellungen

Öffnen Sie das Chassis Ihres Computers. Beachten Sie die Anweisungen in Ihrem PC-Handbuch.

Hardwarekomponenten können durch elektrostatische Entladung dauerhaft beschädigt werden. Verwenden Sie daher die vorgeschriebenen antistatischen Ausrüstungen.

Es gibt einige mögliche Hardwarekonfigurationen. Ein Maximum von drei Karten wird unterstützt. Es ist notwendig, die Hardware entsprechend der folgenden Tabelle zu installieren. Beachten Sie, welche Karte als Interrupt-Master ausgewählt werden muss (unterstrichen und durch Rot gekennzeichnet). Beachten Sie, dass nur 2 IDs pro Kartenadresse unterstützt werden! Beachten Sie ferner, dass die PCMCIA-Karte nicht unterstützt wird! FOB-F kann durch FOB ersetzt werden.

#	Anwendung	Karte 1	Karte 2	Karte 3
1	Simadyn-D-Zugang 2 SD-Verbindungen	FOB SD * CS22 Adresse ist 0xE0000 (meist) ID muss 0 sein	-	-
2	Simadyn-D Zugang 3 oder 4 SD Verbindungen **	FOB SD * CS22 Adresse ist 0xE0000 ID muss 0 sein	FOB-SD 0xE0000; ID = 1 Kaskaden- Stecker nicht vergessen!	FOB-SD Nicht unterstützt
3	Simadyn-D plus (mehrere) FOBs	FOB SD * CS22 Adresse ist 0xE0000 ID muss 0 sein	FOB-F 0xDC000 ID = 0 No IR	FOB-F 0xDC000 ID = 1 No IR
4	Simadyn-D plus Profibus	FOB SD * CS22 Adresse ist 0xE0000 (meist) ID muss 0 sein	L2B-F Adresse: D8000 ID = 0 No IR	
5	Simadyn-D plus Profibus plus FOB	FOB SD * CS22 Adresse ist 0xE0000 (meist) ID muss 0 sein	L2B-F Adresse: D8000 ID = 0 No IR	FOB-F Adresse: DC000 ID = 0 oder 1 No IR
6	Planheits-PC oder Profibus Anwendung	L2B-F *** Adresse: 0xD8000 ID = 0 Interner Interrupt	-	-
7	Planheits-PC und / oder Profibus Anwendung	L2B-F *** Adresse: 0xD8000 ID = 0 Interner Interrupt	L2B-F *** Adresse: 0xD8000 ID = 1 No IR	-
8	Planheit (oder Profibus) plus (mehrere) FOBs	L2B-F *** Adresse: D80000 ID = 0 No IR	FOB-F Adresse: DC000 ID = 0 Externer od. int. IR	FOB-F Adresse: DC000 ID = 1 No IR
9	Mehrere FOBs	FOB-F Adresse: 0xDC000 ID = 0 Externer oder interner IR	FOB-F Adresse: 0xDC000 ID = 1 No IR	FOB-F Nicht unterstützt
10	Notebooks	PCMCIA-F****	IR immer	-

FOB 2/2 IO kann wie eine FOB-F gehandhabt werden. Achtung: Der Interrupt muss auf "Intern" gesetzt sein !

* *Um eine einwandfreie Funktion sicherzustellen (z.B. FOB SD) stellen Sie sicher, dass Segment E nicht benutzt wird, da die FOB-SD Karte das gesamte Segment belegt !*

** *Mehr als eine Verbindung bedeutet, dass ibaLogic auf unterschiedliche CS1x-Module zugreift. Es ist nicht möglich, mehr als einen Anschluss pro CS1x-Modul aufzubauen!*

*** *Überprüfen Sie, ob die Profibus DP-Slaveadresse mit der programmierten Anwendung (z.B. S7) übereinstimmt. Stellen Sie sicher, dass die Betriebsart (Ganzzahl S7, Planheit....) korrekt eingestellt wurde (siehe auch L2B-Handbuch). Nur zwei Planheitsregelungen werden durch ibaLogic und QDA unterstützt! Bei Einstellung als Profibus-DP Slave, fungiert die L2B wie ein FOB-F und muss entsprechend behandelt werden.*

**** *Zur Software-Installation der PCMCIA-F Karte siehe auch PCMCIA_F Handbuch. Wählen Sie nie Adressbereich CC000, da diese Adressen durch das PC-Motherboard bei Onboard-SCSI-Unterstützung in Gebrauch sein könnten!*

6.3.2. Bedeutung der iba_drv.cfg-Konfigurationsdatei

Die von ibaLogic erstellte Konfigurationsdatei "iba_drv.cfg" ist eine ASCII-Datei, die wie folgt gedeutet werden kann, falls Probleme während der Installation auftreten.

```
....
portFOB = 0                     // muss 1 sein, wann immer eine dieser Karten
portFOBF = 1                    // vorhanden ist
portFOBSD = 0
portPROFI = 0
portFOBIO = 0
portASCIIOUT = 0
PCMCIA = 0
....
FOB_AcqAddress = 0xD8000        // Die Adress-Einstellungen werden überprüft, wenn
FOB_AcqLength = 0x440           // das portX = 1 ist (für FOB-Karten)
FOBF_AcqAddress = 0xDC000       // für FOB-F Karten und FOB-IO Karten !!!!
FOBF_AcqLength = 0x3100
FOBSD_AcqAdress = 0xE0000       // die FOB-SD Karte benötigt 64kB Speicher!bitte prü-
fen!
PROFI_AcqAddress = 0xDc000      // für FOB L2B Karten
PROFI_AcqLength = 0x440
PCMCIA = 0                      // für PCMCI-F Karten
CS22_BgtName = PDA001           // die folgenden Parameter gelten nur für FOB-SD
CS22_AcqAddress = 0xD0000       // und CS22
Simadyn_Sync_Timeout = 15
Simadyn_Proc_Timeout = 15
CS22_0_OwnName = DPDA1A         // alle folgenden Parameter müssen für CS22 und
CS22_0_Partner = D0900B         // dementsprechend auch für FOB-SD gesetzt werden
CS22_0_SoftwareVersion = V420
CS22_1_OwnName = DPDA2A
CS22_1_Partner = D0900B
CS22_1_SoftwareVersion = V420
CS22_2_OwnName = DPDA3A
CS22_2_Partner = D1200B
CS22_2_SoftwareVersion = V430
CS22_3_OwnName = DPDA4A
CS22_3_Partner = D1500B
CS22_3_SoftwareVersion = V430
CS22_Nboards = 0          // Die Anzahl der CS22 muss hier gesetzt werden, nicht FOB-
SD!
```

Anmerkung: Zwei FOB-Karten können die gleiche Adresse haben, müssen aber durch zwei unterschiedliche "Board-IDs " gekennzeichnet werden (z.B. 0 und 1 wird durch den Treiber unterstützt).

