

TiCoBasic

**Echtzeit-Entwicklungstool für
TiCo-Prozessoren**

***TiCoBasic* Version 1.0**

Feb. 2017

License Key:

ADwin - die schnellsten Echtzeitsysteme unter Windows

Hier finden Sie immer einen Ansprechpartner für Ihre Fragen:

Hotline: (0 62 51) 9 63 20
Fax: (0 62 51) 5 68 19
E-Mail: info@ADwin.de
Internet: www.ADwin.de



Jäger Computergesteuerte
Messtechnik GmbH
Rheinstraße 2-4
D-64653 Lorsch

Inhaltsverzeichnis

Inhaltsverzeichnis	III
Konventionen	2
1 Einführung	3
2 TiCoBasic für ADbasic-Nutzer	7
3 Entwicklungsumgebung nutzen	11
3.1 Grundlegende Schritte	11
3.1.1 Entwicklungsumgebung erstmals starten	11
3.1.2 Lizenzen für TiCoBasic prüfen und ändern	12
3.1.3 Kommunikation initialisieren	14
3.1.4 Grundelemente der Entwicklungsumgebung	15
3.2 Quelltexte erstellen	19
3.2.1 Online-Hilfe aufrufen	19
3.2.2 Kontextmenü im Quelltextfenster	21
3.2.3 Editor-Leiste	23
3.3 Quelltext übersetzen	23
3.4 Quelltext formatieren	25
3.4.1 Syntax hervorheben	25
3.4.2 Automatisch formatieren	26
3.4.3 Textzeilen einrücken	26
3.4.4 Kommentare positionieren	26
3.4.5 Zeilen in Kommentar ändern	27
3.4.6 Eigene Befehle und Variablen dokumentieren	27
3.4.7 Faltbaren Textbereich definieren	29
3.4.8 Textbereiche ein- / ausfalten	31
3.5 Suchen, markieren und ersetzen	32
3.5.1 Text schnell suchen	32
3.5.2 Text suchen und ersetzen	33
Beispiele für das Suchen von Text	37
Beispiele für das Ersetzen von Text	38
3.5.3 Reguläre Ausdrücke	38

3.5.4 Kontrollstruktur markieren	41
3.5.5 Textmarken nutzen	42
3.5.6 Zu einer Programmzeile springen	42
3.5.7 Zur Deklaration eines Befehls oder Variablen springen	42
3.5.8 Zwischen Sprungzielen wechseln	43
3.6 Programme leichter schreiben	44
3.6.1 Befehle und Variablen automatisch vervollständigen	44
3.6.2 Textbausteine einfügen	46
3.6.3 Befehlsparameter anzeigen	46
3.6.4 Deklaration eines Befehls oder Variablen anzeigen	47
3.6.5 Deklarationen einer Datei anzeigen	48
3.6.6 Verwendete globale Variablen und Felder anzeigen	48
3.6.7 Aufrufkürzel zu Dateien / Ordnern einfügen	49
3.7 TiCo-Binärdatei zum TiCo-Prozessor übertragen	49
3.7.1 TiCo-Binärdatei übertragen	49
3.7.2 TiCo-Bootloader programmieren	50
3.8 Projekte verwalten	51
3.9 Menüs	53
3.9.1 Das Menü „File“	55
3.9.2 Das Menü „Edit“	56
3.9.3 Das Menü „View“	56
3.9.4 Das Menü „Build“	57
3.9.5 Make Lib FileDas Menü „Options“	58
Dialogfenster „Compiler Options“	58
Dialogfenster „Process Options“	60
Dialogfenster „Settings“	64
Dialogfenster „Project Settings“	69
3.9.6 Das Menü „Debug“	71
Option Debug mode	72
3.9.7 Das Menü „Tools“	73
3.9.8 Das Menü „Window“	74
3.9.9 Das Menü „Help“	74
3.10 Fenster	76
3.10.1 Toolbox	76
3.10.2 Projektfenster	76
3.10.3 Parameterfenster	78
3.10.4 Prozessfenster	80

3.10.5 Registerfenster	81
3.10.6 Statusleiste	82
3.11 Info-Bereich	83
3.11.1 Infofenster	84
3.11.2 ToDo-Liste	85
3.11.3 Fenster „Global Variables“	86
3.11.4 Fenster „Declarations“	87
3.12 ADtools	89
4 Prozesse programmieren	91
4.1 Programmaufbau	91
4.1.1 Die Programmabschnitte	93
4.1.2 Befehle für den Hardware-Zugriff	93
4.1.3 Benutzerdefinierte Befehle und Variablen	94
4.2 Variablen und Felder	96
4.2.1 Übersicht	96
4.2.2 Datenstrukturen	96
4.2.3 Datentypen	97
4.2.4 Zahlenwerte eingeben	98
4.2.5 Globale Variablen (Parameter)	98
4.2.6 Globale Felder (Arrays)	99
4.2.7 System-Variablen	101
4.2.8 Lokale Variablen und Felder	102
4.3 Variablen und Felder – Details	103
4.3.1 Variablen und Felder im Datenspeicher	103
4.3.2 Speicherbereiche	103
4.3.3 Die Datenstruktur RingBuffer	105
4.4 Berechnungsausdrücke	113
4.4.1 Auswertung von Operatoren	113
4.5 Bedingungen, Schleifen und Module	114
4.5.1 Unterprogramm- und Funktions-Makros	115
4.5.2 Include-Dateien	115
4.5.3 Bibliotheken (Libraries)	116
5 Prozesse optimieren	119
5.1 Bearbeitungszeit messen	119
5.2 Verschiedene Tipps	120

5.2.1 Zugriff auf Hardware-Adressen	120
5.2.2 Konstanten anstelle von Variablen	121
5.2.3 Schnellere Messfunktion	121
5.2.4 Wartezeit genau einstellen	122
5.2.5 Wartezeiten nutzen	122
5.2.6 Optimierung des Speicherzugriffs	124
5.3 Debugging	124
5.3.1 Laufzeitfehler erkennen (Debug-Modus)	124
6 Prozesse im Betriebssystem.	127
6.1 Prozessverwaltung	128
6.1.1 Zeitgesteuerter Prozess (Timer)	128
6.1.2 Extern gesteuerter Prozess (External)	129
6.1.3 Prozess ohne Ansteuerung (None)	130
6.2 Zeitverhalten von Prozessen	131
6.2.1 Processdelay	131
6.2.2 Auslastung des TiCo-Prozessors	132
6.2.3 Verschiedene Betriebszustände im Betriebssystem	133
6.3 Kommunikation.	134
6.3.1 Datenaustausch zwischen Prozessen	134
6.3.2 Kommunikation zwischen PC und TiCo-Prozessor	135
6.3.3 Kommunikation zwischen ADwin CPU und TiCo-Prozessor	136
6.3.4 Die Device No	137
7 Befehlsreferenz.	139
7.1 Befehlssyntax	139
7.2 Basis-Befehlssatz <i>TiCoBasic</i>	140
7.3 Mathematik-Befehle	227
7.4 Gold II: TiCo-Prozessor	229
7.5 Pro II: <i>TiCo</i> -Prozessor	275
8 Was tun bei Problemen?	325
Anhang	A-1
A.1 Tastaturkürzel in <i>TiCoBasic</i>	A-1
A.2 ASCII-Zeichensatz	A-4

A.3 Lizenzvertrag	A-5
A.4 Kommandozeilen-Aufruf	A-9
A.5 Befehle für <i>ADwin-Gold II</i>	A-18
A.6 Befehle für <i>ADwin-Pro</i>	A-20
A.7 Index	A-23
A.8 Befehle in diesem Handbuch.	A-39

Liebe Leserin, lieber Leser!

TiCoBasic ist das Werkzeug, mit dem Sie *TiCo*-Prozessoren im *ADwin*-System für Ihre spezielle Mess-, Regel- oder Steuer-Aufgabe programmieren. Dieses Handbuch führt Sie in die Grundlagen der Programmierung von Echtzeit-Prozessen ein und soll Ihnen als Nachschlagewerk dienen. Nutzen Sie parallel auch die Online-Hilfe mit F1.

Die Entwicklungsumgebung wie auch die Sprache *TiCoBasic* sind eng verwandt mit *ADbasic*; ein Umstieg fällt daher leicht. Beachten Sie bitte [Kapitel 2](#), in dem die Unterschiede zwischen *TiCoBasic* und *ADbasic* dargestellt sind.

Wenn Sie den *TiCo*-Prozessor und *TiCoBasic* erstmals verwenden, empfehlen wir den schnellen Einstieg mit [Kapitel 2](#) und [4](#). Wir setzen voraus, dass Sie bereits über grundlegende Kenntnisse des Programmierens z.B. in Basic verfügen.

[Kapitel 3](#) beschreibt die Entwicklungsumgebung und wird für alle Anwender empfohlen.

Sollten Sie Hinweise haben, wo wir unsere Dokumentation verbessern können, bitten wir um Ihre Rückmeldung. Sie helfen uns damit, Ihnen ein gut verständliches und leicht bedienbares Werkzeug an die Hand zu geben.

Wir wünschen Ihnen viel Erfolg beim Programmieren.

Für Rückfragen wenden Sie sich bitte an unseren Support (Adresse in der vorderen Innenseite des Handbuchs).

Konventionen

Wir verwenden in diesem Handbuch die folgenden typographischen Konventionen und Zeichen:



Dieses Zeichen (Achtung) steht neben einem Absatz mit wichtigen Informationen für eine korrekte Funktion und fehlerfreien Betrieb.



Bei einem „Hinweis“ finden Sie Tipps und Ratschläge für einen effizienten Betrieb.



Das Informations-Zeichen verweist auf weiterführende Informationen im Handbuch oder in anderen Quellen (Dokumentation, Datenblätter, Literatur etc.).



Ein Beispiel verdeutlicht Ihnen, wie Sie das Gelesene einfach in die Praxis umsetzen können.

Am Schrifttyp „Courier“ erkennen Sie Texte, die auf dem Bildschirm erscheinen, z.B. in Fenstern oder Menüs, oder die Sie mit der Tastatur eingeben. In ähnlicher Weise sind die Namen von Menüs und Untermenüs dargestellt: „Menü ► Untermenü“.

Dateinamen und Pfadnamen sind zusätzlich in spitze Klammern gesetzt: `<path\xx.ext>`.

Elemente eines Quelltextes wie **Befehle**, Variablen, Kommentar und sonstiger Text werden ähnlich dargestellt wie in der Entwicklungsumgebung.

Tastenbezeichnungen werden in eckigen Klammern und in Kapitälchen angegeben wie [RETURN] oder [CTRL].

Die Bits eines Datenwortes (hier: 16 Bit) werden wie folgt nummeriert:

Bitnr.	15	14	13	...	01	00
Wert des Bits	2^{15}	2^{14}	2^{13}	...	$2^1=2$	$2^0=1$
Bezeichnung	MSB	-	-	-	-	LSB

Wenn Zahlen nicht dezimal angegeben werden, erhalten sie als Anhang einen kennzeichnenden Buchstaben; z.B. für die Zahl 17:

- hexadezimale Schreibweise: **11h**
- binäre Schreibweise: **10001b**

1 Einführung

Das *ADwin*-Echtzeitsystem übernimmt alle zeitkritischen Aufgaben in Ihren schnellen dynamischen Prüfständen und Fertigungsanlagen. In diesem Rahmen ist der – im Echtzeitsystem integrierte – *TiCo*-Prozessor dort vorgesehen, wo besonders exaktes und fein einstellbares Timing gefordert ist.

Während Sie das *ADwin*-Echtzeitsystem in *ADbasic* programmieren, verwenden Sie für den *TiCo*-Prozessor die Sprache und Bedienoberfläche *TiCo-Basic*.

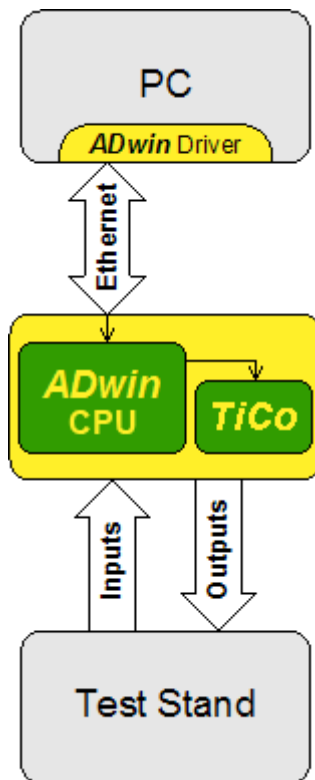
Damit Sie schnell und effizient mit der Programmierung beginnen können, möchten wir Ihnen zunächst das Konzept des *ADwin*-Systems mit *TiCo*-Prozessor erklären:

Jedes *ADwin*-System besitzt als zentrale Einheit die *ADwin* CPU, die zeitkritische Aufgaben in Echtzeit ausführt. Daneben kann ein *ADwin*-System (*Pro II* und *Gold II*) einen oder auch mehrere *TiCo*-Prozessoren besitzen. Analoge und digitale Ein- und Ausgänge sowie Erweiterungen wie Zähler und Bussysteme bilden die Verbindung zu Ihrem Prüfstand.

Im *ADwin*-System arbeitet der *TiCo*-Prozessor (*Timing Controller*) als eigenständiger, frei programmierbarer Zusatzprozessor, der auf Ein- und Ausgänge zugreift und dadurch Sonderfunktionen wie Filterung, Umrechnung, Kommunikationsprotokoll (SPI), Signalgenerator, Regelung etc. ausführen kann. Je nach Zielsetzung kann der *TiCo*-Prozessor der *ADwin* CPU zuarbeiten, indem er z.B. Daten vorverarbeitet; oder er übernimmt eine vollkommen eigenständige Aufgabe.

Der *TiCo*-Prozessor ist für schnelle Reaktionszeiten und exaktes Timing im Nanosekundenraster optimiert.

Die Kommunikation zwischen PC und *ADwin* CPU findet direkt über Ethernet statt. Der *TiCo*-Prozessor dagegen ist als Softcore im FPGA implementiert, zu dem der PC keine direkte Verbindung hat. Der Datenaustausch zwischen PC



und *TiCo*-Prozessor läuft daher immer über die *ADwin* CPU als Zwischenstation.

Der *TiCo*-Prozessor wird mit dem Echtzeit-Entwicklungs-Tool *TiCoBasic* programmiert, das die einfache und schnelle Erstellung von zeitkritischen Echtzeit-Prozessen ermöglicht. *TiCoBasic* ist eine integrierte Entwicklungsumgebung unter Windows für die gleichnamige Programmiersprache. Die Befehle entsprechen weitgehend denen von *ADbasic* und erlauben Ihnen, auf Ein- und Ausgänge zuzugreifen, Echtzeitprozesse zu steuern und den Datenaustausch mit der *ADwin* CPU vorzubereiten. In [Kapitel 4](#) ist erklärt, wie Sie *TiCoBasic*-Programme aufbauen, in [Kapitel 6](#) außerdem der Ablauf Ihrer Prozesse im Betriebssystem.



Mit nur wenigen Programmzeilen können Sie beispielsweise:

- Messgrößen bis zu Abtastfrequenzen von 800kHz erfassen
- schnelle digitale Regler mit Abtastraten bis zu 400kHz entwickeln
- gleichzeitig analoge Signale erzeugen *und* messen, z. B. für dynamische Kennlinien-Messungen

Die Entwicklungsumgebung *TiCoBasic* unterstützt Sie bei der Umsetzung Ihrer Aufgabe in einen Prozess. Zunächst erstellen Sie den Quelltext in einer erweiterten Basic-Syntax; mit den Befehlen und Funktionen können Sie die Hardware Ihres *ADwin*-Systems komfortabel programmieren. In [Kapitel 4](#) ist erklärt, wie Sie Programme aufbauen.