Bis maximal 2 FOB-Karten können in einen PC installiert werden. Es muss immer eine Karte mit der ID 0 geben, die den Prozess-Interrupt generiert. Folglich müssen Sie entweder angeschlossene PADUs als Interruptquelle auswählen -in diesem Fall den Interrupt-Wahlschalter in Stellung "Außerhalb des PC" schalten - oder Sie verwenden die interne Interruptquelle der FOB-Karte. Dann muss der Schalter in Stellung "PC-intern" stehen. Die erste Option hat den Vorteil, dass die Lichtleiterstrecke und die PADUs mit überwacht werden. Eine defekte Verbindung wird sofort erkannt. Im Fall einer defekten Verbindung erzeugt die FOB-Karte automatisch einen "Default"-Interrupt, aber mit einer viel niedrigeren Frequenz. Wenn ibaLogic merkwürdiger Weise online sehr langsam arbeitet, überprüfen Sie bitte, ob die Verbindung und der PDU in Ordnung sind.

Anmerkung: FOB2/2 IO muss immer mit der internem Interrupt konfiguriert werden.

Wenn Sie unterschiedliche Adressen wählen möchten, vergessen Sie nicht, die Treibereinstellungen entsprechend zu ändern!

Anmerkung: FOB-SD ist eine iba-Karte. Beachten Sie, dass FOB-SD immer einen freien Speicher von 64kB in Ihrem PC benötigt. Die Windows-Diagnose zeigt nicht immer den korrekten Status des freien Speichers an. Es kann vorkommen, dass der angeforderte Speicher als frei markiert ist, obgleich der Block nicht völlig frei ist. Dieses würde ernsthaft den FOB-SD-Betrieb beeinflussen.

Vergessen Sie nicht, alle Simadyn-D Parameter zu überprüfen !

Nach der Installation booten Sie den PC neu und starten ibaLogic.

6.3.3. Systemkonfiguration mit PCI-Karten

Die Hardware-Installation von PCI-Karten ist in der Kartendokumentation detailliert beschrieben.

Wenn die Karten hinsichtlich Steckplatz und PCI-Interrupt korrekt installiert wurden, müssen anschließend in ibaLogic die entsprechenden Einstellungen vorgenommen werden.

Im Menü ↪*Datei* ↪*Systemeinstellungen* zunächst den Button "Autoconfig" betätigen, um die für das System gültigen Grundeinstellungen zu erhalten.

Anschließend sollten unter den einzelnen Registern die von ibaLogic nicht verwendeten Karten abgewählt bzw. die installierten Karten konfiguriert werden.

Die Masken für die Karteneinstellungen können jeweils über den Button "Konfiguration..." (unten rechts) geöffnet werden, bzw. alternativ über ↪*Datei* ↪*PCI-Konfiguration* ↪*Kartentyp*. (vergl. Kapitel 2.5).

Bei Verwendung einer FOB-SD / -TDC sollte diese als Master konfiguriert und als Interrupt-Erzeuger aktiviert sein.

7 Zusatzinformationen und Beispiele

7.1 Beispiel-Listing für DLL-Erstellung

Beachten Sie bitte auch die Hinweise in Kapitel 3.12

Aus drucktechnischen Gründen sind in der folgenden Darstellung der Listings die Programmzeilen z.T. umgebrochen.

7.1.1. dllForm.hpp

```
//*************************************************************
****
//
// Filename:  dllForm.hpp
//
// Author:    Dipl.-Ing. Hubert Andris
//
// Created:   05-Sep-1998
//
// Description:
//   Interface definition for DLL Forms.
//
// External Definitions:
//   DLLExport      compile for DLL export rather than for
DLL import
//
//*************************************************************
****

#if !defined(DLLFORM_HPP)
#define DLLFORM_HPP

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


#include <windows.h>
#include <assert.h>


#if defined(DLLExport)
#define DLL __declspec(dllexport)
#else
#define DLL __declspec(dllimport)
#endif

#define DLL_INTERFACE_VERSION_HIGH  (1 << 16)

////////////////////////////////////////////////////////////
////
// Define function prototypes for DLL functions used.
//
// Naming convention:
//   PF<type><arg1><arg2>...
//
//   where
//     type  = V  returns void
//             I  returns int
//             W  returns DWORD
//             S  returns short
//
//     arg#n = I  arg#n is int
//           = PI arg#n is pointer to unsigned int
//           = PC arg#n is char pointer
//           = PV arg#n is void pointer
//
typedef void  (*PFV)(void);
typedef int   (*PFI)(void);
typedef DWORD (*PFWV)(void);
typedef int   (*PFII)(int io);
typedef short (*PFSII)(int io, int index);
typedef void  (*PFVPV)(void *ptrData);
typedef void  (*PFVPCI)(char *pName, int cbName);
typedef void  (*PFVIIPCI)(int io, int index, char *pName,
int cbName);
typedef void  (*PFVIIPVIPV)(int io, int index, void
*ptrData, int size,
                            void *pInstanceData);
typedef void  (*PFVIPVIIPV)(int index, void *ptrData, int
size, int valid,
                            void *pInstanceData);
typedef int   (*PFIIPVIPV)(int index, void *ptrData, int
size,
                           void *pInstanceData);
typedef void  (*PFVIPVPI)(int cbSize, const void *pGlobal,
                          const __int64 *pMilliSeconds);
typedef void  (*PFVPVIPV)(const void *pGlobal, int cbSize,
                          void *pInstanceData);
typedef DWORD (*PFWPVIPV)(const void *pGlobal, int cbSize,
                          void *pInstanceData);

#define MAX_SERIAL_NO_LENGTH  12

typedef struct
{
  __int64 g_EvalTime;
  __int64 g_EvalDeltaTime;
  int     g_Online;
  int     g_Unlocked;
  DWORD   g_System1;
  char    g_DongleSerialNum[MAX_SERIAL_NO_LENGTH];
} globalVarType;

#if 0
////////////////////////////////////////////////////////////
////
// DllMain
//
// Called from operating system on load and unload.
//
extern DLL BOOL WINAPI DllMain(HINSTANCE hInstDLL,    //
handle to DLL module
                               DWORD     fdwReason,   //
reason for calling function
                               LPVOID    lpReserved); // re-
served
#endif

////////////////////////////////////////////////////////////
////
// GetDllVersion
//
// Each Dll shall have a unique version number
//
// Returns:
//   hiword  major version number: describes the program
// interface
//   loword  minor version number: describes the semantic
//
extern DLL DWORD GetDllVersion(void);

////////////////////////////////////////////////////////////
////
// General parameters for subsequent functions:
//-------------------------------------------
//
// Each instance of the Dll form has its own data pointer,
which
// points to
// dynamicaly allocated memory of size returned by the pre-
vious
// call to
// GetInstanceDynamicDataSize().
//
// Parameter:
//   void *pInstanceData
//
//-----------------------------------------------------------
----
//
// The Dll has read-only access to global variables of Sig-
nal
// Manager
// application, which may be required for the evaluation.
//
// Currently defined global variables:
//
//+--------+-------+---------+-----------------------------
----+
//| offset | bytes | type    | description
|
//|--------+-------+---------+-----------------------------
----+
//|    0   |   8   | __int64 | time in 0.1 milliseconds
|
//|        |       |         | relative to last
|
//|        |       |         | InitEvaluation call
|
//+--------+-------+---------+-----------------------------
----+
//|    8   |   8   | __int64 | time in 0.1 milliseconds
|
//|        |       |         | relative to last
|
//|        |       |         | call to Evaluate
|
//+--------+-------+---------+-----------------------------
----+
//|   16   |   1   |  char   | = 0: layer is offline/
|
//|        |       |         | = 1: layer is online
|
//+--------+-------+---------+-----------------------------
----+
```

```
//|   17   |   3   |  none   | unused (all bytes 0)
|
//+--------+-------+---------+-----------------------------
----+
//|   20   |   1   |  char   | = 0: layer is locked/
|
//|        |       |         | = 1: layer is unlocked
|
//|        |       |         | If locked, DLL shall NOT
change  |
//|        |       |         | any default value!
|
//+--------+-------+---------+-----------------------------
----+
//|   21   |   3   |  none   | unused (all bytes 0)
|
//+--------+-------+---------+-----------------------------
----+
//|   24   |   4   |  DWORD  | reserved
|
//+--------+-------+---------+-----------------------------
----+
//|   28   |  12   |  char   | index 0 .. 7  : serial number
|
//|        |       |         | index 8       : = 0
|
//|        |       |         | index 9 .. 11 : reserved
|
//+--------+-------+---------+-----------------------------
----+
//
// Parameter:
//   pGlobal        pointer to global variables array
//   cbSize         number of available global variables
//
////////////////////////////////////////////////////////////
////


////////////////////////////////////////////////////////////
////
// GetInstanceDynamicDataSize
//
// Each instance of the Dll formula has its own pointer,
which
// may point to
// any data of any size. This function returns the size for
that
// memory.
// Memory allocation/deallocation is completely performed by
the // calling
// application.
//
// Returns:
//   required memory size in bytes
//
extern DLL int GetInstanceDynamicDataSize(void);


////////////////////////////////////////////////////////////
////
// GetDllDescription
//
// Parameter:
//   pDesc    buffer to receive description as ASCIIZ string
//   cbDesc   size of buffer (including terminating Null
byte)
//
extern DLL void GetDllDescription(char *pDesc, int cbDesc);

////////////////////////////////////////////////////////////
////
// GetCount
//
// Parameter:
//   io  0 = input, 1 = output
//
// Returns:
//   number of inputs/outputs (1..128)
//
extern DLL int GetCount(int io);


////////////////////////////////////////////////////////////
////
// GetName
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pName    buffer to receive name as ASCIIZ string
//   cbName   size of buffer (including terminating Null
byte)
//
// Note: Each input/output name must to be unique!
//
extern DLL void GetName(int io, int index, char *pName, int
cbName);


////////////////////////////////////////////////////////////
////
// GetDescription
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pDesc    buffer to receive description as ASCIIZ string
//   cbDesc   size of buffer (including terminating Null
byte)
//
extern DLL void GetDescription(int io, int index, char
*pDesc, int cbDesc);

////////////////////////////////////////////////////////////
////
// GetType
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//
// Returns:
//+-------+--------------+---------------------------------