Mit dem integrierten Compiler erzeugen Sie aus dem Quelltext lauffähigen Binärcode, der als Prozess auf das *ADwin*-System übertragen und getestet wird. *TiCoBasic* bietet Ihnen auch die Hilfsmittel, mit denen Sie Ihre Prozesse



beobachten, Fehler suchen und die Programme optimieren können (siehe [Kapitel 2](#)).

Sobald die Echtzeit-Prozesse zu Ihrer Zufriedenheit laufen, ist Ihre Arbeit mit *TiCoBasic* bereits beendet.

Sie können die Prozesse und Prozessdaten des *TiCo*-Prozessors von der *ADwin* CPU aus steuern und beobachten, d. h. Daten lesen und schreiben sowie Prozesse starten, steuern und stoppen.

Obwohl der *TiCo*-Prozessor autark arbeitet, können Sie aus von der *ADwin* CPU jederzeit auf globale Variablen und Felder zugreifen, ohne zeitkritische Prozesse zu verzögern. Über die globalen Variablen und Felder können alle Prozesse untereinander oder mit der *ADwin* CPU schnell Daten austauschen. Mehr Informationen finden Sie im [Kapitel 6.3.3 „Kommunikation zwischen ADwin CPU und TiCo-Prozessor“](#) auf [Seite 136](#).

Die klare Trennung von Echtzeit-Prozessen im *TiCo*-Prozessor und im *ADwin*-System einerseits und der Bedienoberfläche im PC andererseits garantiert Ihnen höchste Betriebssicherheit und zeitlich nachvollziehbare Abläufe.

2 TiCoBasic für ADbasic-Nutzer


Mit *TiCoBasic* arbeiten Sie wie mit *ADbasic 5*: Sie werden viele Funktionen und Arbeitsabläufe in gewohnter Weise nutzen können, Sie finden Menüeinträge und Schaltflächen an der gleichen Stelle. Das erleichtert Ihnen den Einstieg in *TiCoBasic*.


Es gibt aber auch einige wichtige Unterschiede, die Sie im Folgenden kennen lernen. So ist der *TiCo*-Prozessor keine zweite *ADwin* CPU im Kleinform, sondern er ist für schnelle Reaktionszeiten und exaktes Timing im Nanosekundenraster optimiert.


Kommunikation mit dem *TiCo*-Prozessor

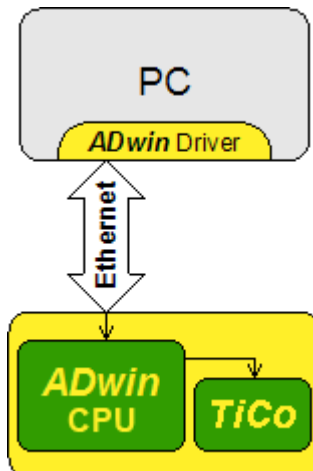
Der *TiCo*-Prozessor ist als Softcore im FPGA implementiert, zu dem der PC keine direkte Verbindung hat. Der Datenaustausch zwischen PC und *TiCo*-Prozessor läuft daher immer über die *ADwin* CPU als Zwischenstation.

Beachten Sie:

- Sie stellen unter *Options > Compiler* neben *ADwin*-System und Gerätenummer (Device No.) auch die *ADwin* CPU und den *TiCo*-Prozessor ein.
- Initialisieren statt Booten: Die Schaltfläche  initialisiert die gewählte *ADwin* CPU für den Datenaustausch mit dem *TiCo*-Prozessor.

Der *TiCo*-Prozessor selbst wird erst initialisiert, wenn ein Prozess kompiliert  wird. Dabei werden alle globalen Variablen auf 0 gesetzt und der kompilierte Prozess startet automatisch.

- Der *TiCo*-Prozessor läuft weiter, auch wenn die *ADwin* CPU nicht verfügbar ist, z. B. weil er gebootet wurde. Um den Datenaustausch wieder einzurichten, genügt es dann, die *ADwin* CPU mit der Schaltfläche  wieder zu initialisieren.



TiCo-Prozesse wie gewohnt

TiCo-Prozesse werden grundsätzlich programmiert wie ADbasic-Prozesse.

- Es gibt folgende Anzahl und Typen von Prozessen:

Prozesstyp	Priorität hoch	Priorität niedrig
zeitgesteuert	1	1
extern gesteuert	1	–
nicht gesteuert	1	–

Bitte beachten Sie: Es sollte nur *ein* Prozess mit hoher Priorität laufen.

- Es gibt die 3 Programmabschnitte: **Init:**, **Event:** und **Finish:**, alle Programmabschnitte haben gleiche Priorität. Es gibt keinen Programmabschnitt **LowInit:**.
- Ein Taktzyklus des Prozessors vom Typ TiCo1 dauert 20ns, beim Typ TiCo2 sind es 10ns.

Vereinfachter Befehlssatz

- Der TiCo-Prozessor verarbeitet ausschließlich den Datentyp **Long**; der Datentyp **Float** steht nicht zur Verfügung.

Die folgenden Befehle gehören zum Basisbefehlssatz. Darüber hinaus gibt es Befehle zum Zugriff auf die Ein-/Ausgänge und Schnittstellen der ADwin-Hardware, die in einem separaten Dokument beschrieben sind.

Standard	Init: , Event: , Finish: , <i>REM</i> , : (Doppelpunkt)
Variablen	Dim , Par_1...Par_80 , Data_1...Data_16 Ringbuffer_For_Read Ringbuffer_For_Write Lokale Felder
Mathematik, Operatoren	+ - * / Inc , Dec , Shift_Left , Shift_Right AbsI , Not , Or , XOr , And
Vergleichen	= < > Or , And
Struktur	For...Next , Do...Until If...Then , SelectCase Function , Sub , Lib_Function , Lib_Sub

Prozesse, Systemvariablen	<code>End</code> <code>Processdelay</code> , <code>Process_Running</code> , <code>NwTime</code>
Pre-Compiler	<code>#If...#Endif</code> , <code>#Define</code> , <code>#Include</code>
Zeitsteuerung	<code>NOP</code> , <code>NOPS</code> , <code>Sleep</code> , <code>Read_Timer</code>
I/O-Zugriff	<code>In</code> , <code>Out</code>

Fließkomma-Berechnungen wie Integer-Potenzen, trigonometrische Funktionen und Division mit Rest sind nicht möglich.

- Es können maximal 16 globale Felder `Data_1...Data_16` deklariert werden (in *ADbasic* bis `Data_200`). Der Datentyp `FIFO` ist nicht vorhanden.
- Es gibt die neuen Datentypen `Ringbuffer_For_Read` und `Ringbuffer_For_Write` (Ringspeicher).

Ringspeicher werden für schnelle Datenübertragung eingesetzt; die möglichen Anwendungen schließen sich gegenseitig aus:

- Der TiCo-Prozess greift auf Daten im externen DRAM zu und zwar lesend wie schreibend über den gleichen Ringspeicher.
- ADbasic-Prozesse (auf der ADwin CPU) und TiCoBasic-Prozesse tauschen über einen Ringspeicher miteinander Daten aus. Für jede Übertragungsrichtung ist ein separater Ringspeicher erforderlich.
- Mehrere Prozesse auf dem TiCo-Prozessor tauschen über einen Ringspeicher miteinander Daten aus. Es sind 2 Ringspeicher erforderlich, einer zum Schreiben und einer zum Lesen.



Der Umgang mit dem Datentyp `Ringbuffer` ist nicht einfach. Bei falscher Anwendung können schwer auffindbare Fehler entstehen. Die Verwendung des Datentyps `Ringbuffer` ist daher erfahrenen Benutzern von *ADbasic* und *TiCoBasic* vorbehalten.

- Der Befehl `Exit` wird durch `End` ersetzt.
- Die Befehle `In` und `Out` ersetzen `Peek` und `Poke`.
- Auf *ADwin*-Hardware greifen Sie mit ähnlichen (manchmal auch den gleichen) Befehlen zu wie in *ADbasic*. Beachten Sie: Es darf entweder die *ADwin* CPU (per *ADbasic*-Befehl) oder der *TiCo*-Prozessor (per *TiCoBasic*-Befehl) auf die gleiche *ADwin*-Hardware (oder Schnittstelle) zugreifen, aber nicht beide gleichzeitig.

ADbasic-Befehle zur TiCo-Steuerung

Die ADwin CPU kann direkt auf Daten des TiCo-Prozessors zugreifen. Hierzu stehen Ihnen eine Reihe von ADbasic-Befehlen zur Verfügung, mit denen Sie Daten austauschen und Prozesse steuern können, siehe [Kapitel 7.4](#) und [Kapitel 7.5](#).

Umstellungen in der Oberfläche

- Die Oberfläche TiCoBasic unterstützt einfache Debug-Möglichkeiten, jedoch nicht den Timing-Modus, der in ADbasic zur Verfügung steht.
- Wenn mehrere Dateien zu einem Projekt gehören, werden sie in jedem Fall gemeinsam kompiliert. Eine einzelne Datei zu kompilieren ist daher nur möglich, wenn sie nicht zu einem Projekt gehört.
- In Zukunft wird es möglich sein, in TiCoBasic Assembler-Befehle einzufügen. Hierzu wurde das [Registerfenster](#) in die Oberfläche eingefügt, das die Registerwerte des TiCo-Prozessors anzeigt.

3 Entwicklungsumgebung nutzen

Die Entwicklungsumgebung *TiCoBasic* dient zum Entwickeln von Programmen für *TiCo*-Prozessoren. Der *TiCoBasic*-Compiler verarbeitet eine erweiterte Basic-Syntax wie *ADbasic*, jedoch mit etwas geringerem Sprachumfang.

Das fertig entwickelte *TiCoBasic*-Programm wird in eine Binärdatei kompiliert und – indirekt über die *ADwin* CPU – auf den *TiCo*-Prozessor übertragen. Im Bootloader-Modus wird das Programm nun automatisch und unabhängig von der Entwicklungsumgebung ausgeführt.

3.1 Grundlegende Schritte

3.1.1 Entwicklungsumgebung erstmals starten

Zum Starten gehen Sie vor wie folgt:

1. Sie starten die Entwicklungsumgebung, indem Sie im Windows-Startmenü **Programs ▶ ADwin ▶ TiCoBasic** wählen.

Beim ersten Starten kann es einige Sekunden dauern, bis die Oberfläche erscheint, weil dabei auch das Windows-Programmpaket *.Net-Framework* gestartet wird.

Sie sehen nun die Entwicklungsumgebung mit den Windows-typischen Elementen wie Fenster, Menü- und Werkzeug-Leiste.

2. Beim ersten Start erscheint das Fenster **License key**. Geben Sie hier Ihren Lizenzschlüssel ein. Sie finden den **License key** auf dem Deckblatt dieses *TiCoBasic*-Handbuchs.

Sie müssen zum Registrieren des Lizenzschlüssels unter Windows NT, 2000, XP, Vista, 7 und 8 zur Benutzergruppe „Administratoren“ gehören. Es genügt nicht, volle Zugriffsrechte auf dem PC zu haben. Fragen Sie hierzu Ihren System-Administrator.

Ohne Eingabe des gültigen **License key** befindet sich *TiCoBasic* im Demo-Modus. In diesem Modus ist das Arbeiten mit der Entwicklungsumgebung nur zu Prüf-, Demonstrations- und Bewertungszwecken erlaubt. Sie können beispielsweise keine Binärdateien erzeugen.

Weitere Informationen zur *TiCoBasic*-Lizenz finden Sie im [Kapitel 3.1.2 auf Seite 12](#).

3. Stellen Sie im Menü *Options\Compiler* ein, wo sich der *TiCo*-Prozessor befindet, den Sie als Nächstes programmieren:
 - den Typ des *ADwin*-Systems und der *ADwin* CPU,
 - die Gerätenummer (Device No.)
 - bei einem Pro II-System auch das Modul mit dem *TiCo*-Prozessor.

Die Entwicklungsumgebung speichert Ihre Angaben, so dass Sie sie bei einem erneuten Start von *TiCoBasic* nicht mehr eingeben müssen – es sei denn, Sie verwenden einen anderen *TiCo*-Prozessor.

3.1.2 Lizenzen für *TiCoBasic* prüfen und ändern

Um den Lizenzschlüssel für *TiCoBasic* zu prüfen oder zu ändern, gehen Sie vor wie folgt:

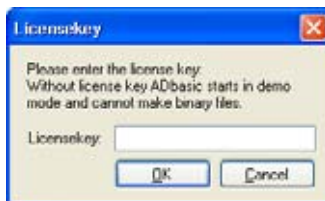
1. Wählen Sie den Menüeintrag *Help ► About*.

Das Fenster *About TiCoBasic* öffnet sich. Dort sind neben der Version der Entwicklungsumgebung unter *Licenses* die aktuellen Lizenzen angegeben (Liste möglicher Lizenzen siehe unten).



2. Zum Eingeben oder Ändern der Lizenzen klicken Sie die Schaltfläche *Change License*.

Das Eingabefenster *License key* öffnet sich.



3. Geben Sie den Lizenzschlüssel ein.

Sie finden den `License key` auf dem Deckblatt dieses *TiCoBasic*-Handbuchs.

Für *TiCoBasic* sind folgende Lizenzen möglich:

- Keine Lizenz (Demo mode)

Ohne Eingabe des gültigen `License key` befindet sich *TiCoBasic* im Demo-Modus. In diesem Modus ist das Arbeiten mit der Entwicklungsumgebung nur zu Prüf-, Demonstrations- und Bewertungszwecken erlaubt. Sie können beispielsweise keine Binärdateien erzeugen.

- Zeitlich begrenzte Lizenz (Evaluation license)

Die Lizenz schaltet alle Funktionen der Entwicklungsumgebung für einen definierten Zeitraum frei. Anschließend arbeitet *TiCoBasic* wieder im Demo-Modus (siehe oben).

- Zeitlich unbegrenzte Lizenz für den Lizenznehmer

Folgende Lizenzen können freigeschaltet werden:

- *ADbasic* 6, arbeitet mit allen *ADwin*-Prozessoren
- *ADbasic* 5, arbeitet mit *ADwin*-Prozessoren bis Version T11
- *ADbasic* 3.0, arbeitet mit *ADwin*-Prozessoren bis Version T9
- *ADbasic* 2.0, arbeitet mit *ADwin*-Prozessoren bis Version T8
- *TiCoBasic*
- *ADlab* (Matlab-Treiber für *ADwin*)

Die Lizenzen für *TiCoBasic* und *ADlab* können mit einer der *ADbasic*-Lizenzen kombiniert sein.

Die Lizenzbedingungen für *TiCoBasic* sind im [Lizenzvertrag](#) (Anhang Seite A-5) beschrieben.

3.1.3 Kommunikation initialisieren

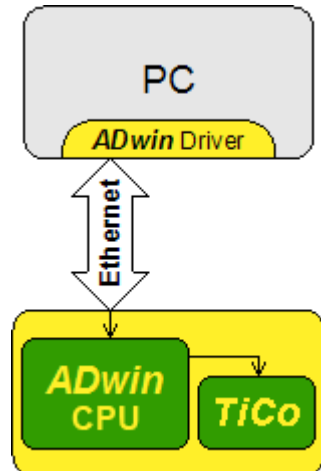
Der *TiCo*-Prozessor ist als Softcore im FPGA implementiert, zu dem der PC keine direkte Verbindung hat. Der Datenaustausch zwischen PC und *TiCo*-Prozessor läuft daher immer über die *ADwin* CPU als Zwischenstation.

Sie initialisieren die gewählte *ADwin* CPU für den Datenaustausch mit dem *TiCo*-Prozessor, indem Sie auf die Schaltfläche **I** klicken (= initialisieren). Dadurch wird ein Prozess in der *ADwin* CPU eingerichtet, der Daten zwischen PC und *TiCo*-Prozessor austauscht.


Der *TiCo*-Prozessor selbst wird erst initialisiert, wenn Sie einen Prozess kompilieren **C**. Dabei werden die Inhalte des Programm- und Datenspeichers überschrieben, alle globalen *TiCo*-Variablen auf 0 gesetzt und der kompilierte Prozess startet automatisch. Wenn kein *TiCo*-Prozessor vorhanden ist, erhalten Sie eine entsprechende Fehlermeldung.