---+
//| type  | iec1131 type | size of value in bytes
|
//+-------+--------------+---------------------------------
---+
//|   1   |    BOOL      |      1
|
//|   3   |    INT       |      2
|
//|   4   |    DINT      |      4
|
//|   8   |    UDINT     |      4
|
//|  10   |    REAL      |      4
|
//|  11   |    LREAL     |      8
|
//|  12   |    TIME      |      8
|
//|  16   |    STRING    |   1024 (including terminating
null) |
//|  19   |    DWORD     |      4
|
//|  22   |    ARRAY     | total element count * size
|
//|       |              | of element
|
//+-------+--------------+---------------------------------
---+
//
// Note:
//   Enumeration valType in 'value.hpp' can be used for
//   convenience.
//   If 22 (ARRAY) is returned, then a call to GetArray-
Header
//   will follow.
//
extern DLL WORD GetType(int io, int index);

////////////////////////////////////////////////////////////
////
// GetArrayHeader
//
//   Called if GetType() returned 22 (array type)
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pBuffer  pointer to array header:
//+--------+------+-----------------------------------------
---+
//| Offset | Type | Description
|
//+--------+------+-----------------------------------------
---+
//|  0     | WORD | Element type (see description of Get-
Type)  |
//|  2     | BYTE | reserved (e.g. for synchronization flag)
|
//|  3     | BYTE | reserved (e.g. for valid flag)
|
//|  4     | DWORD| array dimension (1..4)
|
//|  8     | DWORD| Start Index of 0-th subscript
|
//| 12     | DWORD| Stop  Index of 0-th subscript
|
//| ...    |      |
|
//|  8+n*8 | DWORD| Start Index of n-th subscript
|
//|        |      | (n < dimension)
|
//| 12+n*8 | DWORD| Stop  Index of n-th subscript
|
//|        |      | (n < dimension)
|
//| ...    |      |
|
//+--------+------+-----------------------------------------
---+
//
//   cbBuf    sizeof buffer in bytes sufficient for max ar-
ray
//   dimension of 4
//
// Note: These functions are only called once for initial
//       initialization of
//       the dll form instance.
//       The total element count is calculated via:
//       Sum over all subscripts: (n-th stop index - n-th
start
//       index + 1).
```

```
//       For multi-dimensional arrays, the rightmost sub-
script is
//       varying
//       most rapidly with respect to the address offset of
the
//       element value.
//       e.g.: a : ARRAY[1..2,2..4] OF INT;
//           total element count = (2 - 1 + 1) + (4 - 2 + 1)
= 6
//               +---------+-----------------+
//               | element | offset in bytes |
//               +---------+-----------------+
//               | a[1,2]  |    0 * 2 =  0   |
//               | a[1,3]  |    1 * 2 =  2   |
//               | a[1,4]  |    2 * 2 =  4   |
//               | a[2,2]  |    3 * 2 =  6   |
//               | a[2,3]  |    4 * 2 =  8   |
//               | a[2,4]  |    5 * 2 = 10   |
//               +---------+-----------------+
//       For arrays, the only element types permitted are:
//       BOOL, INT, DINT, UDINT, REAL, LREAL, TIME and
DWORD.
//
extern DLL void GetArrayHeader(int io, int index, void
*pBuffer, int cbBuf);


////////////////////////////////////////////////////////
////
// GetDefaultValue
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pBuffer  pointer to buffer to receive value(s):
//+-------+-------------+-----------------------------------
---+
//| type  | iec1131 type | size of buffer in bytes
|
//+-------+-------------+-----------------------------------
---+
//|   1   |    BOOL      |      1
|
//|   3   |    INT       |      2
|
//|   4   |    DINT      |      4
|
//|   8   |    UDINT     |      4
|
//|  10   |    REAL      |      4
|
//|  11   |    LREAL     |      8
|
//|  12   |    TIME      |      8
|
//|  16   |    STRING    |   1024 (including terminating
null) |
//|  19   |    DWORD     |      4
|
//|  22   |    ARRAY     | total element count * size
|
//|       |              | of element
|
//+-------+-------------+-----------------------------------
---+
//   cbBuf    sizeof buffer in bytes
//   pInstanceData  see description above
//
// Note: These functions are called once for initial
//       initialization of the dll
//       form instance (pInstanceData is NULL).
//       For outputs, this function will be called if Evalu-
ate()
//       exited with
//       bit0 of return value = 1 (pInstanceData is not
NULL).
//       For arrays, the only element types permitted are:
//       BOOL, INT, DINT, UDINT, REAL, LREAL, TIME and
DWORD.
//
extern DLL void GetDefaultValue(int io, int index, void
*pBuffer, int cbBuf,
                                void *pInstanceData);


////////////////////////////////////////////////////////
////
// SetInputValue
//
// This function is called once for each input index before
each
// evaluation.
//
// Parameter:
//   index    0..GetCount(1)-1
//   pBuffer  pointer to buffer to read value(s)
//            It has the same format as described in
//            GetDefaultValue
//   cbBuf    sizeof buffer in bytes
//   bValid   valid flag of value: 0 = valid, not 0 = inva-
lid
//
extern DLL void SetInputValue(int index, void *pBuffer, int
cbBuf, BOOL bValid,
                              void *pInstanceData);


////////////////////////////////////////////////////////
////
// GetOutputValue
//
// This function is called once for each output index after
each
// evaluation.
//
// Parameter:
//   index           0..GetCount(1)-1
//   pBuffer         pointer to buffer to read value(s)
//                   It has the same format as described in
//                   GetDefaultValue
//   cbBuf           sizeof buffer in bytes
//   pInstanceData   see description above
//
// Returns:
//   0        if output value is invalid
//   != 0     if output value is valid
//
extern DLL BOOL GetOutputValue(int index, void *pBuffer, int
cbBuf,
                               void *pInstanceData);


////////////////////////////////////////////////////////
////
// InitEvaluation
//
// Do specific initialization of any local data.
// Values are already set to their defaults.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
extern DLL void InitEvaluation(const void *pGlobal, int
cbSize,
                               void *pInstanceData);


////////////////////////////////////////////////////////
////
// Evaluate
//
// Calculate all output values and their corresponding valid
// bits.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
// Returns:
//   Bit0     = 1: DLL has changed default values
//                 (SET_DEFAULT function)
//                causes call to GetDefaultValue() for all
I/O's
//   Bit1..31 = 0 (reserved)
//
extern DLL DWORD Evaluate(const void *pGlobal, int cbSize,
                          void *pInstanceData);

////////////////////////////////////////////////////////
////
// ExitEvaluation
//
// Called immediately before the dll form instance is re-
moved.
// Do any form instance specific cleanup.
//
// E.g.: The memory set by SetInstanceDataPointer() may con-
taion
//       another pointer to dynamically allocated memory in
//       InitEvaluation(). Here is the point to deallocate
that
//       memory.
//
// Note: Do NOT deallocate the memory set by
//       SetInstanceDataPointer()! This
//       memory is managed by the calling application.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
extern DLL void ExitEvaluation(const void *pGlobal, int
cbSize,
                               void *pInstanceData);

#endif // DLLFORM_HPP
```