Der *TiCo*-Prozessor läuft weiter, auch wenn die *ADwin* CPU nicht verfügbar ist, z. B. weil sie gebootet wurde. Um den Datenaustausch wieder einzurichten, genügt es dann, die *ADwin* CPU mit der Schaltfläche **I** (Initialize) wieder zu initialisieren.

Sie können den *TiCo*-Prozessor auch mit der Schaltfläche **R** (Reset) stoppen und zurücksetzen. Dadurch gehen alle Werte in den globalen *TiCo*-Variablen verloren und der Prozess wird gelöscht.



3.1.4 Grundelemente der Entwicklungsumgebung

Die Entwicklungsumgebung besteht aus mehreren Leisten und Fenstern (siehe [Abb. 1](#)); Sie können die Höhe der Fenster frei anpassen. Sie erhalten Hilfe zu einem Fenster oder zu dem aktuell markierten Kennwort, wenn Sie die Taste [F1] drücken (siehe auch [Online-Hilfe aufrufen](#)). Mit der Schaltfläche  öffnen Sie das Indexverzeichnis der Hilfe.

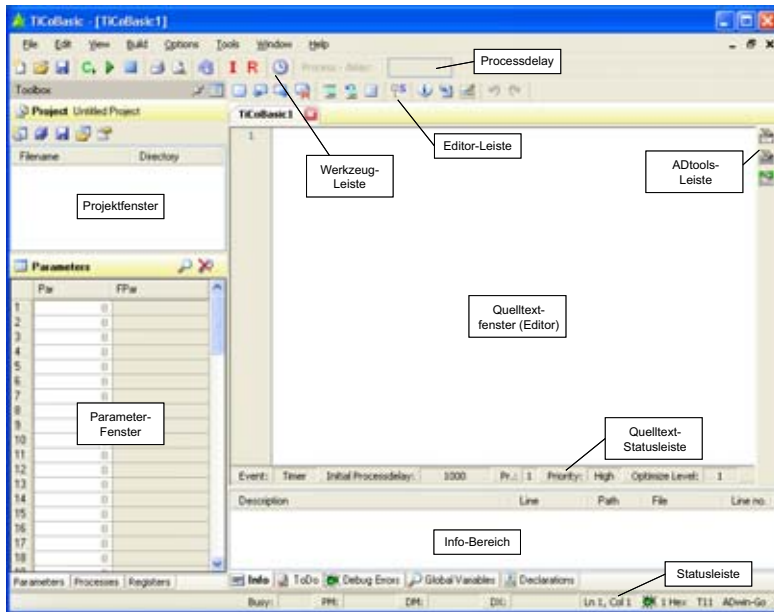


Abb. 1 – Elemente der *TiCoBasic*-Entwicklungsumgebung

Sie finden die Befehle der Entwicklungsumgebung über:

- die Werkzeugleiste und die Editorleiste (siehe [Abb. 2](#)).
- die Projektleiste im [Projektfenster](#)
- die Kontextmenüs der Fenster (rechte Maustaste).
- die Menüleiste.
- die [Tastaturkürzel in TiCoBasic](#) (siehe Anhang).
- die [ADtools](#)-Leiste.
Hier können Sie auch eigene Aufrufkürzel einfügen (siehe [Kapitel 3.6.7 auf Seite 49](#)).

Wenn Sie einen Befehl anwenden, sehen Sie am linken Rand der Statusleiste eine Befehlserklärung.



Abb. 2 – Die Werkzeugleiste


Sie wählen ein Menü (und dann einen Befehl) aus, indem Sie das Menüfeld mit der linken Maustaste anklicken, oder indem Sie die Tastenkombination [ALT] + [ANFANGSBUCHSTABE] des entsprechenden Menüs eingeben. Viele Befehle besitzen auch Tastatur-Kürzel (Short-Cuts, siehe Anhang [A.1](#)), die in den Menüs angezeigt werden.

Im aktiven Quelltext-Fenster (Editor) bearbeiten Sie den Quelltext für je einen Prozess. Sie können mehrere Fenster parallel geöffnet haben; die Fenstergrößen sind frei einstellbar. Weitere Informationen

zum jeweiligen Quelltextfenster werden an verschiedenen Stellen angezeigt.

- Die Titelleiste zeigt den Namen des aktiven Quelltext-Fensters an.
- Die Quelltext-Statusleiste zeigt die von Ihnen eingestellten Prozess-Optionen.

Ein Doppelklick auf die Quelltext-Statusleiste öffnet das [Dialogfenster „Process Options“](#).

- Im [Parameterfenster](#) (siehe [Kapitel 3.10.3 auf Seite 78](#)) können Sie durch Drücken der Schaltfläche `Scan Global Variables`  (einmalig) markieren lassen, welche globalen Parameter Sie im aktiven Quelltext oder Projekt verwenden; siehe [Verwendete globale Variablen und Felder anzeigen](#) auf [Seite 48](#).
- Im Infobereich unten werden in verschiedenen Fenstern eine Reihe von Informationen angezeigt:
 - [Infofenster](#): Fehlermeldungen (rot hinterlegt) und Warnungen des Compilers (siehe [Kapitel 3.11.1 auf Seite 84](#)).
 - [ToDo-Liste](#): Eine einfache To-Do-Liste aus Kommentarzeilen (siehe [Kapitel 3.11.2 auf Seite 85](#)).
 - Suchergebnisse bei einer Suche über alle Dateien eines Projekts (siehe [Kapitel 3.5.2 auf Seite 33](#)).
 - Debug-Informationen, wenn der Debug-Modus aktiv ist (siehe [Option Debug mode, Seite 72](#)).
 - [Fenster „Global Variables“](#): Eine Liste der verwendeten globalen Variablen und Felder (siehe [Kapitel 3.11.3 auf Seite 86](#)).
 - [Fenster „Declarations“](#): Eine Liste aller zu einer Quelltextdatei gehörenden Deklarationen (siehe [Kapitel 3.11.4 auf Seite 87](#)).

Beachten Sie, dass das Editieren im Quelltextfenster durch eine Reihe von Hilfsfunktionen unterstützt wird (siehe [Seite 19](#)).

Im [Projektfenster](#) werden der Name des offenen Projekts und die zugehörigen Dateien angezeigt, anderenfalls bleibt es leer.

Einige Daten des *TiCo*-Prozessors werden kontinuierlich ausgelesen und angezeigt (nur, wenn der PC mit dem *TiCo*-Prozessor kommuniziert):

- Das Processdelay (Prozess-Zykluszeit) zu der Prozessnummer, die Sie für das aktive Quelltextfenster eingestellt haben, dargestellt rechts von der Werkzeugleiste.
- Die Werte der globalen Variablen im [Parameterfenster](#); wenn Sie einen dieser Werte verändern, wird er sofort zum *ADwin*-System übertragen.
- Der Zustand der laufenden Prozesse im [Prozessfenster](#) ([Seite 80](#)).
- Der Registerwerte des *TiCo*-Prozessor im [Registerfenster](#) ([Seite 81](#)).
- Informationen zur Speicherauslastung in der [Statusleiste](#) (siehe [Kapitel 3.10.6](#)).

3.2 Quelltexte erstellen

Öffnen Sie für jeden Prozess-Quelltext ein eigenes Fenster (mit **File ▶ New** oder **[CTRL]+[N]**).

Sie können die Reihenfolge der Reiter der Quelltextfenster verändern. Dazu drücken Sie die **[ALT]**-Taste, klicken mit der Maus auf einen Reiter und ziehen ihn an die neue Position.

Wenn Sie für eine Aufgabe mehrere Dateien verwenden, empfehlen wir, diese gemeinsam in einer Projektdatei zu verwalten (siehe [Seite 51: Projekte verwalten](#)).

Um einen Quelltext auf der *ADwin*-Hardware zu testen, müssen Sie den [Quelltext übersetzen](#), siehe [Seite 23](#).

Der Editor und der *TiCoBasic*-Compiler unterscheiden nicht zwischen Groß- und Kleinschreibung. In unseren Beispielen verwenden wir allerdings zur besseren Unterscheidung eine einheitliche Schreibweise.

Bei Unklarheiten oder Problemen sollten Sie die [Online-Hilfe aufrufen](#) (siehe unten).

Beim Editieren im Quelltextfenster können Sie eine Reihe von Hilfsfunktionen nutzen. Sie rufen die Funktionen über das [Kontextmenü im Quelltextfenster](#) ([Seite 21](#)) oder über die [Editor-Leiste](#) ([Seite 23](#)) auf:

Sie können Zahlenwerte im Quelltext in dezimaler, hexadezimaler, binärer und Exponential-Schreibweise eingeben (siehe [Kapitel 4.2.4 „Zahlenwerte eingeben“](#)).

Weitere Editor-Funktionen finden Sie unter den Rubriken:

- [Quelltext formatieren](#), [Seite 25](#)
- [Suchen, markieren und ersetzen](#), [Seite 32](#)
- [Programme leichter schreiben](#), [Seite 44](#)

3.2.1 Online-Hilfe aufrufen

Das Menü **„Help“** ([Seite 74](#)) erlaubt den Aufruf von ausgesuchten Hilfe-Seiten, z.B. das Inhaltsverzeichnis oder sortierte Befehlslisten.


Mit **[F1]** öffnen Sie eine Hilfeseite zu dem gerade geöffneten Dialogfenster oder zu dem Befehl an der Cursor-Position.

Wenn an der Cursor-Position ein ungültiger Befehl steht, öffnet die Hilfe das Indexverzeichnis. Gründe hierfür sind in der Regel:

- Es handelt sich nicht um einen Befehl, sondern um eine vom Benutzer definierte Deklaration: Variable / Feld, symbolischer Name, Makro (Sub, Function). Hierzu kann es keine Hilfeseiten geben.
- Der Befehl ist falsch geschrieben, z.B. `Digin_Wrod` anstatt `Digin Word`.
Wenn Sie den Fehler korrigiert haben, wird der Befehl wieder farbig hervorgehoben.
- Stellen Sie die passende ADwin-Hardware ein (siehe [Dialogfenster „Compiler Options“](#)) oder verwenden Sie einen für eingestellte Hardware gültigen Befehl.
- Im Quelltext fehlt die (benutzerdefinierte) Include- oder Library-Datei, in der der Befehl definiert ist.
Fügen Sie die entsprechende Zeile am Anfang des Quelltexts ein.

Das Hilfefenster zeigt links mehrere Registerkarten, rechts die aufgerufene Hilfeseite. Die Registerkarten führen Sie zum Inhaltsverzeichnis (Contents), Index-Register (Index), Such-Fenster (Search) und zu sortierten Befehlslisten (Commands).

Sie können mehrere Hilfeseiten nebeneinander anzeigen:













- Klicken Sie mit der rechten Maustaste auf einen Link und wählen Sie im Kontextmenü den Eintrag `In neuem Fenster öffnen`; der Link öffnet sich dann in einem neuen Tab.
- Alternativ: Über die Schaltfläche  rechts oben (Add new tab) öffnen Sie beliebig viele (zunächst leere) Tab. Sobald Sie einen Link anklicken, wird das Link-Ziel im nächsten leeren Tab geöffnet. Wenn alle Tabs gefüllt sind, wird das Link-Ziel im aktiven Fenster angezeigt.

Sie können viele Vektorgrafiken in der Online-Hilfe (z.B. Pinbelegungen) zum Erkennen von Details vergrößern, indem Sie das Hilfefenster am Rahmen größer ziehen.

Bei der Suche innerhalb der Hilfe sind nur folgende Zeichen erlaubt: 0-9, A-Z, - (Minus) und Umlaute; Groß-/Kleinschreibung wird nicht unterschieden. Bei mehreren Wörtern dient das Leerzeichen als Trennzeichen, gefunden werden nur Seiten, die alle Wörter enthalten. Die Suche nach ganzen Wörtern ist optional.

3.2.2 Kontextmenü im Quelltextfenster

Im Quelltextfenster können Sie verschiedene Hilfsfunktionen über das Kontextmenü erreichen (Klick mit rechter Maustaste).

	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
<hr/>		
	Comment Block	Ctrl+B
	Uncomment Block	Ctrl+Shift+B
	Create Region from Block	
<hr/>		
	Indent	Ctrl+I
	Outdent	Ctrl+Shift+I
	Mark Controlblock	
	Unmark Controlblock	
<hr/>		
	Add to Project	
<hr/>		
	Declaration Info	F2
	Jump to Declaration	Ctrl+F2
	Codesnippets	Ctrl+K X

Die folgenden Befehle verwenden die Cursor-Position oder die aktive Markierung:

- Cut: Markierung löschen und in die Zwischenablage kopieren.
- Copy: Markierung in die Zwischenablage kopieren.
- Paste: Markierung löschen und durch den Inhalt der Zwischenablage ersetzen.
- Comment Block, Uncomment Block: [Zeilen in Kommentar ändern, Seite 27](#).
- Create Region from Block: [Faltbaren Textbereich definieren, Seite 29](#).
- Indent, Outdent: [Textzeilen einrücken, Seite 26](#).
- Mark Control block, Unmark Control block: [Kontrollstruktur markieren, Seite 41](#).
- Declaration Info: [Deklaration eines Befehls oder Variablen anzeigen, Seite 47](#).
- Jump to Declaration: [Zur Deklaration eines Befehls oder Variablen springen, Seite 42](#).

Ohne vorherige Markierung sind folgende Befehle verfügbar:

- Add to Project: Datei dem aktiven Projekt zufügen.
- Declaration Info: [Deklarationen einer Datei anzeigen, Seite 48](#).
- Code snippets: [Textbausteine einfügen, Seite 46](#).

3.2.3 Editor-Leiste

Im Quelltextfenster können Sie verschiedene Hilfsfunktionen über die Editor-Leiste erreichen.



Textmarken nutzen, Seite 42.



Zeilen in Kommentar ändern, Seite 27.



Faltbaren Textbereich definieren, Seite 29.



Textbereiche ein- / ausfalten, Seite 31.



Deklaration eines Befehls oder Variablen anzeigen, Seite 47.



Zur Deklaration eines Befehls oder Variablen springen, Seite 42.



Textbausteine einfügen, Seite 46.



Einen Befehl rückgängig machen oder wieder herstellen.




Zwischen Sprungzielen wechseln, Seite 43.

3.3 Quelltext übersetzen

Sie übersetzen (kompilieren) einen Quelltext mit *TiCoBasic* in eine Binärdatei. Die Binärdatei kann auf die *ADwin*-Hardware übertragen und dort als Prozess ausgeführt werden.

Dazu stehen Ihnen folgende Möglichkeiten zur Verfügung:

- Quelltext im Entwicklungsprozess testen:
Projekt-Quelltexte (oder einzelnen Quelltext) übersetzen und ausführen: Schaltfläche  in der Werkzeugleiste.
- Fertigen Quelltext als Binärdatei speichern:

Projekt-Quelltexte (oder einzelnen Quelltext) übersetzen und als Binärdatei speichern: Build ► Make Bin.

- Vor und nach dem Kompilieren Kommandozeilen-Aufrufe durchführen.
Kommandozeilen werden im Dialogfenster „Project Settings“ definiert, [Reiter Prebuild / Postbuild \(Seite 70\)](#).

Sie können erzeugte Binärdateien auf verschiedene Weise nutzen:

- Eine Binärdatei aus der Oberfläche *TiCoBasic* auf den *TiCo*-Prozessor übertragen und ausführen, siehe [TiCo-Binärdatei übertragen](#).
- Eine Binärdatei aus einem externen Programm – immer in Kombination mit einem *ADbasic*-Prozess – auf den *TiCo*-Prozessor übertragen, ausführen und steuern.

Es stehen Schnittstellen für gängige Entwicklungsumgebungen (z. B. Java, Visual Basic, C#.NET) und zahlreiche Bedienoberflächen (wie DIAdem, MATLAB) zur Verfügung.

- Eine Binärdatei dauerhaft im Bootloader des *TiCo*-Prozessors speichern und beim Einschalten automatisch starten: [Kapitel 3.7.2 „TiCo-Bootloader programmieren“](#), [Seite 50](#).

Ergänzend verweisen wir auf die Möglichkeit, Library-Funktionen aus einem Quelltext als Binärdatei zu erzeugen, siehe [Bibliotheken \(Libraries\)](#). Library-Funktionen können in anderen Quelltexten eingebunden und genutzt werden.

3.4 Quelltext formatieren

Quelltext kann (meist automatisch) formatiert werden, um die Übersicht über die Programmstruktur zu zeigen:

- [Syntax hervorheben, Seite 25](#)
- [Automatisch formatieren, Seite 26](#)
- [Textzeilen einrücken, Seite 26](#)
- [Kommentare positionieren, Seite 26](#)
- [Zeilen in Kommentar ändern, Seite 27](#)
- [Eigene Befehle und Variablen dokumentieren, Seite 27](#)
- [Faltbaren Textbereich definieren, Seite 29](#)
- [Textbereiche ein- / ausfalten, Seite 31](#)

Hier finden Sie weitere Editor-Funktionen:

- [Quelltexte erstellen, Seite 19](#)
- [Suchen, markieren und ersetzen, Seite 32](#)
- [Programme leichter schreiben, Seite 44](#)

3.4.1 Syntax hervorheben

Wenn Sie eine Befehlszeile eingegeben haben, ändert der Editor automatisch die Farbe der eingegebenen Befehlswörter, Variablen- und Feldnamen, und er rückt die Zeilen so ein, dass sich ein übersichtliches Bild ergibt.

Der Editor unterscheidet die von Ihnen eingegebenen Zeichenketten in mehrere Gruppen von Syntaxelementen, die unterschiedlich dargestellt werden. Sie können die Farbgebung unter `Options ▶ Settings, Editor - Syntax Colors` (siehe [Seite 65](#)) nach eigener Vorstellung verändern; dort ist auch eine Übersicht der Syntax-Gruppen angegeben.

Die vollständige Syntax-Hervorhebung erfordert, dass die Option `Parse Declarations` unter `Editor - General` (siehe [Seite 64](#)) aktiv ist.

3.4.2 Automatisch formatieren

Wenn Sie eine Befehlszeile eingegeben haben, korrigiert der Editor automatisch die Leerzeichen, so dass ein einheitliches Bild entsteht. So erhalten z.B. Operatoren wie „=“ oder Kennwörter wie „`if`“ rechts und links ein Leerzeichen.

Wenn Sie lieber manuell formatieren, müssen Sie zunächst unter [Editor - General](#), `Smart format` das automatische Formatieren abschalten (siehe [Seite 64](#)).

3.4.3 Textzeilen einrücken

Wenn Sie eine Befehlszeile eingegeben haben, rückt der Editor die Zeilen automatisch so ein, dass sich ein übersichtliches Bild ergibt. Wegen der Automatik ist manuelles Einrücken dann nicht möglich.

Wenn Sie lieber manuell formatieren, müssen Sie zunächst unter [Editor - General](#), `AutoIndent` die automatische Einrückung abschalten. Anschließend können Sie Einrückungen mit Tabulator [TAB] oder Leerzeichen [SPACE] erzeugen. Mehrere markierte Zeilen können gemeinsam ein- oder ausgerückt werden, indem im Kontextmenü des Quelltextfensters (Klick mit rechter Maustaste) die Einträge `Indent` oder `Outdent` gewählt werden.

Unter dem Menüpunkt `Options ▶ Settings`, [Editor - General](#), `Tab-size` wird eingestellt, wie viele Leerzeichen eine Einrückung hat.

3.4.4 Kommentare positionieren

Wenn Sie einen Kommentar mit `'` am Zeilenende eingeben, setzt der Editor den Kommentaranfang automatisch auf eine definierte Position. Das soll die Lesbarkeit der Kommentare verbessern.

Mit doppelten Kommentarzeichen `"` können Sie einen Kommentar auch bei automatischer Positionierung manuell positionieren.

Sie können die automatische Positionierung unter [Editor - General](#), `Align comments` an- oder abschalten und die Kommentarpозиtion einstellen (siehe [Seite 64](#)).

3.4.5 Zeilen in Kommentar ändern

Mehrere markierte Zeilen können gemeinsam in Kommentarzeilen umgewandelt werden, indem im Kontextmenü des Quelltextfensters (Klick mit rechter Maustaste) der Eintrag `Comment Block` gewählt wird. Als Folge fügt der Editor an jedem Zeilenanfang ein Kommentarzeichen ein, so dass der Compiler diese Zeilen überspringt.

In gleicher Weise löscht `Uncomment Block` ein Kommentarzeichen am Zeilenanfang wieder.

3.4.6 Eigene Befehle und Variablen dokumentieren

Sie können selbst geschriebene Befehle (`Sub`, `Function`), Variablen, Felder und symbolische Namen (`#Define`) so mit Hinweisen dokumentieren, dass die Hinweistexte beim Programmieren als Tooltips und bei der automatischen Vervollständigung angezeigt werden. Mehr darüber ist bei den Funktionen „[Befehle und Variablen automatisch vervollständigen](#)“ (Seite 44), „[Befehlsparameter anzeigen](#)“ (Seite 46) und „[Deklaration eines Befehls oder Variablen anzeigen](#)“ (Seite 47) beschrieben.

Beispiel

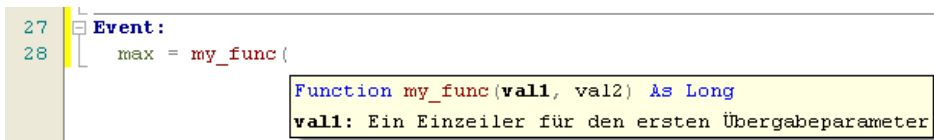
Die folgenden Beispiele zeigen, wie Sie die Hinweistexte aufbauen.

```
''' Hier beginnt der Hinweistext zur Funktion my_func;
''' + alle Folgezeilen, die mit 3 Kommentar- und einem Plus-
''' + Zeichen beginnen, gehören zum ersten Textabschnitt,
''' + der den Zweck der Funktion und des Rückgabewerts
''' + beschreibt.
''' vall: Ein Einzeiler für den ersten Übergabeparameter
''' Der zweite Übergabeparameter val2. Wie man sieht, kann
''' + die Zeile auch ohne Parameternamen beginnen, mit ist
''' + es aber übersichtlicher.
''' Diese Zeilen starten einen Bereich, der keinem
''' + Parameter zugeordnet werden kann und daher nicht
''' + verwendet wird.
Function my_func(val1, val2) As Long
    my_func = (val1 + val2) / 2
EndFunction

''' Hier folgt die Beschreibung einer Variablen.
''' Für mehrzeilige Bereiche ist das Plus-Zeichen nicht
''' + erforderlich, es schadet aber auch nicht.
Dim var As Long
```

```
''' Beschreibung für einen symbolischen Namen
#Define max Par_1
```

So erscheint der Tooltip beim Verwenden der Funktion:



Beachten Sie folgende Regeln zum Anlegen eigener Hinweise:

- Sie können folgende selbst definierten Elemente mit Hinweistexten dokumentieren:
 - symbolische Namen (**#Define**)
 - Variablen und Felder (**Dim**)
 - Makros (**Sub**, **Function**)
 - Bibliotheken (**Lib_Sub**, **Lib_Function**)
- Fügen Sie alle Hinweistexte in Kommentarzeilen ein, die mit 3 Kommentarzeichen ' ' ' beginnen. Eine Zeile mit weniger als 3 Kommentarzeichen beendet den Hinweisbereich.
- Hinweiszeilen müssen direkt über der Zeile stehen, die eines der oben genannten Kennwörter enthält (Definitionszeile). Bei richtiger Eingabe wird der Hinweistext als faltbarer Textbereich angezeigt (siehe [Seite 29](#)).

Wenn zwischen Definition und Hinweiszeilen eine andere Zeile steht, ist der Hinweisbereich vollständig deaktiviert.

- Bei Makros und Bibliotheken können Sie für den Befehl und jeden Übergabeparameter jeweils einen Unterbereich anlegen.
- Ein Unterbereich kann ein- oder mehrzeilig sein:
 - Die erste Zeile eines Unterbereichs beginnt mit 3 Kommentarzeichen ' ' ', jede weitere Zeile des Unterbereichs erhält zusätzlich ein Plus, beginnt also mit ' ' '+.
 - Die Anzahl der Zeilen in einem Unterbereich ist unbegrenzt.
 - In einem Unterbereich eines Übergabeparameters beginnt der Text mit dem Parameternamen, direkt gefolgt von einem Doppelpunkt. Alternativ kann der Name auch entfallen.

Die Anzeige der Hinweise (z.B. in Tooltips) ist nur möglich, wenn die Option `Parse Declarations` unter **Editor - General** (siehe [Seite 64](#)) aktiv ist.

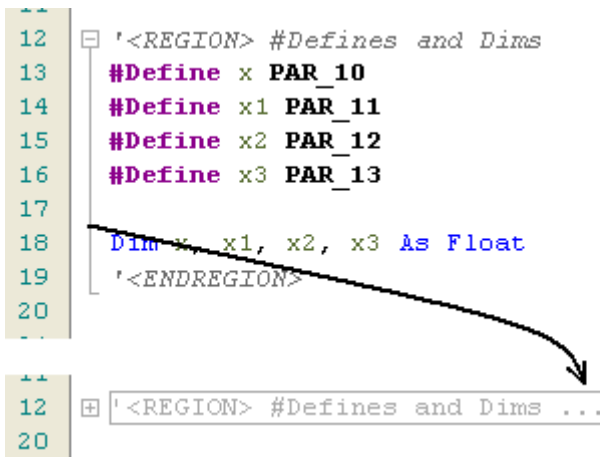
3.4.7 Faltbaren Textbereich definieren


Sie können beliebige Quelltextzeilen als faltbaren Bereich definieren. Ein faltbarer Bereich wird durch eine graue Linie am Zeilenanfang markiert, mit einem Minuszeichen in der ersten Zeile des Bereichs. Sie falten den Bereich mit einem Klick auf das Minuszeichen in der ersten Zeile.

Der Editor erkennt auch Strukturen wie Bedingungen und Schleifen, Programmabschnitte, Makros und Bibliotheks-Module als faltbare Textbereiche.

Sie definieren einen eigenen faltbaren Bereich mit den Schlüsselwörtern `<Region>` und `<EndRegion>` auf folgende Weise:

- Markieren Sie die gewünschten Quelltextzeilen.
Achten Sie darauf, dass die Markierung vorhandene Kontrollstrukturen immer vollständig oder gar nicht einschließt. Das Einschließen (oder Überlappen) eines anderen `<Region>`-Bereichs ist nicht erlaubt.
- Wählen Sie im Kontextmenü des Quelltextfensters den Menüeintrag `Create Region from Block`.
Ein Eingabefenster öffnet sich.
- Geben Sie eine kurze Beschreibung für den faltbaren Bereich unter `Region description` an und bestätigen Sie mit OK.
- Der markierte Bereich ist nun von den Schlüsselwörtern `<Region>` und `<EndRegion>` umgeben und kann ein- und ausgefoldet werden.
Sie können die Schlüsselwörter auch manuell eingeben.



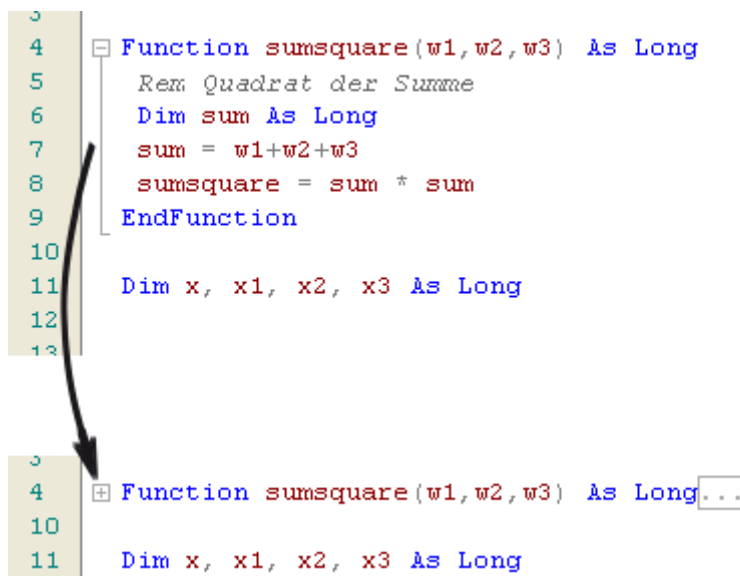
Mit der Schaltfläche `Toggle Folding`  können alle faltbaren Textbereiche auf einmal ein- oder ausgefoldet werden.


Faltbare Textbereiche werden nur erkannt, wenn die Option `Parse Declarations` unter **Editor - General** (siehe [Seite 64](#)) aktiv ist.

3.4.8 Textbereiche ein- / ausfalten

Der Editor erkennt Kontrollstrukturen wie Bedingungen und Schleifen, Programmabschnitte, Makros und Bibliotheks-Module als faltbare Textbereiche. Außerdem können Sie beliebige Quelltextzeilen als **Faltbaren Textbereich definieren** (siehe [Kapitel 3.4.7](#)). Ein solcher Bereich wird durch eine graue Linie am Zeilenanfang markiert, mit einem Minuszeichen in der ersten Zeile des Bereichs.

Sie falten einen einzelnen Bereich mit einem Klick auf das Minuszeichen in der ersten Zeile; im Beispiel unten würden Sie links neben `Function sumsquare` klicken.



Mit der Schaltfläche **Toggle Folding**  können alle faltbaren Textbereiche auf einmal ein- oder ausgefaltet werden.

Faltbare Textbereiche werden nur erkannt, wenn die Option `Parse Declarations` unter **Editor - General** (siehe [Seite 64](#)) aktiv ist.

3.5 Suchen, markieren und ersetzen

Suchen, markieren oder ersetzen Sie einen beliebigen Text mit diesen Funktionen:

- [Text schnell suchen, Seite 32](#)
- [Text suchen und ersetzen, Seite 33](#)
- [Reguläre Ausdrücke, Seite 38](#)
- [Kontrollstruktur markieren, Seite 41](#)
- [Textmarken nutzen, Seite 42](#)
- [Zu einer Programmzeile springen, Seite 42](#)
- [Zur Deklaration eines Befehls oder Variablen springen, Seite 42](#)
- [Zwischen Sprungzielen wechseln, Seite 43](#)

Hier finden Sie weitere Editor-Funktionen:

- [Quelltexte erstellen, Seite 19](#)
- [Quelltext formatieren, Seite 25](#)
- [Programme leichter schreiben, Seite 44](#)

3.5.1 Text schnell suchen

Mit dem Tastenkürzel [CTRL]-[F3] können Sie Text schnell suchen. Für die Rückwärtssuche verwenden Sie das Tastenkürzel [CTRL]-[SHIFT]-[F3].

Gesucht wird der markierte Text oder, falls kein Text markiert ist, das Wort an der Cursor-Position. Folgende Suchoptionen sind fest eingestellt:

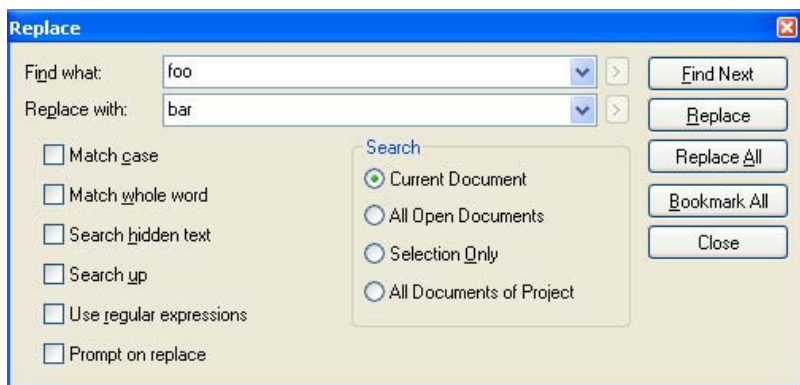
- Groß-Kleinschreibung spielt keine Rolle.
- Der Suchausdruck kann auch ein Teil eines Wortes sein.
- Es werden auch eingeklappte Textbereiche durchsucht.
- Alle offenen Dokumente werden durchsucht.