7.1.2. SampleDLL.cpp

```
//*********************************************************
****
//
// Filename:  sampleDll.cpp
//
// Author:    Dipl.-Ing. Hubert Andris
//
// Created:   05-Sep-1998
//
// Description:
//   Required definitions for specific DLL Formulas.
//
// External Definitions:
//   DLLExport      compile for DLL export rather than for
DLL
//                 import
//
//*********************************************************
****

#define DLLExport

#include "dllForm.hpp"

#define NUM_INPUTS  2
#define NUM_OUTPUTS 3
#define ARRAY_SIZE  10

static int nInstance = 0;

////////////////////////////////////////////////////////////
////
// data structure used for formula instance specific data
typedef struct dynamicData
{
  struct dynamicData()
  { memset(this, 0, sizeof(*this)); }

  BOOL  inputValueValid[NUM_INPUTS];
  float inputValue[NUM_INPUTS][ARRAY_SIZE];
  float inputDefaultValue[NUM_INPUTS][ARRAY_SIZE];

  BOOL  outputValueValid[NUM_OUTPUTS];
  float outputValue[1][ARRAY_SIZE];
  float outputDefaultValue[1][ARRAY_SIZE];

  char DongleId[9];

  int   nInstance;
} dynamicDataType;

////////////////////////////////////////////////////////////
////
// Common data for all instances
static const char strDescription[] = "Sample DLL for iba-
LogIC V1.2";
static const char strDongleDefault[] = "none";

static const char inputName[NUM_INPUTS][32] =
{
  "a",
  "b",
};

static const char outputName[NUM_OUTPUTS][32] =
{
  "out",
  "instance",
  "dongleNumber",
};

static const char inputDescription[NUM_INPUTS][32] =
{
  "1st input array",
  "2nd input array",
};

static const char outputDescription[NUM_OUTPUTS][32] =
{
  "dot product",
  "instance number",
  "Dongle Number",
};


////////////////////////////////////////////////////////////
////
// DllMain
//
// Called from operating system on load and unload.
//
DLL BOOL WINAPI DllMain(HINSTANCE hInstDLL,    // handle to
DLL module
                        DWORD     fdwReason,   // reason for
calling function
                        LPVOID    lpReserved)  // reserved
{
  BOOL bRet = FALSE;

  // Perform actions based on the reason for calling.
  switch( fdwReason )
  {
    case DLL_PROCESS_ATTACH:
      // Initialize once for each new process.
      // Return FALSE to fail DLL load.
      bRet = TRUE; // Successful DLL_PROCESS_ATTACH.
      break;

    case DLL_THREAD_ATTACH:
      // Do thread-specific initialization.
      bRet = TRUE; // Successful DLL_THREAD_ATTACH.
      break;

    case DLL_THREAD_DETACH:
      // Do thread-specific cleanup.
      bRet = TRUE; // Successful DLL_THREAD_DETACH.
      break;

    case DLL_PROCESS_DETACH:
      // Perform any necessary cleanup.
      bRet = TRUE; // Successful DLL_PROCESS_DETACH.
      break;
  }

  return bRet;
}

////////////////////////////////////////////////////////////
////
// GetDllVersion
//
// Each Dll shall have a unique version number
//
// Returns:
//   hiword  major version number: describes the program
//                                 interface
//   loword  minor version number: describes the semantic
//
DLL DWORD GetDllVersion(void)
{
  return (DLL_INTERFACE_VERSION_HIGH | 1);
}

////////////////////////////////////////////////////////////
////
// General parameters for subsequent functions:
//---------------------------------------------
//
// Each instance of the Dll form has its own data pointer,
// which points to dynamicaly allocated memory of size
// returned by the previous call to
// GetInstanceDynamicDataSize().
//
// Parameter:
//   void *pInstanceData
//
//-----------------------------------------------------------
----
//
// The Dll has read-only access to global variables of Sig-
nal
// Manager
// application, which may be required for the evaluation.
//
// Currently defined global variables:
//
//+--------+-----------------------------------------------
---+
//| offset | bytes | type    | description
|
//|--------+-------+---------+-----------------------------
---+
//|    0   |   8   | __int64 | time in 0.1 milliseconds
|
//|        |       |         | relative to last
|
//|        |       |         | InitEvaluation call
|
//+--------+-------+---------+-----------------------------
---+
//|    8   |   8   | __int64 | time in 0.1 milliseconds
|
//|        |       |         | relative to last
|
//|        |       |         | call to Evaluate
|
//+--------+-------+---------+-----------------------------
---+
//|   16   |   1   |  char   | = 0: layer is offline/
|
//|        |       |         | = 1: layer is online
|
//+--------+-------+---------+-----------------------------
---+
//|   17   |   3   |  none   | unused (all bytes 0)
|
//+--------+-------+---------+-----------------------------
---+
//|   20   |   1   |  char   | = 0: layer is locked/
|
//|        |       |         | = 1: layer is unlocked
|
//|        |       |         | If locked, DLL shall NOT
change |
//|        |       |         | any default value!
|
//+--------+-------+---------+-----------------------------
---+
//|   21   |   3   |  none   | unused (all bytes 0)
|
//+--------+-------+---------+-----------------------------
---+
//|   24   |   4   |  DWORD  | reserved
|
```

```
//+--------+-------+---------+-----------------------------
---+
//
// Parameter:
//   pGlobal        pointer to global variables array
//   cbSize         number of available global variables
//
////////////////////////////////////////////////////////////
////

////////////////////////////////////////////////////////////
////
// GetInstanceDynamicDataSize
//
// Each instance of the Dll formula has its own pointer,
// which may point to any data of any size. This function
// returns the size for that memory.
// Memory allocation/deallocation is completely performed by
// the calling application.
//
// Returns:
//   required memory size in bytes
//
DLL int GetInstanceDynamicDataSize(void)
{
  return sizeof(dynamicDataType);
}

////////////////////////////////////////////////////////////
////
// GetDllDescription
//
// Parameter:
//   pDesc    buffer to receive description as ASCIIZ string
//   cbDesc   size of buffer (including terminating Null
byte)
//
DLL void GetDllDescription(char *pDesc, int cbDesc)
{
  strncpy(pDesc, strDescription, cbDesc);
}

////////////////////////////////////////////////////////////
////
// GetCount
//
// Parameter:
//   io  0 = input, 1 = output
//
// Returns:
//   number of inputs/outputs (1..128)
//
DLL int GetCount(int io)
{
  int iRet;

  if ( io == 0 )
  {
    iRet = NUM_INPUTS;
  }
  else
  {
    iRet = NUM_OUTPUTS;
  }

  return iRet;
}

////////////////////////////////////////////////////////////
////
// GetName
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pName    buffer to receive name as ASCIIZ string
//   cbName   size of buffer (including terminating Null
byte)
//
// Note: Each input/output name must to be unique!
//
DLL void GetName(int io, int index, char *pName, int cbName)
{
  if ( io == 0 )
  {
    strncpy(pName, inputName[index], cbName);
  }
  else
  {
    strncpy(pName, outputName[index], cbName);
  }
}

////////////////////////////////////////////////////////////
////
// GetDescription
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pDesc    buffer to receive description as ASCIIZ string
//   cbDesc   size of buffer (including terminating Null
byte)
//
DLL void GetDescription(int io, int index, char *pDesc, int
cbDesc)
{
  if ( io == 0 )
  {
    strncpy(pDesc, inputDescription[index], cbDesc);
  }
  else
  {
    strncpy(pDesc, outputDescription[index], cbDesc);
  }
}

////////////////////////////////////////////////////////////
////
// GetType
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//
// Returns:
//+-------+--------------+---------------------------------
---+
//| type  | iec1131 type | size of value in bytes
|
//+-------+--------------+---------------------------------
---+
//|   1   |    BOOL      |      1
|
//|   3   |    INT       |      2
|
//|   4   |    DINT      |      4
|
//|   8   |    UDINT     |      4
|
//|  10   |    REAL      |      4
|
//|  11   |    LREAL     |      8
|
//|  12   |    TIME      |      8
|
//|  16   |    STRING    |   1024 (including terminating
null) |
//|  19   |    DWORD     |      4
|
//|  22   |    ARRAY     | total element count * size of
|
//|       |              | element
|
//+-------+--------------+---------------------------------
---+
//
// Note:
//   Enumeration valType in 'value.hpp' can be used for
//   convenience.
//   If 22 (ARRAY) is returned, then a call to GetArray-
Header
//   will follow.
//
DLL WORD GetType(int io, int index)
{
  WORD wRet = 22; // all inputs/outputs are arrays

  if ( io == 1 )
  {
    if ( index == 1 )
      wRet = 4;
    if ( index == 2 )
      wRet = 16;
    if ( index == 3 )
      wRet = 1;
  }

  return wRet;
}

////////////////////////////////////////////////////////////
////
// GetArrayHeader
//
//   Called if GetType() returned 22 (array type)
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pBuffer  pointer to array header:
//+--------+------+-----------------------------------------
---+
//| Offset | Type | Description
|
//+--------+------+-----------------------------------------
---+
//|  0     | WORD | Element type (see description of Get-
Type)  |
//|  2     | BYTE | reserved (e.g. for synchronization flag)
|
//|  3     | BYTE | reserved (e.g. for valid flag)
|
//|  4     | DWORD| array dimension (1..4)
|
//|  8     | DWORD| Start Index of 0-th subscript