Bei der Schnellsuche können Sie keine regulären Ausdrücke verwenden und auch keine Textmarken erzeugen.

3.5.2 Text suchen und ersetzen

Sie können nach jedem Auftreten einer beliebigen Kombination von Zeichen suchen, zum Beispiel auch nach Groß- und Kleinbuchstaben, ganzen Wörtern, Teilen von Wörtern und regulären Ausdrücken (siehe [Reguläre Ausdrücke](#) auf [Seite 38](#)).

1. Wählen Sie den Menüeintrag **Edit ► Find** für eine Suche oder **Edit ► Replace** zum Ersetzen. Es erscheint ein Dialogfeld, das geöffnet bleibt, bis Sie es schließen.



2. Geben Sie im Eingabefeld **Find what** den gewünschten Suchausdruck an. Sie können auch einen der zuletzt benutzten Ausdrücke aus der Liste wählen.
3. Nur bei Ersetzen: Geben Sie im Eingabefeld **Replace With** den gewünschten Ersetzungsausdruck an. Sie können auch einen der zuletzt benutzten Ausdrücke aus der Liste wählen.
4. Stellen Sie die Optionen für die Suche ein:

Option	Beschreibung
Match case	<p>Option aktiviert: Es wird nur nach Text gesucht, der genau die Groß-/Kleinschreibung des Suchausdrucks hat.</p> <p>Option deaktiviert: Die Groß-/Kleinschreibung spielt bei der Suche keine Rolle.</p>

Option	Beschreibung
Match whole word	<p>Option aktiviert: Es wird nur nach Text gesucht, bei dem der Suchausdruck ein ganzes Wort ist.</p> <p>Option deaktiviert: Der Suchausdruck kann auch ein Teil eines Wortes sein.</p>
Search hidden text	<p>Die Option bezieht sich auf Faltbaren Textbereich definieren (siehe Seite 29).</p> <p>Option aktiviert: Es werden auch eingeklappte Textbereiche durchsucht.</p> <p>Option deaktiviert: Eingeklappte Textbereiche werden übersprungen.</p>
Search up	<p>Option aktiviert: Es wird in Richtung zum Dateianfang hin gesucht.</p> <p>Option deaktiviert: Es wird in Richtung zum Dateiende hin gesucht.</p>
Use regular expressions	<p>Die Option bezieht sich auf Reguläre Ausdrücke (siehe Seite 38).</p> <p>Option aktiviert: Der eingegebene Suchausdruck ist ein regulärer Ausdruck; das ist eine Methode, die variable Textmuster beschreibt.</p> <p>Option deaktiviert: Der eingegebene Suchausdruck ist konstanter Text.</p>
Prompt on replace	<p>Die Option ist nur in Verbindung mit Replace All einsetzbar.</p> <p>Option aktiviert: Bei jedem Vorkommen erscheint ein Dialogfenster, mit dem Sie das Ersetzen steuern.</p> <p>Option deaktiviert: Alle Vorkommen werden ohne Rückfrage ersetzt.</p>

5. Stellen Sie einen Bereich für die Suche ein:

Option	Beschreibung
Current Document	Die Suche beginnt im aktuellen Quelltext an der Cursor-Position. Wenn Text markiert ist, steht der Cursor hinter der Markierung.
All open Documents	Alle geöffneten Dateien werden durchsucht, beginnend mit dem aktuellen Quelltext.
Selection only	Nur der markierte Bereich wird durchsucht. Falls keine Markierung vorhanden ist, startet die Suche an der Cursor-Position.
All Documents of Project	Alle Dateien des Projekts ^a werden durchsucht, auch wenn der aktuelle Quelltext nicht zum Projekt gehört. Die Option ist für das Ersetzen nicht verfügbar. Die Suchergebnisse werden im Fenster Find im Info-Bereich angezeigt. Mit einem Doppelklick auf ein Ergebnis springt der Cursor in die entsprechende Zeile. Alternativ springen Sie mit den Pfeil-Schaltflächen im Fenster Find von einem Ergebnis zum nächsten.

a. Dateien aus dem Bereich `Other Files` sind von der Suche in jedem Fall ausgenommen.

6. Starten Sie die gewünschte Aktion durch eine der Schaltflächen.
 - **Find Next**: markiert das nächste Vorkommen des Suchausdrucks.
 - **Replace**: ersetzt die aktuelle Markierung durch den Ersetzungsausdruck und markiert das nächste Vorkommen des Suchausdrucks.
 - **Replace All**: ersetzt alle Vorkommen des Suchausdrucks durch den Ersetzungsausdruck.
 - **Bookmark All**: markiert alle Zeilen, in denen der Suchausdruck vorkommt, mit einer **Textmarke**.
7. Schließen Sie das Dialogfeld, indem Sie auf **Close** klicken. Sie können das Fenster auch geöffnet lassen und im Quelltext weiter arbeiten.

Mit der Option **All Documents of Project** schließt das Dialogfeld automatisch. Die Suchergebnisse finden Sie im Fenster **Find** (im Infobereich unten).

Anmerkungen

- Der Menüeintrag **Edit ► Find Next** sucht das nächste Vorkommen des Suchausdrucks. Die aktuellen Suchoptionen sind dabei gültig, auch wenn das Suchen-Dialogfenster geschlossen ist.
- Die Aktion **Replace** ersetzt die aktuelle Markierung nur dann, wenn die Markierung dem Suchausdruck entspricht.
- Vorsicht ist angebracht beim Ersetzen mit regulären Ausdrücken, die auch auf „nichts“ passen, z. B. „.+“ oder „a*“. In solchen Fällen können Schleifen entstehen, in denen so lange ersetzt wird, bis die Zeile zu lang ist.



- **Tipp**: Wenn Sie mit regulären Ausdrücken eine große Anzahl von Ersetzungen in einem oder gar allen geöffneten Dokumenten vornehmen wollen, können Sie sich zunächst mit **Find Next** und **Replace** vergewissern, dass Sie den Such- und den Ersetzungsausdruck korrekt formuliert haben, bevor Sie mit **Replace All** den Rest ersetzen lassen.

Beispiele für das Suchen von Text

Beispiele für das Suchen mit regulären Ausdrücken.

- Alle Leerzeichen oder Tabulatoren am Ende einer Zeile finden:

`[]+$`

Der Suchausdruck findet ein oder mehrere Leerzeichen oder Tabulatoren, auf die das Ende der Zeile folgt.

- Alles, was in einer Zeile steht, finden:

`^.+`

Der Suchausdruck findet den Anfang einer Zeile, auf den ein oder mehrere beliebige Zeichen folgen, bis zum Ende der Zeile.

- `$12.34` finden:

`\$12\.34`

Vor `.` und `$` steht jeweils der Backslash (`\`), weil diese beiden Zeichen Metazeichen sind. Metazeichen werden nicht als Zeichen selbst, sondern als Bestandteil eines Musters interpretiert. Der Backslash (als „Escape-Code“) hebt diese besondere Wirkung auf, so dass tatsächlich nach einem Punkt und nach einem Dollar-Zeichen gesucht wird.

- Eine Zeichenfolge finden, die in *TiCoBasic* als Variablenname gültig ist:

`\b[a-z][_a-z0-9]*`

Der Suchausdruck findet ein Wort, das mit einem Groß- oder Kleinbuchstaben beginnt, worauf kein, ein oder mehrere Buchstaben, Ziffern oder Unterstriche folgen.

- Bei verschachtelten Klammern den innersten Ausdruck finden:

`\([^\\(\\)]*\)`

Der Suchausdruck findet eine öffnende Klammer, auf die kein, ein oder mehrere beliebige Zeichen außer der öffnenden und der schließenden Klammer folgen, auf die wiederum eine schließende Klammer folgt.

- Wiederholungen finden:

`([0-9]+)-\1`

Das Suchmuster in Klammern (...) steht für eine oder mehrere Ziffern; durch die Klammern wird das Muster gespeichert. Anschließend folgen ein Bindestrich und mit `\1` wieder das gespeicherte Muster. Dieser reguläre Ausdruck würde also etwa `14-14` und `08-08` finden, aber nicht zum Beispiel `08-15`.

Beispiele für das Ersetzen von Text

Beispiele für das Ersetzen mit regulären Ausdrücken.

- Finde zwei Zahlen, die durch ein oder mehrere Leerzeichen getrennt sind:

`([0-9]+) + ([0-9]+)`

vertausche sie, und trenne sie durch einen Doppelpunkt:

`$2:$1`

- Um die folgenden Veränderungen simultan durchzuführen:

aus `X100000` soll werden `X100.000`, aus `Y100123` wird `Y100.123`

aus `Z600` soll werden `Z.600`

Suche nach: `([XYZ]) ([0-9]*) ([0-9] [0-9] [0-9])`

Ersetze durch: `$1$2.$3`

3.5.3 Reguläre Ausdrücke

Ein regulärer Ausdruck ist ein Suchausdruck, bei dem so genannte Metazeichen (siehe Liste unten) ein variables Suchmuster beschreiben. Die Metazeichen sind nur für das Suchen gültig, nicht für das Ersetzen.

Um mit einem regulären Ausdruck beim Suchen/Ersetzen zu verwenden, müssen Sie die Option `Use regular expressions` im Dialogfenster aktivieren. Rechts neben den Eingabefeldern werden dadurch Schaltflächen „>“ aktiv, über die Sie Metazeichen eingeben können.

Die Syntax für reguläre Ausdrücke ist im .NET-Framework 2.0 definiert. Eine ausführliche Beschreibung finden Sie im Internet unter der Adresse <http://msdn2.microsoft.com> (nach „reguläre Ausdrücke“ suchen).

Meta- zeichen :	Bedeutung:
.	Ein beliebiges einzelnes Zeichen. Beispiel: Das Muster <code>Ma.s</code> trifft unter anderem zu auf <code>Mais</code> , <code>Mars</code> und <code>Maus</code> , nicht aber auf <code>Mas</code> .

Meta- zeichen :	Bedeutung:
[]	Ein beliebiges Zeichen von denen, die <ol style="list-style-type: none"> explizit in den eckigen Klammern angegeben sind, oder eines aus dem Bereich von Zeichen, der durch den Bindestrich (-) definiert ist. Beispiele: Der Suchausdruck <code>h[aeiou][a-z]t</code> findet unter anderem: <code>hart</code> , <code>halt</code> , <code>heft</code> , <code>holt</code> und <code>hut</code> . Das Muster <code>[A-Za-z]</code> entspricht einem beliebigen Buchstaben. Der reguläre Ausdruck <code>x[0-9]</code> entspricht <code>x0</code> , <code>x1</code> usw. bis <code>x9</code> .
[^]	Ein beliebiges Zeichen außer denen, die hinter dem Caret (^, „Dach“) stehen. Beispiel: Das Muster <code>h[^uo]t</code> passt unter anderem auf <code>hat</code> und <code>hit</code> , aber nicht auf <code>hut</code> oder <code>hot</code> .
^	Der Anfang einer Zeile (Spalte 1). Beispiel: Der Suchausdruck <code>^Anfang</code> entspricht nur dann dem Text <code>Anfang</code> , wenn dieser ganz vorne in einer Zeile steht.
\$	Das Ende einer Zeile (letzte Spalte). Benutzen Sie dieses Zeichen und nicht das Zeilenschaltzeichen <code>\n</code> , wenn Sie nach einem Ausdruck suchen, der am Ende einer Zeile steht. Beispiel: Der Suchausdruck <code>Ende\$</code> findet die Zeichenfolge <code>Ende</code> nur dann, wenn <code>Ende</code> das letzte Wort einer Zeile ist.
\b	Der Anfang eines Wortes.
\B	Das Ende eines Wortes.

Meta- zeichen :	Bedeutung:
\n	<p>Zeilenschaltzeichen (Newline). Zu verwenden bei Ausdrücken, die sich über mehrere Zeilen erstrecken.</p> <p>Hinter \n dürfen nicht die Operatoren *, + oder { } stehen. Um ein Suchmuster am Ende einer Zeile zu finden, sollten Sie unbedingt den Operator \$ und nicht das Zeilenschaltzeichen verwenden.</p>
()	<p>Der Text innerhalb der Klammern wird als Muster in internen Registern abgelegt. Der Inhalt eines Registers kann an anderer Stelle im Suchausdruck oder im Ersetzungsausdruck wieder abgerufen werden.</p> <p>Bis zu 9 gespeicherte Muster sind zulässig. Sie werden in der Reihenfolge ihres Auftretens in dem regulären Ausdruck durchnummeriert. Der Rückbezug auf das Muster lautet \$x im Ersetzungsausdruck und \x im Suchausdruck, mit x = 1 ... 9.</p> <p>Beispiel: Der Ausdruck ([a-z]+) ([a-z]+) findet <i>isses so. \$2 \$1</i> würde ihn ersetzen durch <i>so isses</i>.</p>
*	<p>Beliebig häufiges Auftreten des vorangehenden Zeichens oder Ausdrucks – kein Mal, einmal oder mehrmals.</p> <p>Beispiel: <i>hu*f</i> findet unter anderem <i>hf</i>, <i>huf</i> und <i>huuuf</i>.</p>
?	<p>Kein oder genau ein Auftreten des vorangehenden Zeichens oder Ausdrucks.</p> <p>Beispiel: <i>hu?f</i> findet <i>hf</i> und <i>huf</i>, nicht aber <i>huuuf</i>.</p>
+	<p>Mindestens ein (d.h. ein- oder mehrmaliges) Auftreten des vorangehenden Zeichens oder Ausdrucks.</p> <p>Beispiel: <i>hu+f</i> findet <i>huf</i> und <i>huuf</i>, nicht aber <i>hf</i>.</p>
	<p>Entweder der Ausdruck auf der linken Seite des Striches oder der auf der rechten Seite.</p> <p>Beispiel: <i>huf huuf</i> findet <i>huf</i> oder <i>huuf</i>.</p>

Meta- zeichen :	Bedeutung:
\	<p>Hebt die besondere Wirkung aller Operatoren auf, so dass sie wie normaler, konstanter Text behandelt werden. Um nach dem Backslash \ zu suchen, muss also \\ verwendet werden.</p> <p>Beispiel: ^a bedeutet, dass nach einem a am Anfang einer Zeile gesucht wird, dagegen sucht \^a nach der Zeichenfolge ^a.</p>

3.5.4 Kontrollstruktur markieren

Sie können die Zeilen einer Kontrollstruktur oder eines Programmabschnitts auf einmal markieren, z.B. um verschachtelte Strukturen optisch zu prüfen. Hierzu setzen Sie den Cursor auf eines der Kennwörter der Struktur und wählen im Kontextmenü des Quelltextfensters (Klick mit rechter Maustaste) den Eintrag **Mark Control block**.

Es kann nur eine Kontrollstruktur gleichzeitig markiert werden.

Die Markierung wird mit **Unmark Control block** (im Kontextmenü) wieder gelöscht. Hierbei ist es gleichgültig, an welcher Stelle der Cursor steht.

Folgende Kontrollstrukturen werden erkannt:

- Programmabschnitte **Init:**, **Event:**, **Finish:**
- **Do ... Until**
- **For ... Next**
- **If ... EndIf**
- **SelectCase ... EndSelect**
- **Function ... EndFunction**
- **Sub ... EndSub**
- **Lib_Function ... Lib_EndFunction**
- **Lib_Sub ... Lib_EndSub**

Alle Kontrollstrukturen sind auch faltbare Bereiche (siehe [Faltbaren Textbereich definieren](#), [Seite 29](#)).

3.5.5 Textmarken nutzen

Mit Textmarken können Sie, ähnlich einem Lesezeichen, bestimmte Zeilen in einem Quelltext kennzeichnen. Derart markierte Zeilen können einfach angesprungen werden.

Folgende Aktionen stehen zur Verfügung:

- Textmarke setzen

Sie können eine Textmarke auf eine Zeile setzen, indem Sie entweder in der Editor-Leiste den Befehl `Toggle Bookmark` wählen oder im Dialogfeld `Replace` auf `Bookmark All` klicken.

Mit `Toggle Bookmark` werden gesetzte Textmarken wieder einzeln entfernt.

- Zur nächsten Textmarke springen

Wählen Sie in der Editor-Leiste den Befehl `Next Bookmark`.

- Zur vorherigen Textmarke springen

Wählen Sie in der Editor-Leiste den Befehl `Previous Bookmark`.

- Alle Textmarken entfernen

Wählen Sie in der Editor-Leiste den Befehl `Delete all Bookmarks`.

Einzeln werden Textmarken mit `Toggle Bookmark` entfernt.

Textmarken werden mit der Datei zusammen abgespeichert.

3.5.6 Zu einer Programmzeile springen

Sie können zu einer bestimmten Programmzeile im Quelltext springen, indem Sie auf die Zeilennummer in der Statusleiste doppelklicken oder im Menü `Edit` den Eintrag `Goto Line` wählen. Es öffnet sich ein Dialogfenster, in dem Sie die Nummer der gewünschten Programmzeile eingeben.

Um Zeilennummern anzuzeigen, muss die Option `show line numbers` unter `Editor - General` (siehe [Seite 64](#)) aktiv sein.

3.5.7 Zur Deklaration eines Befehls oder Variablen springen

Sie können von einem Variablennamen aus direkt zu deren Deklaration springen. Diese Möglichkeit gilt für alle selbst deklarierten Namen:

lokale Variablen, Felder, Befehle ([Sub](#), [Function](#)) und symbolische Namen ([#Define](#)).

Sie springen zur Deklaration, indem Sie den Cursor auf den selbst deklarierten Namen setzen und dann entweder im Kontextmenü (rechte Maustaste) den Eintrag [Jump to Declaration](#) aufrufen, oder die Schaltfläche [Jump to Declaration](#) in der Editor-Leiste aufrufen.

Der Sprung zur Deklaration ist nur möglich, wenn die Option [Parse Declarations](#) unter [Editor - General](#) (siehe [Seite 64](#)) aktiv ist.

Für Befehle aus den Standard-Include-Dateien sowie für globale Variablen [Par](#) steht die Funktion verständlicherweise nicht zur Verfügung.

3.5.8 Zwischen Sprungzielen wechseln

Sie können zwischen bereits verwendeten Sprungzielen hin- und herwechseln, indem Sie auf die Pfeile [Jump back-ward](#) / [Jump forward](#) in der [Editor-Leiste](#) klicken.



Die Oberfläche erstellt automatisch einen Verlauf der verwendeten Sprungziele. Folgende Sprungaktionen werden in den Verlauf aufgenommen:

- [Zur Deklaration eines Befehls oder Variablen springen](#)
- [Zu einer Programmzeile springen](#)
- Doppelklick in einem der folgenden Fenster des [Info-Bereichs](#):
 - [Infofenster](#) (Compiler-Fehler und -Warnungen)
 - [ToDo-Liste](#)
 - [Fenster „Global Variables“](#)
 - [Fenster „Declarations“](#)
- [Text suchen und ersetzen](#)

Angesprungene Textmarken werden nicht in den Verlauf aufgenommen, siehe [Textmarken nutzen](#).

Wenn Sie mitten aus dem Verlauf heraus ein neues Ziel anspringen, bleiben die vorherigen Sprungziele im Verlauf erhalten und das neue Ziel wird zum letzten Sprungziel im Verlauf.

Wenn Sie ein Projekt öffnen, wird der Verlauf gelöscht.

3.6 Programme leichter schreiben

Die folgenden Funktionen erleichtern das Programmieren erheblich:

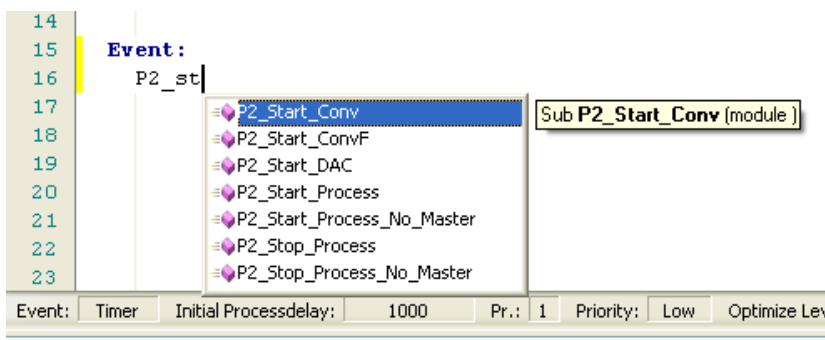
- Befehle und Variablen automatisch vervollständigen, Seite 44
- Befehlsparameter anzeigen, Seite 46
- Textbausteine einfügen, Seite 46
- Eigene Befehle und Variablen dokumentieren, Seite 27
- Deklaration eines Befehls oder Variablen anzeigen, Seite 47
- Deklarationen einer Datei anzeigen, Seite 48
- Verwendete globale Variablen und Felder anzeigen, Seite 48
- Aufrufkürzel zu Dateien / Ordnern einfügen, Seite 49

Hier finden Sie weitere Editor-Funktionen:

- Quelltexte erstellen, Seite 19
- Quelltext formatieren, Seite 25
- Suchen, markieren und ersetzen, Seite 32

3.6.1 Befehle und Variablen automatisch vervollständigen

Sie können Schlüsselwörter, Befehls- und Variablennamen und sogar Textbausteine automatisch vervollständigen, indem Sie die Tastenkombination [CTRL-SPACE] drücken.



Durch die automatische Vervollständigung müssen Sie in den meisten Fällen Schlüsselwörter, Befehle und Variablen nicht vollständig aus-schreiben.

Um die automatische Vervollständigung zu nutzen, gehen Sie vor wie folgt:

1. Schreiben Sie die ersten Buchstaben des Wortes und drücken [CTRL-SPACE].

Es wird eine Wortliste angezeigt, deren Einträge den bisher eingegebenen Text vervollständigen können.

Wenn Sie die Vervollständigung nach einem Leerzeichen aufrufen, enthält die Liste alle verfügbaren Kennwörter.

2. Wählen Sie den gewünschten Listeneintrag mit der Maus oder mit den Pfeiltasten aus.

Rechts neben dem gerade markierten Listeneintrag wird nach einem Moment eine von mehreren Erläuterungen angezeigt:

- die Deklaration des Befehls oder der Variablen
- die Angabe „Reserved Keyword“
- der vollständige Textbaustein (siehe unten [Textbausteine einfügen](#))

3. Wenn Sie weiteren Text eingeben, wird die Liste nicht automatisch aktualisiert. Drücken Sie zur Aktualisierung der Liste erneut die Tastenkombination [CTRL-SPACE].

4. Sie übernehmen den markierten Eintrag aus der Liste am besten durch Eingeben einer Klammer (bei einem Befehl) oder eines Leerzeichens.


Sie können stattdessen aber auch die [EINGABE]-Taste verwenden oder ein anderes, nicht-alphanumerisches Zeichen eingeben.

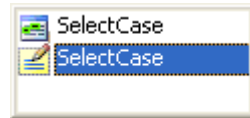
Automatisches Vervollständigen ist nur möglich, wenn die Option `Parse Declarations` unter [Editor - General](#) (siehe [Seite 64](#)) aktiv ist.

3.6.2 Textbausteine einfügen

Der Editor bietet Ihnen die Möglichkeit, mit Textbausteinen (code snippets) zu arbeiten, welche in einer vorgegebenen Sammlung zusammengefasst sind. Je nach Definition kann ein Textbaustein wenige Zeichen, mehrere Zeilen oder ein ganzes Programm einfügen.

Sie fügen einen Textbaustein an der Cursor-Position ein, indem Sie eine der folgenden Aktionen ausführen:

- Sie geben die ersten Zeichen eines Textbaustein-Kennworts ein, z.B. `Sele` für eine `SelectCase`-Struktur, drücken die Tastenkombination [CTRL-SPACE] und wählen aus der Liste den Textbaustein 



`SelectCase`.

Siehe auch [Befehle und Variablen automatisch vervollständigen](#).

- Sie rufen über Kontextmenü oder Editor-Leiste die Funktion `Code-snippets` auf. Es erscheint eine Auswahlliste mit Ordnern, die jeweils mehrere Textbausteine (oder weitere Ordner) enthalten.

Sie navigieren durch die Menüs per Maus oder per Tastatur. Folgende Tasten sind belegt:

- Pfeil oben/unten: Listeneintrag markieren.
- Eingabe: Markierten Textbaustein einfügen oder neue Menüebene öffnen.
- Backspace: In vorige Menüebene zurückkehren.

Sobald Sie einen Textbaustein markiert haben, erscheint rechts davon das zugehörige Tastaturkürzel.

- Sie geben das Tastaturkürzel des Textbausteins ein, gefolgt von [TAB].

Um die Liste der Textbausteine und der Tastaturkürzel zu sehen, öffnen Sie die Datei `<codesnippets.xml>` im Ordner `C:\ADwin\TiCo-Basic\Common\` mit einem Browser.

3.6.3 Befehlsparameter anzeigen

Bei Befehlen werden die zugehörigen Übergabeparameter mit einer Beschreibung automatisch als Tooltip angezeigt, sobald Sie nach dem Befehlsnamen eine öffnende Klammer eingeben. Im Tooltip wird derjenige Übergabeparameter fett dargestellt, den Sie gerade eingeben.

```
27 Event:  
28   max = my_func(  
    Function my_func(val1, val2) As Long  
    val1: Ein Einzeiler für den ersten Übergabeparameter
```

Der Tooltip verschwindet, sobald der Cursor außerhalb der Klammern um die Parameter steht. Sie können die Anzeige wieder aktivieren, wenn Sie die öffnende Klammer überschreiben. Alternativ können Sie über Kontextmenü oder Editor-Leiste die Funktion `Declaration Info` aufrufen, die die vollständige Deklaration des Befehls anzeigt.

Auch selbstdefinierte Befehle und ihre Parameter können Sie mit Hinweistexten dokumentieren und auf gleiche Weise anzeigen lassen, siehe „[Eigene Befehle und Variablen dokumentieren](#)“.

Die automatische Anzeige von Befehlsparametern ist nur möglich, wenn die Option `Parse Declarations` unter [Editor - General](#) (siehe [Seite 64](#)) aktiv ist.

3.6.4 Deklaration eines Befehls oder Variablen anzeigen

Sie können zu Befehlen, Variablen und anderen deklarierten Kennwörtern die zugehörige Deklaration sowie Hinweise als Tooltip anzeigen, indem Sie

- den Mauszeiger auf das Wort positionieren.

Die Deklaration wird nur dann automatisch angezeigt, wenn die Option `Display quick info on mouse over` unter [Editor - General](#) (siehe [Seite 64](#)) aktiv ist.

- den Cursor auf das Kennwort setzen und die Taste `[F2]` drücken.
- den Cursor auf das Kennwort setzen und in der Editor-Werkzeugliste oder im Kontextmenü den Eintrag `Declaration Info` wählen.

Diese Möglichkeit gilt für alle Kennwörter, die zum Sprachumfang von *TiCoBasic* gehören oder selbst deklariert wurden: lokale und globale Variablen, Felder, Befehle ([Sub](#), [Function](#)) und symbolische Namen ([#Define](#)).


Die Anzeige der Deklaration ist nur möglich, wenn die Option `Parse Declarations` unter [Editor - General](#) (siehe [Seite 64](#)) aktiv ist.

3.6.5 Deklarationen einer Datei anzeigen

Sie können alle zu einer Quelltextdatei gehörenden Deklarationen, Include- und Library-Dateien anzeigen, indem Sie im [Info-Bereich](#) das [Fenster „Declarations“](#) in den Vordergrund stellen.

Die Deklarationen können nur angezeigt werden, wenn die Option `Parse Declarations` unter [Editor - General](#) (siehe [Seite 64](#)) aktiv ist.




3.6.6 Verwendete globale Variablen und Felder anzeigen

Um globale Variablen und Felder anzuzeigen, die im aktiven Quelltext und im zugehörigen Projekt (falls vorhanden) verwendet werden, drücken Sie die Schaltfläche `Scan Global Variables`  im [Parameterfenster](#) (siehe auch [Seite 78](#)) drücken.

Sie erhalten zwei Anzeigen:

- im [Fenster „Global Variables“](#) werden alle verwendeten globalen Variablen und Felder (siehe [Seite 86](#)) angezeigt.
- im [Parameterfenster](#) werden alle verwendeten globalen Variablen farbig hervorgehoben (globale Felder nicht).

Die Hervorhebung ist farbig je nach Verwendung der Parameter:

- Grün: Parameter wird nur im aktiven Quelltext verwendet.  2 1336227
- Rot: Parameter wird im aktiven Quelltext verwendet, aber auch in einem anderen Quelltext des Projekts.  1 707279
- Blau: Parameter wird im Projekt verwendet, jedoch nicht im aktiven Quelltext.  9 5B12H

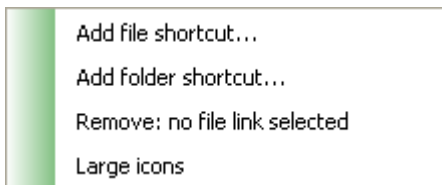
Mit der Schaltfläche `Clear Scan`  werden beide Anzeigen wieder gelöscht.

Bei Änderungen im Quelltext werden die Anzeigen nicht automatisch aktualisiert. Rufen Sie stattdessen `Scan Global Variables` erneut auf.

3.6.7 Aufrufkürzel zu Dateien / Ordnern einfügen

Sie können Links zu externen Dateien, Programmen oder Ordner als Aufrufkürzel unterhalb der **ADtools**-Leiste einfügen.

Zum Einfügen klicken sie mit der rechten Maustaste auf die **ADtools**-Leiste und wählen im Kontextmenü den Menüeintrag `Add file shortcut` oder `Add folder shortcut`.



Im Dateiauswahlfenster legen

Sie fest, zu welcher Datei der neu angelegte Link führen soll.

Mit einem Doppelklick auf den Link wird die Datei oder der Ordner aktiviert; das Betriebssystem des Rechners führt dann die Aktion aus, die mit dem Dateityp verknüpft ist.

Zum Löschen klicken sie mit der rechten Maustaste auf den Link und wählen im Kontextmenü den Menüeintrag `Remove: <file name>`.

3.7 TiCo-Binärdatei zum TiCo-Prozessor übertragen

Sie können mit der Oberfläche *TiCoBasic* eine vorhandene Binärdatei zum *TiCo*-Prozessor oder in den *TiCo*-Bootloader übertragen.

3.7.1 TiCo-Binärdatei übertragen

Um eine Binärdatei als Prozess zum *TiCo*-Prozessor zu übertragen, gehen Sie vor wie folgt:

- Erzeugen Sie eine Binärdatei mit `Build ► Make Bin File`; siehe auch [Seite 57](#).

Stellen Sie dazu auch die Eigenschaften (Priorität, Prozessnummer, Prozessstyp etc.) für den Prozess ein, siehe [Seite 60](#).

- Wählen Sie im Menü `Tools` den Eintrag `Load Bin File` und im nächsten Fenster die Binärdatei.

Bestätigen Sie die Auswahl mit `Öffnen`; *TiCoBasic* überträgt nun die Binärdatei als Prozess zum *TiCo*-Prozessor, der in den Compiler-Optionen eingestellt ist.