|
//| 12     | DWORD| Stop  Index of 0-th subscript
|
//| ...    |      |
|
//|  8+n*8 | DWORD| Start Index of n-th subscript
|
```

```
//|        |      | (n < dimension)
|
//| 12+n*8 | DWORD| Stop  Index of n-th subscript
|
//|        |      | n < dimension)
|
//| ...    |      |
|
//+--------+------+-----------------------------------------
---+
//
//   cbBuf    sizeof buffer in bytes sufficient for max ar-
ray
//            dimension of 4
//
// Note: These functions are only called once for initial
//       initialization of the dll form instance.
//       The total element count is calculated via:
//       Sum over all subscripts: (n-th stop index - n-th
start
//       index + 1).
//       For multi-dimensional arrays, the rightmost sub-
script
//       is varying most rapidly with respect to the address
//       offset of the element value.
//       e.g.: a : ARRAY[1..2,2..4] OF INT;
//           total element count = (2 - 1 + 1) + (4 - 2 + 1)
= 6
//               +---------+-----------------+
//               | element | offset in bytes |
//               +---------+-----------------+
//               | a[1,2]  |    0 * 2 =  0   |
//               | a[1,3]  |    1 * 2 =  2   |
//               | a[1,4]  |    2 * 2 =  4   |
//               | a[2,2]  |    3 * 2 =  6   |
//               | a[2,3]  |    4 * 2 =  8   |
//               | a[2,4]  |    5 * 2 = 10   |
//               +---------+-----------------+
//       For arrays, the only element types permitted are:
//       BOOL, INT, DINT, UDINT, REAL, LREAL, TIME and
DWORD.
//
DLL void GetArrayHeader(int io, int index, void *pBuffer,
int cbBuf)
{ // all inputs/outputs are arrays of the same type
  ((WORD *)  pBuffer)[0] = 10;             // element type
is REAL
  ((DWORD *) pBuffer)[1] = 1;              // dimension
  ((DWORD *) pBuffer)[2] = 0;              // start index 0-
th subscript
  ((DWORD *) pBuffer)[3] = ARRAY_SIZE - 1; // stop index 0-
th subscript
}


////////////////////////////////////////////////////////
////
// GetDefaultValue
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pBuffer  pointer to buffer to receive value(s):
//+-------+-------------+---------------------------------
---+
//| type  | iec1131 type | size of buffer in bytes
|
//+-------+-------------+---------------------------------
---+
//|   1   |    BOOL     |      1
|
//|   3   |    INT      |      2
|
//|   4   |    DINT     |      4
|
//|   8   |    UDINT    |      4
|
//|  10   |    REAL     |      4
|
//|  11   |    LREAL    |      8
|
//|  12   |    TIME     |      8
|
//|  16   |    STRING   |   1024 (including terminating
null) |
//|  19   |    DWORD    |      4
|
//|  22   |    ARRAY    | total element count * size of
|
//|       |             | element
|
//+-------+-------------+---------------------------------
---+
//   cbBuf    sizeof buffer in bytes
//   pInstanceData  see description above
//
// Note: These functions are called once for initial
//       initialization of the dll form instance
//       (pInstanceData is NULL).
//       For outputs, this function will be called if
//       Evaluate() exited with bit0 of return value = 1
//       (pInstanceData is not NULL).
//       For arrays, the only element types permitted are:
//       BOOL, INT, DINT, UDINT, REAL, LREAL, TIME and
DWORD.
//
DLL void GetDefaultValue(int io, int index, void *pBuffer,
int cbBuf,
                         void *pInstanceData)
{ // all inputs/outputs same type and defaults
  int i;

  if ( (io == 1) && (index == 1) )
  {
    *((int *) pBuffer) = 0;
  }
  else if ( (io == 1) && (index == 2) )
  {
    strcpy((char *) pBuffer,strDongleDefault);
  }
  else if ( (io == 1) && (index == 3) )
  {
    *((BOOL *) pBuffer) = FALSE;
  }
  else
  {
    for ( i = 0; i < ARRAY_SIZE; ++i )
    { // for multiplication, default 1 is convenient
      ((float *) pBuffer)[i] = 1.0;
    }
  }
}


////////////////////////////////////////////////////////
////
// SetInputValue
//
// This function is called once for each input index before
// each evaluation.
//
// Parameter:
//   index    0..GetCount(1)-1
//   pBuffer  pointer to buffer to read value(s)
//            It has the same format as described in
//            GetDefaultValue
//   cbBuf    sizeof buffer in bytes
//   bValid   valid flag of value: 0 = valid, not 0 = inva-
lid
//
DLL void SetInputValue(int index, void *pBuffer, int cbBuf,
BOOL bValid,
                       void *pInstanceData)
{
  dynamicDataType *pData = (dynamicDataType *) pInstance-
Data;

  if ( pData != NULL )
  {
    pData->inputValueValid[index] = bValid;

    assert(cbBuf == sizeof(pData->inputValue[index]));
    memcpy(pData->inputValue[index], pBuffer, sizeof(pData-
>inputValue[index]));
  }
}


////////////////////////////////////////////////////////
////
// GetOutputValue
//
// This function is called once for each output index after
// each evaluation.
//
// Parameter:
//   index           0..GetCount(1)-1
//   pBuffer         pointer to buffer to read value(s)
//                   It has the same format as described in
//                   GetDefaultValue
//   cbBuf           sizeof buffer in bytes
//   pInstanceData   see description above
//
// Returns:
//   0        if output value is invalid
//   != 0     if output value is valid
//
DLL BOOL GetOutputValue(int index, void *pBuffer, int cbBuf,
                        void *pInstanceData)
{
  BOOL bRet = FALSE; // default invalid
  dynamicDataType *pData = (dynamicDataType *) pInstance-
Data;

  assert(pData != NULL);
  if ( pData != NULL )
  {
    if ( pData->outputValueValid[index] )
    {
      switch (index )
      {
        case 0:
          memcpy(pBuffer, pData->outputValue[index],
sizeof(pData->outputValue[index]));
          break;
        case 1:
          memcpy(pBuffer, &pData->nInstance, sizeof(pData-
>nInstance));
          break;
        case 2:
          memcpy(pBuffer, &pData->DongleId,
9);//strlen(pData->DongleId));
          break;
        default:
          pData->outputValueValid[index] = FALSE;
      }

      bRet = TRUE;
```

```
    }
  }

  return bRet;
}


////////////////////////////////////////////////////////
////
// InitEvaluation
//
// Do specific initialization of any local data.
// Values are already set to their defaults.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
DLL void InitEvaluation(const void *pGlobal, int cbSize,
void *pInstanceData)
{
  globalVarType   *pGlobals = (globalVarType *)   pGlobal;
  dynamicDataType *pData    = (dynamicDataType *) pInstance-
Data;

  assert(pGlobals != NULL);
  assert(pData    != NULL);

  if ( pData != NULL )
  {
    memcpy(pData->DongleId,pGlobals->g_DongleSerialNum,9);
    pData->DongleId[8]=0;
//    pData->DongleHasLogic = DongleHasLogic();
    pData->nInstance = nInstance++;

    pData->outputValueValid[1] = TRUE;
    pData->outputValueValid[2] = TRUE;
    pData->outputValueValid[3] = TRUE;
    if ( memcmp(pData->DongleId,"999999",6) != 0 )
    {
      pData->outputValueValid[0] = FALSE;
    }
  }
}


////////////////////////////////////////////////////////
////
// Evaluate
//
// Calculate all output values and their corresponding valid
// bits.
// In this sample DLL each output value has a corresponding
value
// in outputValueValid to indicate if the value is valid or
// invalid.
// If outputValueValid[] is FALSE, GetOutputValue() will re-
turn
// FALSE to ibaLogic, the Output will be marked as invalid
and
// the data will not be updated.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
// Returns:
//   Bit0     = 1: DLL has changed default values
//                 (SET_DEFAULT function)
//                 causes call to GetDefaultValue() for all
I/O's
//   Bit1..31 = 0 (reserved)
//
DLL DWORD Evaluate(const void *pGlobal, int cbSize, void
*pInstanceData)
{
  globalVarType   *pGlobals = (globalVarType *)   pGlobal;
  dynamicDataType *pData    = (dynamicDataType *) pInstance-
Data;

  assert(pGlobals != NULL);
  assert(pData    != NULL);

  if ( pData != NULL )
  {
    memcpy(pData->DongleId,pGlobals->g_DongleSerialNum,9);
    pData->DongleId[8]=0;
    if ( memcmp(pData->DongleId,"00999999",8) != 0 )
    {
      pData->outputValueValid[0] = FALSE;
    }
    else
    {
      pData->outputValueValid[0] = pData->inputValueValid[0]
&& pData->inputValueValid[1];
    }

    if ( pData->outputValueValid[0] )
    {
      int i;

      for ( i = 0; i < ARRAY_SIZE; ++i )
      {
        pData->outputValue[0][i] = pData->inputValue[0][i] *
pData->inputValue[1][i];
      }
    }
  }

  return 0;
}

////////////////////////////////////////////////////////
////
// ExitEvaluation
//
// Called immediately before the dll form instance is re-
moved.
// Do any form instance specific cleanup.
//
// E.g.: The memory set by SetInstanceDataPointer() may con-
taion
//       another pointer to dynamically allocated memory in
//       InitEvaluation(). Here is
//       the point to deallocate that memory.
//
// Note: Do NOT deallocate the memory set by
//       SetInstanceDataPointer()! This memory is managed by
//       the calling application.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
DLL void ExitEvaluation(const void *pGlobal, int cbSize,
void *pInstanceData)
{
  globalVarType   *pGlobals = (globalVarType *)   pGlobal;
  dynamicDataType *pData    = (dynamicDataType *) pInstance-
Data;

  assert(pGlobals != NULL);
  assert(pData    != NULL);
  assert(pData->nInstance >= 0);
}
```