- Der Prozess wird automatisch gestartet.

3.7.2 TiCo-Bootloader programmieren

Der *TiCo*-Bootloader lädt und startet ausgewählte *TiCoBasic*-Prozesse beim Starten der *ADwin*-Hardware automatisch.

Um den *TiCo*-Bootloader zu programmieren, gehen Sie vor wie folgt:

- Erzeugen Sie eine Binärdatei mit **Build ► Make Bin File**; siehe [Das Menü „Build“ Seite 57](#).

Stellen Sie dabei schon die Eigenschaften (Priorität, Prozessnummer, Prozessstyp etc.) für den Prozess ein.

- Wählen Sie im Menü **Tools** den Eintrag **Bootloader...**

Das Dialogfenster **Bootloader** öffnet sich.



- Klicken Sie auf die Schaltfläche **Load Bootloader** und wählen im nächsten Fenster die Binärdatei.
- Geben Sie mit der Schaltfläche **Enable Bootloader** den Bootloader-Betrieb frei. Damit werden beim Einschalten der *ADwin*-Hardware die Prozesse der Binärdatei automatisch gestartet.

Mit der Schaltfläche `Disable Bootloader` können Sie den Bootloader-Betrieb zeitweilig sperren und später mit `Enable Bootloader` wieder freigeben.

- Schließen Sie das Dialogfenster mit `Close`.
- Beim nächsten Einschalten der *ADwin*-Hardware wird der Prozess der Binärdatei automatisch gestartet.

3.8 Projekte verwalten

Sie können beliebig viele Quelltexte, Include-Dateien, Library-Dateien sowie Dateien anderen Typs gemeinsam als Projekt verwalten, beispielsweise wenn Sie eine Anwendung mit mehreren Prozessen programmieren. Die Dateien werden im [Projektfenster \(Seite 76\)](#) dargestellt. Es kann jeweils nur ein einziges Projekt geöffnet sein.


Mit dem Projekt wird die Darstellung der Bedienoberfläche gespeichert: Fenstergröße, Fensterposition, geöffnete Projektdateien. Außerdem (siehe [Dialogfenster „Project Settings“](#)) können Sie weitere Programmeinstellungen mit dem Projekt speichern lassen sowie Programmaufrufe vor und nach dem Kompilieren (`Prebuild` / `Post-build`). Beim Öffnen des Projekts werden die gespeicherten Einstellungen wieder hergestellt.

Für die Zusammenstellung der Projektdateien gelten folgende Regeln (siehe auch [Kapitel 6.1 „Prozessverwaltung“](#)):

- Mindestens ein Prozess muss hohe Priorität haben.
- Erlaubt ist nur ein einziger zeitgesteuerter Prozess mit hoher Priorität.
- Es kann einen zeitgesteuerten Prozess mit niedriger Priorität geben, wenn gleichzeitig ein Prozess mit hoher Priorität zum Projekt gehört.

Sie können einen zeitgesteuerten und einen extern gesteuerten Prozess miteinander kombinieren. Wenden Sie sich in diesem Fall bitte an unseren Support (support@adwin.de); wir informieren Sie über die erforderlichen Vorkehrungen.

Folgende Funktionen stehen in einem Projekt zur Verfügung.

- Projekt verwalten
 - Neues Projekt erstellen: File ▶ New Project.
 - Gespeichertes Projekt öffnen: File ▶ Open Project.
 - Projekt speichern: File ▶ Save Project / Save Project As.
Es werden nur die Projekteinstellungen gespeichert. Änderungen in Projektdateien können z.B. mit Save all Files of Project gespeichert werden.
 - Projekt schließen: File ▶ Close Project.
Mit dem Projekt werden auch die zum Projekt gehörenden Dateien geschlossen.
 - Projekteinstellungen ändern: Options ▶ Project Settings
oder Schaltfläche Project Settings  in der Projektleiste;
siehe [Dialogfenster „Project Settings“](#).
- Dateien ins Projekt einbinden
 - Eine Datei in das Projekt einbinden: File ▶ Add to Project oder Add to Project im Kontextmenü des Quelltextfensters.
 - Alle geöffneten Dateien in das Projekt einbinden: Add Open Files To Project im Kontextmenü des [Projektfensters](#), siehe [Seite 76](#).
 - Eine Datei eines anderen Typs in das Projekt einbinden: Add other File to Project im Kontextmenü des [Projektfensters](#), siehe [Seite 76](#).
Die Datei wird im Projektfenster unter Other Files aufgeführt.

Wir empfehlen, die Projektdateien im gleichen Ordner zu speichern wie das Projekt (*.abp) oder einem zugehörigen Unterordner. Damit sind beim Kopieren des Ordners alle Dateien im neuen Projekt enthalten und die Dateien können unabhängig vom alten Projekt benutzt werden.

Beim Einbinden einer Datei ins Projekt wird der Dateipfad relativ zum Projektordner gespeichert. Nur wenn die Datei auf einem anderen Laufwerk liegt, wird der Dateipfad absolut gespeichert.

- Dateien eines Projekts verwalten
 - Eine Datei des Projekts öffnen und zum aktiven Quelltext machen, mit einem Doppelklick auf die Datei im [Projektfenster](#), siehe [Seite 76](#).
 - Markierte Dateien aus dem Projekt entfernen: Taste DEL oder Remove from Project im Kontextmenü des [Projektfensters](#), siehe [Seite 76](#).
 - Alle Dateien des Projekts auf einmal speichern, mit Save all Files of Project im Kontextmenü des [Projektfensters](#), siehe [Seite 76](#).
- Projekt-Quelltexte übersetzen
 - Alle Dateien des Projekts kompilieren: Build ► Compile.
 - Binärdateien des Projekts erzeugen: Build ► Make Bin File.

Wenn im [Dialogfenster „Compiler Options“](#) die Option Autostart aktiv ist, werden Binärdateien automatisch in die ADwin-Hardware übertragen und gestartet.

- Alle Dateien des Projekts durchsuchen, darunter auch die nicht geöffneten Dateien des Projekts (ausgenommen Dateien aus dem Projektbereich Other Files).

Aktivieren Sie dazu die Option All Documents of Project im Suchen-Fenster (siehe [Kapitel 3.5.2 „Text suchen und ersetzen“](#)). Diese Option ist für das Ersetzen nicht verfügbar.

- Globale Variablen im Projekt hervorheben (siehe [Kapitel 3.6.6](#)).
- Windows Explorer mit dem Pfad der markierten Datei öffnen, mit Open Path in Explorer Window im Kontextmenü des [Projektfensters](#).

Die Befehle zur Projektverwaltung finden Sie im Kontextmenü des [Projektfensters](#) (siehe auch „[Projektfenster](#)“ auf [Seite 76](#)), in der Projektleiste des [Projektfensters](#) oder im Menü File ([Kapitel 3.9.1](#)).

3.9 Menüs

Sie finden in der Menüleiste folgende Funktionen:

- File: Dateien und Projekte verwalten. ([Seite 55](#))

- Edit: Quelltexte editieren. ([Seite 56](#))
- View: Fenster und Leisten anzeigen. ([Seite 56](#))
- Build: Ausführbare Programme erzeugen. ([Seite 57](#))
- Options: Optionen aller Art einstellen. ([Seite 58](#))
- Debug: Hilfe zur Fehlersuche. ([Seite 71](#))
- Tools: Verschiedene Hilfsfunktionen. ([Seite 73](#))
- Window: Quelltextfenster anordnen. ([Seite 74](#))
- Help: Hilfe, Versions- und Lizenz-Informationen. ([Seite 74](#))

3.9.1 Das Menü „File“

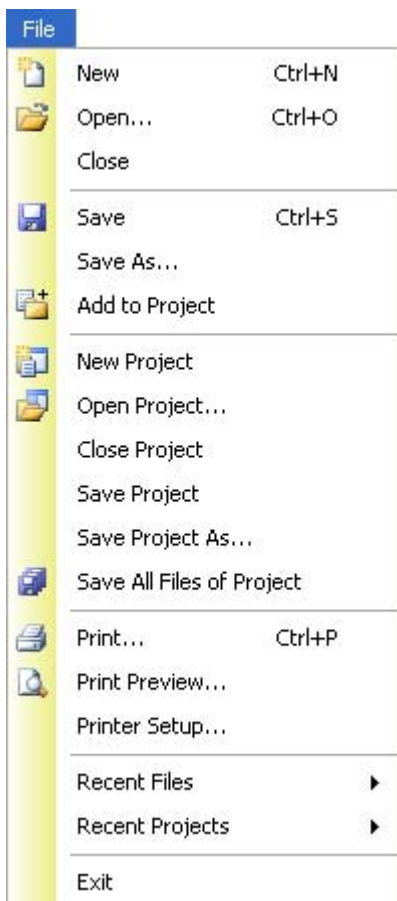
Das Menü `File` enthält Befehle zum Verwalten von Dateien und Projekten.

Mit den Befehlen können Sie Dateien öffnen, neu anlegen, speichern oder schließen. Sie können beliebig viele Quelltexte bearbeiten.

In gleicher Weise wie Dateien können Sie auch ein Projekt öffnen, speichern und neu erzeugen; es kann nur ein einziges Projekt gleichzeitig geöffnet sein kann. Weitere Befehle sind über das Projektfenster zugänglich (siehe [Kapitel 3.10.2](#)).

Das Menü enthält auch die Druckfunktionen (Drucken, Druckvorschau und Drucker-einrichtung).

Unter `Recent Files` und `Recent Projects` wird eine Liste der 10 zuletzt geöffneten Dateien und Projekte angezeigt.










3.9.2 Das Menü „Edit“

Das Menü **Edit** enthält die nach den Windows-Konventionen üblichen Editier-Funktionen.

Darüber hinaus enthält das Menü eine Such-Funktion (**Find**, **Find Next**) sowie eine Ersetzen-Funktion (**Replace**); siehe [Text suchen und ersetzen](#).

Wir empfehlen Ihnen nicht, mit **Cut** and **Paste** Zeichen oder Zeilen aus anderen Programmen in einen Quelltext einzufügen. Es kann hierbei zu unvorhergesehenen Fehlfunktionen kommen.

Edit		
	Undo	Ctrl+Z
	Redo	Ctrl+Shift+Z
<hr/>		
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
<hr/>		
	Select All	Ctrl+A
<hr/>		
	Find...	Ctrl+F
	Find Next	F3
	Replace...	Ctrl+H
	Goto Line..	Ctrl+G

3.9.3 Das Menü „View“

Im Menü **View** stellen Sie ein, welche Leisten der Entwicklungsumgebung angezeigt werden:

- die Werkzeugleiste (Standard Toolbar)
- die Editor-Leiste
- die ADtools-Leiste
- die Statuszeile

Näheres zum [Prozessfenster](#) finden Sie in [Kapitel 3.10.4 auf Seite 80](#), zur Werkzeugleiste siehe [Abb. 2](#).

View	
<input checked="" type="checkbox"/>	Standard Toolbar
<input checked="" type="checkbox"/>	Editor Toolbar
<input checked="" type="checkbox"/>	ADtools Toolbar
<input checked="" type="checkbox"/>	Statusbar
<hr/>	
	Restore Default Layout


Mit `Restore Default Layout` stellen Sie mit einem Tastendruck die Standard-Ansicht wieder her, die beim ersten Öffnen des Programms *TiCoBasic* aktiv war, darunter auch die Einstellungen der [Toolbox](#).

3.9.4 Das Menü „Build“

Im Menü `Build` können Sie Quelltext übersetzen:

- mit `Compile` in einen Prozess.
- mit `Make Bin File` in eine Binärdatei.
- mit `Make Lib File` in eine Library.



Bitte beachten Sie: Vor dem Kompilieren werden automatisch alle geänderten Quelltexte, Library- und Include-Dateien gespeichert. (AutoSave) 

Eine Änderung kann auch durch automatisches Einrücken von Textzeilen (siehe [Kapitel 3.4.3 auf Seite 26](#)) stattfinden, beispielsweise beim Öffnen einer vorher nicht formatierten Datei.

`Compile` ist der umfassendste Befehl des Menüs: Er übersetzt das gesamte Projekt (oder den einzelnen Quelltext) und überträgt den erzeugten Binär-code als Prozess auf den *TiCo*-Prozessor.

Der Prozess wird auf dem *TiCo*-Prozessor automatisch gestartet.

Fehler und Warnungen beim Kompilieren werden im [Infofenster](#) angezeigt. Ein Doppelklick auf die Fehlermeldung markiert die betreffende Zeile rot.

`Make Bin File` ist nur für lizenzierte *TiCoBasic*-Benutzer verfügbar. Der Befehl übersetzt das gesamte Projekt (oder den einzelnen Quelltext) in Binär-code und speichert diesen automatisch in einer Datei ab. Die Binär-datei wird mit dem Namen und im Verzeichnis der Quelltext-Datei abgelegt, jedoch mit der Dateierdung `.TIX`. Hierbei steht `x` für den

Prozessortyp (siehe auch [Make Lib File](#) Das Menü „Options“, Dialogfenster „Process Options“).



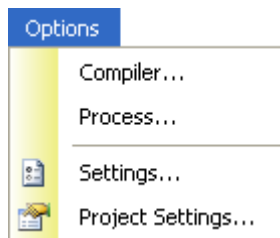
Eine Binärdatei mit der Endung `<*.TI2>` beispielsweise können Sie auf den *TiCo*-Prozessor vom Typ *TiCo2* übertragen. Sie können die Binärdatei mit *TiCoBasic* übertragen (siehe [Kapitel 3.7.1 „TiCo-Binärdatei übertragen“](#)) oder aber aus einer Entwicklungsumgebung (wie C oder Visual Basic).

Make Lib File ist nur für lizenzierte *TiCoBasic*-Benutzer verfügbar. Der Befehl übersetzt den aktiven Quelltext – die Datei muss mit dem Dateityp *LibFile* gespeichert sein – in Binärcode und speichert diesen automatisch als Library-Datei ab. Die Library wird mit dem Namen und im Verzeichnis der Quelltext-Datei abgelegt, jedoch mit der Dateiendung `.TLx`. Hierbei steht `x` wieder für den Prozessortyp (s.o.).

Anschließend können Sie die Library in andere Quelltexte einbinden und auf deren Funktionen und Unterprogramme zugreifen (siehe [Kapitel 4.5.3 auf Seite 116](#)).

3.9.5 *Make Lib File* Das Menü „Options“

Im Menü *Options* können Sie eine Reihe von Optionen einstellen, die unmittelbar wirksam werden. Zu jedem Menüpunkt wird ein eigenes Dialogfenster aufgerufen, in dem Sie Ihre Einstellungen vornehmen.



Dialogfenster „Compiler Options“

Die Einstellungen, die Sie in diesem Dialogfenster machen (bitte von oben nach unten), werden für jedes Kompilieren eines Quelltextes verwendet. Insbesondere sind dies Informationen über das *ADwin*-System und den *TiCo*-Prozessor, auf dem die übersetzten Quelltexte als Prozess laufen sollen.

Wenn Sie Quelltexte für verschiedene *TiCo*-Prozessoren kompilieren möchten, müssen Sie die Einstellungen in diesem Dialogfenster für jeden *TiCo*-Prozessor neu anpassen.



Abb. 3 – Das Dialogfenster Compiler Options

- **System:** Wählen Sie den Eintrag aus, der Ihrem *ADwin*-System entspricht.
- **ADwin CPU:** Wählen Sie den Prozessor des *ADwin*-Systems.
- **Device No.:** Wählen Sie die Gerätenummer, mit der das gewünschte *ADwin*-System angesprochen werden kann.

Sie vergeben die Gerätenummer mit dem Programm <ADconfig.exe>. Die Werkseinstellung ist 150 Hex.