7.1.3. SampleDLL.def

```
LIBRARY sampleDll

VERSION     1.2
DESCRIPTION "Sample form DLL"

EXETYPE WINDOWS

EXPORTS
  DllMain                   @1

  GetDllVersion             @2
  GetInstanceDynamicDataSize @3

  GetCount                  @4

  GetDllDescription         @5

  GetName                   @6
  GetDescription            @7

  GetType                   @8
  GetArrayHeader            @9

  GetDefaultValue           @10

  SetInputValue             @11
  GetOutputValue            @12

  InitEvaluation            @13
  Evaluate                  @14
  ExitEvaluation            @15
```

7.2 Liste der für ibaLogic reservierten Namen

Es gibt einige Namen für Funktionen und Prozeduren, die exklusiv für ibaLogic reserviert sind. Versucht man diese Namen für neue FBs, Konnektoren vosn FBs, OTC, IPCs, Makrobausteine oder Tasks verwenden, dann erscheint eine Fehlermeldung.

Um diese Konflikte zu vermeiden, berücksichtigen Sie bitte die folgende Tabelle.

Für ibaLogic reservierte Namen
add_dt_time
add_time
add_tod_time
concat_d_tod
divtime
dt_to_date
dt_to_tod
multime
pi
pid
pidt1
pt1
pt2
ramp
sub_date_date
sub_dt_dt
sub_dt_time
sub_time
sub_tod_time
sub_tod_tod

8 Support und Kontakt

Bei Problemen wenden Sie sich bitte an folgende Nummern oder Adressen:

Telefon: +49 911 97282-14

Fax: +49 911 97282-33

Email: support@iba-ag.com

Über unsere Homepage können jeweils die neuesten Softwareversionen und Produktinformationen (auch diese Dokumentation) herunter geladen werden.

Die Web-Adresse lautet: www.iba-ag.com

Für Verbesserungsvorschläge und Hinweise auf Fehler innerhalb dieser Dokumentation sind wir dankbar. Einfach eine Email oder ein Fax an iba senden. Besten Dank für Ihre Unterstützung.

Zentrale

iba AG
Königswarterstraße 44
D-90762 Fürth / Bayern
Deutschland
Telefon: +49 (911) 97282-13
Fax: +49 (911) 97282-33
Kontakt: Harald Opel
iba@iba-ag.com

Benelux,
Frankreich und
Großbritannien,
Spanien

IBA-Benelux BVBA
Rivierstraat 64
B-9080 Lochristi
Belgien
Telefon: +32 9 226 2304
Fax: +32 9 226 2902
Kontakt: Roeland Struye
roeland.struye@iba-benelux.com

Nordamerika,
US Territories,
Karibik, Bermuda

iba America, LLC
6845 Shiloh Road East,
Suite D-7
Alpharetta, GA 30005
USA
Telefon: +1 (770) 886-2318
Fax: +1 (770) 886-9258
Kontakt: Scott Bouchillon
sb@iba-america.com

Venezuela und
Südamerika

iba LAT, S.A.
C.C San Miguel 1, Piso 1, Oficina 1.
Calle Neveri, Redoma de Harbor
YV 8050 Puerto Ordaz
Venezuela
Kontakt: Eric Di Luzio
Tel.: + 58 (286) 951 9666
Fax.: + 58 (286) 951 2915
Cel.: + 58 (414) 386 0427
eric.di.luzio@iba-ag.com

ibaChina,
ibaKorea,
ibaIndia,
ibaIndonesien
ibaMalaysia,
ibaThailand

ibaASIA GmbH & Co. KG
Saturnstrasse 32
D-90522 Oberasbach
Germany
Telefon: +49 (911) 969 4346
Fax: +49 (911) 969 4351
Kontakt: Mario Gansen
iba@iba-asia.com

Glossar

Ablaufsteuerung
Steuerungsverfahren, das einzelne Schritte in einer vorherbestimmten Reihenfolge abarbeitet. Von den Schritten kann jeweils nur einer aktiv sein. Projektierung mit SFC (Sequential Function Chart).

Anweisungsliste (AWL)
Assemblerartige Programmiersprache für SPS, genormt (früher DIN 19239), jetzt IEC 61131-3.

Berechnungs-Modus
Während der grafischen Projektierung von ibaLogic kann sofort und ohne Wartezeiten in die Offline-Berechnung (Berechnungs-Modus) zu Test- und Diagnosezwecken umgeschaltet werden kann. Die Funktionalität des erstellten Programms kann auf diese Art und Weise einfach und schnell einem Einzeltest unterzogen werden. In diesem Modus werden keine Ausgänge an den Prozess ausgegeben.

csv
Comma Separated Value; allgemeine Bezeichnung für ASCII-Textdateien mit Wertereihen, wobei die einzelnen Werte durch ein Trennzeichen getrennt sind. Trennzeichen sind üblicherweise das Komma (,) das Semikolon (;) oder das Tabulatorzeichen (<TAB>). Tabellenverarbeitungsprogramme wie MS Excel können diese Dateien im- bzw. exportieren.

Funktion
Unterprogramme, die beliebig viele Eingangsparameter haben und genau ein Ergebnis zurückliefern (Beispiel: sin(x)). Funktionen liefern bei gleicher Eingangsbeschaltung stets das gleiche Ergebnis (besitzen kein Gedächtnis)

Funktionsbaustein
Funktionsbausteine haben beliebig viele, klar definierte Ein- und Ausgangsparameter und können interne Variablen verwenden, d.h. sie besitzen ein Gedächtnis (z.B. PID-Regler)

HOT SWAP
Eigenschaft von ibaLogic. Bei Aktivierung dieser Funktion erzeugt ibaLogic ein Kopie des aktiven Projektes. Dieser Plan kann im HOT SWAP Bereich evaluiert werden. Durch zyklusgerechte Umschaltung zwischen HOT SWAP und Online Layer können umfangreichere Änderungen im Projekt vorgenommen und dann aktiviert werden.

IEC 61131
5-teiliger Standard für speicherprogrammierbare Steuerungen, insbesondere Teil 3 (IEC 61131-3) behandelt die Programmiersprachen für SPS.

Konfiguration
Automatisierungskomponenten z.B. eine SPS mit Rahmen und Baugruppen oder ein ibaLogic-PC, die miteinander kommunizieren können.

Online-Modus
Im Online-Modus erfolgt die Aktivierung des Arbeitspaketes mit einlesen und ausgeben von Variablen an den Prozess. Die Hintergrundfarbe des Arbeitsbereichs wechselt von grau in violett und zeigt damit an, dass das Projekt im Online-Modus arbeitet.

Programm
Normbegriff; beinhaltet die Verschaltung von **Funktionen** und **Funktionsbausteinen**. Kann in jeder der von IEC 61131 definierten Programmiersprache geschrieben werden. Programme sind explizit einer **Task** mit einem bestimmten Zeitverhalten zugeordnet.

POUs
Normbegriff; Program Organization Unit, Programmorganisations-Einheit gemäß IEC 61131, also Programm, Funktionsbaustein oder Funktion.

Ressource (Projekt)
Normbegriff; eine Konfiguration besteht aus einer oder mehreren **Ressourcen.** Eine Ressource ist einer CPU zugeordnet. ibaLogic kennt eine Ressource pro CPU und nennt diese "**Projekt**".

Ressource (I/O)
Innerhalb ibaLogic werden I/O Kanäle (Signale) als Input (Eingangs-) bzw. Output (Ausgangs-) Ressourcen bezeichnet.

SFC
Sequential Function Chart; Ablaufsprache gemäß IEC 61131 für Ablaufsteuerungen.

Soft SPS / Soft PLC
PC-speicherprogrammierbare Steuerung (Programable Logic Controller); besteht aus einem PC sowie der zugehörigen Software, der einen Prozess steuert, regelt und überwacht.

SPS
Speicherprogrammierbare Steuerung (Programable Logic Controller). Gerät, das einen Prozess steuert, regelt und überwacht. Besteht aus einem Baugruppenträger, Software, CPU, I/O-Baugruppen.

Strukturierter Text (ST)
Programmiersprache nach IEC 61131-3, ähnelt der Hochsprache PASCAL.

Task
Einer **Ressource** (Projekt) können eine oder mehrere **Tasks** zugeordnet werden. Wesentlich an einer Task ist ihr Zeitverhalten. Dieses Zeitverhalten kann explizit beschrieben werden. In einer Task werden zeitlich zusammengehörige Aufgaben zusammengefasst.

A

Literatur- und Quellenangaben

[1] IEC 61131-3: a standard programming resource http://www.plcopen.org/intro_nw.htm
[2] Karl Pusch, Grundkurs IEC 61131, Vogel Verlag, 1. Auflage, 1999
[3] E.Grötsch, SPS1 Speicherprogrammierbare Steuerungen, Oldenbourg Verlag, 4. Auflage, 2000
[4] OPC-Foundation: OLE for Process Controls OPC Common Definitions and Interfaces V1.0
[5] OPC-Foundation: Data Access Automation Interface Standard V 2.02

Stichwortverzeichnis

C

C

S

T

U

V

W

X

Z

C