- **Processor:** Wählen Sie den Typ des *TiCo*-Prozessors auf der Hardware (Modul).
- **Module Address** (nur für *ADwin-Pro II*-System): Wählen Sie die Adresse des Moduls, auf dem sich der *TiCo*-Prozessor befindet.
- **Do Not access the device:** Bei deaktivierter Option wird die beim Kompilieren erzeugte Binärdatei automatisch zur Hardware übertragen. Die Hardware muss daher für das Kompilieren erreichbar sein.

Bei aktiver Option können Quelltexte auch dann für die eingestellte ADwin-Hardware kompiliert werden, wenn sie nicht mit dem PC verbunden ist.

Beachten Sie: Die Menüeinträge zum Übertragen von Bootloader- oder Binärdateien ([Kapitel 3.9.7 „Das Menü „Tools““](#), [Seite 73](#)) sind nur aktiv, wenn die Option deaktiviert ist.

- Remember Device No.: Bei aktiver Option wird die zuletzt verwendete Device No. (siehe oben) beim Schließen des Programms gespeichert; beim Neustart wird diese Nummer automatisch eingestellt.

Bei deaktivierter Option wird die Device No. nicht gespeichert. Beim Start von *TiCoBasic* wird die zuletzt – bei aktiver Option – gespeicherte Device No. verwendet.

Dialogfenster „Process Options“

Sie legen in diesem Dialogfenster Compiler-Optionen für das aktive Quelltextfenster fest, d.h. Sie bestimmen Eigenschaften des Prozesses, der aus dem aktiven Quelltext übersetzt und zur ADwin-Hardware übertragen wird.

Dies gilt sinngemäß auch für Library-Dateien, bei denen aber nur die Option *Optimize* eingestellt werden kann.

Sie müssen für jedes Quelltextfenster separat die nötigen Einstellungen vornehmen, indem Sie das Dialogfenster jeweils neu aufrufen (es sei denn, Sie möchten die Voreinstellung verwenden). Sie können das Dialogfenster mit einem Doppelklick auf die Statuszeile im Quelltextfenster öffnen.

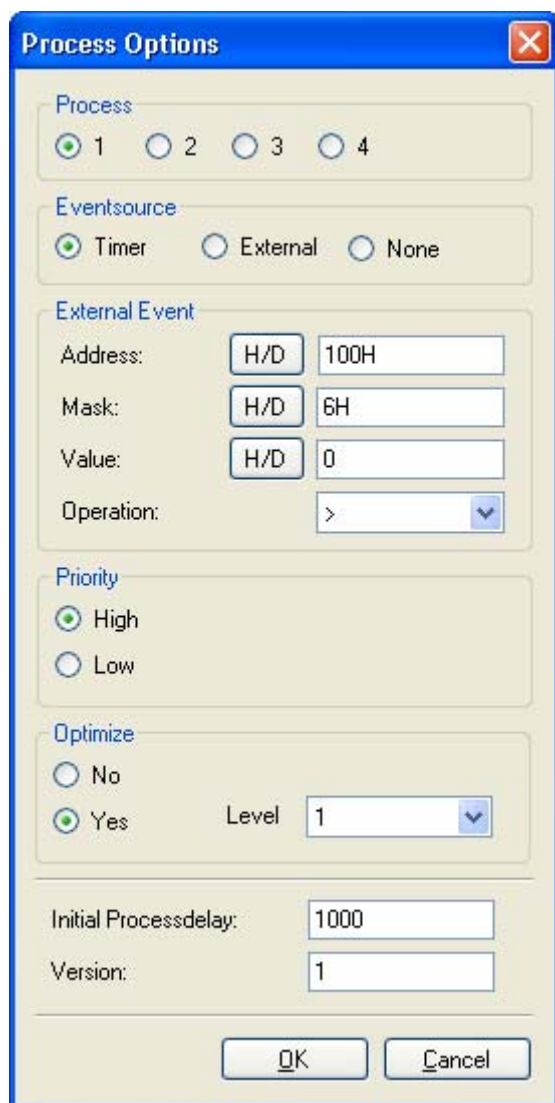


Abb. 4 – Das Dialogfenster Process Options

- **Process:** Prozessnummer

Stellen Sie die Nummer ein, unter der der übertragene Prozess auf dem *TiCo*-Prozessor angesprochen wird.

Wenn Sie mehrere Prozesse gleichzeitig auf einem *TiCo*-Prozessor ablaufen lassen, müssen Sie jedem Prozess eine eigene Nummer zuweisen.

- **Eventsource:** Stellen Sie ein, welches Event-Signal den Abschnitt **Event:** Ihres Prozesses starten soll.

- **Timer**
regelmäßige Impulse des internen Zählers dienen als Event-Signal. Mit der Systemvariablen **Processdelay** legen Sie fest, in welchen Zeitabständen der Zähler ein Event-Signal auslöst.
- **External**
Bestimmte Werte in der Hardware-Adresse **Address** im *TiCo*-Prozessor dienen als Event-Signal. Der Prozess läuft immer mit hoher Priorität ab.
Weitere Einstellungen siehe unten bei **External Event**.
- **None**
Der Prozessstyp **None** (ohne Event-Signal) wird nur für – meist in Assembler programmierte – Spezialanwendungen benötigt und schließt andere Prozessstypen aus. Wenn nicht anders programmiert, reagiert der Prozess nicht auf Event-Signale und wird nur einmal durchlaufen.

- **External Event:** Einstellungen für externe Event-Signale.

Hier legen Sie fest, welches Hardware-Ereignis unter welchen Bedingungen ein Event-Signal auslöst. Die Werte können jeweils hexadezimal oder dezimal eingegeben werden.

Der Ablauf zum Auslösen eines Event-Signals ist wie folgt: Der Wert an der Hardware-Adresse **Address** wird mit der Bitmaske **Mask** Und-verknüpft und anschließend mit dem Wert **Value** über den Operator **Operation** verglichen. Wenn der Vergleich wahr ist, wird ein Event-Signal ausgelöst.

Hardware-Adresse, Bitmaske und Vergleichswert sind für jede *ADwin*-Hardware unterschiedlich. Beachten Sie dazu die Beispielprogramme unter `C:\ADwin\TiCoBasic\`, bei denen das Wort **Extern_** im Dateinamen enthalten ist.

Beispiel: Die Einstellung oben maskiert den Wert der Adresse 70h mit 12h, so dass nur die Bits 2 und 5 erhalten bleiben. Wenn das Ergebnis > 0 ist, wenn also eines der beiden Bits gesetzt ist, wird der Prozess über ein Event-Signal gestartet.

- **Priority:** Priorität des Prozesses.

Stellen Sie die Priorität des Prozesses ein, mit der er im System laufen soll.

- **Optimize:**

Die wahlweise einschaltbare Optimierung kann die Prozess-Ausführungszeit (je nach Programmierung) um bis zu 20% verkürzen sowie den benötigten Speicherplatz verringern. Eine höherer Wert unter `Level` führt zu kürzeren Ausführungszeiten.

Wenn Sie unerwartete Compiler- oder Laufzeitfehler eines Prozesses feststellen, können Sie dies in Ausnahmefällen durch die erneute Einstellung eines niedrigeren `Level` beheben.

- **Initial Processdelay:** Stellen Sie hier das Processdelay (Zykluszeit) ein, mit dem der Prozess beginnen soll.

Alternativ können Sie die Variable `Processdelay` im Quelltext setzen.

- **User Version:** Ganzzahliger Wert, der als Versionsnummer des *TiCoBasic*-Programms dient.

Die Versionsnummer wird in der erzeugten Binärdatei abgelegt. Sie können die Versionsnummer direkt aus der Binärdatei auslesen; wenden Sie sich dazu bitte an unseren Support (support@adwin.de).

Dialogfenster „Settings“

Das Dialogfenster hat mehrere Seiten, die Sie über das Baumdiagramm im linken Teilfenster anwählen:

- Editor
 - [Editor - General](#)
 - [Editor - Syntax Colors](#)
 - [Editor - Print Settings](#)
- [Language](#)
- [Directories](#)
- [Help](#)
- [ADtools](#)

Editor - General

Parse and format: Der Editor kann den Quelltext automatisch formatieren, z.B. einrücken und die Syntax hervorheben. Hierfür ist es nötig, dass der Editor alle Quelltexte kontinuierlich durchsucht (engl.: to parse). Die gefundenen Informationen dienen auch als Basis für komfortable Funktionen wie: [Befehle und Variablen automatisch vervollständigen](#), [Deklarationen einer Datei anzeigen](#) oder [Befehlsparameter anzeigen](#).



Beachten Sie, dass das ständige Durchsuchen der Quelltexte auf langsamen Rechnern zu Geschwindigkeitseinbußen führen kann.

Parse Declarations: Der Editor durchsucht die Quelltexte kontinuierlich. Davon abhängig ist eine Reihe komfortabler Funktionen.

Autoindent: Der Quelltext wird automatisch eingerückt (engl.: to indent). Die Einrückpositionen werden über `Tabsize` festgelegt. Siehe auch [„Textzeilen einrücken“](#) auf [Seite 26](#).

Indent TiCoBasic sections: Programmabschnitte werden zusätzlich um eine Stufe eingerückt.

Smart format: Zeilen automatisch formatieren, siehe [„Automatisch formatieren“](#) auf [Seite 26](#).

Align comments at specified position: Kommentar hinter Quellcode wird automatisch auf die angegebene `Position` gesetzt.

Beachten Sie: Mit doppelten Kommentarzeichen ' ' können Sie einen Kommentar dennoch manuell positionieren.

Display quick info on mouse over: Wenn Sie den Mauszeiger auf ein Wort setzen, erscheinen zugehörige Deklaration und Hinweise als Tooltip. Siehe auch „[Deklaration eines Befehls oder Variablen anzeigen](#)“ auf [Seite 47](#).

Tabsize: Geben Sie ein, um wieviele Leerzeichen ein einzelner Tabulatorsprung eine Zeile einrückt. Zum Einrücken werden grundsätzlich nur Leerzeichen verwendet.

Font size: Geben Sie ein, mit welcher Zeichengröße der Quelltext angezeigt wird, siehe auch [Editor - Syntax Colors](#).

Beachten Sie: Andere Texte in der Bedienoberfläche bleiben unverändert.

“#If Processor = ...” graying: Ein bedingter Anweisungsblock wird grau dargestellt, wenn die aktuelle Prozesseureinstellung nicht der Bedingung mit dem Systemparameter `Processor` entspricht; siehe auch [#If ... Then ... {#Else ... } #EndIf](#) auf [Seite 173](#). Andere Bedingungen werden nicht berücksichtigt.

Show line numbers: Im Randstreifen (engl.: gutter) links vom Quelltext werden die Zeilennummern des Quelltexts angezeigt. Siehe auch „[Zu einer Programmzeile springen](#)“ auf [Seite 42](#).

Column mark, visible: An der angegebenen Position wird eine Linie angezeigt, so dass man eine selbst gewählte Zeilenlänge leicht einhalten kann. Auf diese Weise kann man z.B. zu lange Zeilen für den Druck vermeiden.

Editor - Syntax Colors

Der Editor hebt die verschiedenen Syntax-Elemente farbig hervor; siehe auch [Kapitel 3.4.1 „Syntax hervorheben“](#) auf [Seite 25](#). Die vollständige Syntax-Hervorhebung erfordert, dass die Option `Parse Declarations` unter [Editor - General](#) aktiv ist.

Sie können die Hervorhebung für jedes Syntaxelement (Definition siehe unten) separat einstellen:

- Color: Textfarbe
- Bold: Schriftschnitt **Fett**
- Italic: Schriftschnitt *Kursiv*

Der Beispieltext oberhalb wird mit den aktuellen Einstellungen formatiert.

Set to Default löscht alle Änderungen und setzt wieder die Standard-Einstellungen ein.

Sie können die Schriftgröße des Quelltexts unter [Editor - General](#), Font size ändern.

Der Editor unterscheidet folgende Syntax-Elemente:

- **TiCoBasic-Syntax** (System related):
 - **TiCoBasic sections:** Die Kennwörter **Init:**, **Event:** und **Finish:** für die Programmabschnitte.
 - **Compiler Directives:** Befehle für den Pre-Compiler wie **#Define**, die mit **#** beginnen.
 - **Reserved Keywords:** Die Basis-Befehle in *TiCoBasic* wie **Dim**.
 - **Global Variables:** Die globalen Variablen **Par_1 ... Par_80** und **Data_1 ... DATA_16**.
 - **External Keywords:** *TiCoBasic*-Befehle für den Zugriff auf Ein- und Ausgänge wie **P2_ADC**. Diese Befehle werden in der Regel in mitgelieferten Include- oder Library-Dateien deklariert.
 - **Symbols:** Rechenzeichen wie Klammern, + oder =.
- **Benutzerdefiniert** (User related):
 - **Defined Names:** Symbolische Namen wie **myName**, die mit **#Define** deklariert sind.
 - **Local Variables:** Mit **Dim** deklarierte lokale Variablen wie **myVar**.
 - **Sub Names:** Namen (wie **mySub**) von benutzerdefinierten Modulen, die mit **Sub** oder **Lib_Sub** deklariert sind.
 - **Function Names:** Namen (wie **myFunction**) von benutzerdefinierten Modulen, die mit **Function** oder **Lib_Function** deklariert sind.
- **Sonstige** (Other):
 - **Numbers:** Zahlenkonstanten in dezimaler (**15**), hexadezimaler (**0Fh**) und binärer Schreibweise (**1111b**).
 - **Strings:** Zeichenketten in doppelten **"Hochkommas"**.
- **Comments:** Kommentare nach *Rem* oder nach Kommentarzeichen **'**.
- **Standard Text:** Alle Elemente, die nicht zu anderen Gruppen gehören, z. B. ungültige Befehle wie **Eixt** (anstelle von **Exit**).

Editor - Print Settings

Die Einstellungen betreffen den Ausdruck eines Quelltexts.

Header bezieht sich auf die Kopfzeile des Ausdrucks.

Print Header: Bei aktiver Option erscheint auf jeder Seite des Ausdrucks eine Kopfzeile.

Header text: Der Text der Kopfzeile.

Layout legt fest, welche Elemente der Bildschirmansicht in den Ausdruck übernommen werden.

Color: Bei inaktiver Option ist der Ausdruck schwarz/weiß.

Syntax Highlight: Bei aktiver Option erscheint die Syntax-Hervorhebung im Ausdruck.

Line numbers: Bei aktiver Option werden die Zeilennummern am linken Rand gedruckt.

Font size: Legt die Schriftgröße des Ausdrucks fest.

Language

Wählen Sie aus, in welcher Landessprache die Fehlermeldungen des Compilers und die Online-Hilfe ausgegeben werden. Zur Wahl stehen Deutsch oder English.

Directories

Legen Sie die Verzeichnisse fest, in denen die Entwicklungsoberfläche und der Compiler nach Dateien suchen:

- **BTLDirectory:** Hier sucht die Entwicklungsumgebung nach Treiberdateien für die Kommunikation mit dem *TiCo*-Prozessor..
- **IncludeDirectory:** In diesem Verzeichnis sucht der Compiler nach Dateien `<*.Inc>`, die Sie mit **#Include** (und ohne Pfadangabe) in einen Quelltext einbinden.
- **LibDirectory:** In diesem Verzeichnis sucht der Compiler nach Library-Dateien `<*.lib>`, die Sie mit **Import** (und ohne Pfadangabe) in einen Quelltext einbinden.
- **Default working directory:** In diesem Verzeichnis sucht die Entwicklungsumgebung, wenn eine Datei oder ein Projekt geöffnet wird.

Wir empfehlen, die voreingestellten Verzeichnisse für BTL, Include und Library nicht zu verändern. Wenn Sie Include- und Library-Dateien aus anderen Verzeichnissen einbinden, können Sie im Einbinde-Befehl den vollständigen oder relativen Pfadnamen angeben.

Help

Mit der Option `stay on top` bleibt das Hilfefenster in jedem Fall das oberste Fenster. Dadurch kann die Bedienoberfläche verdeckt werden.

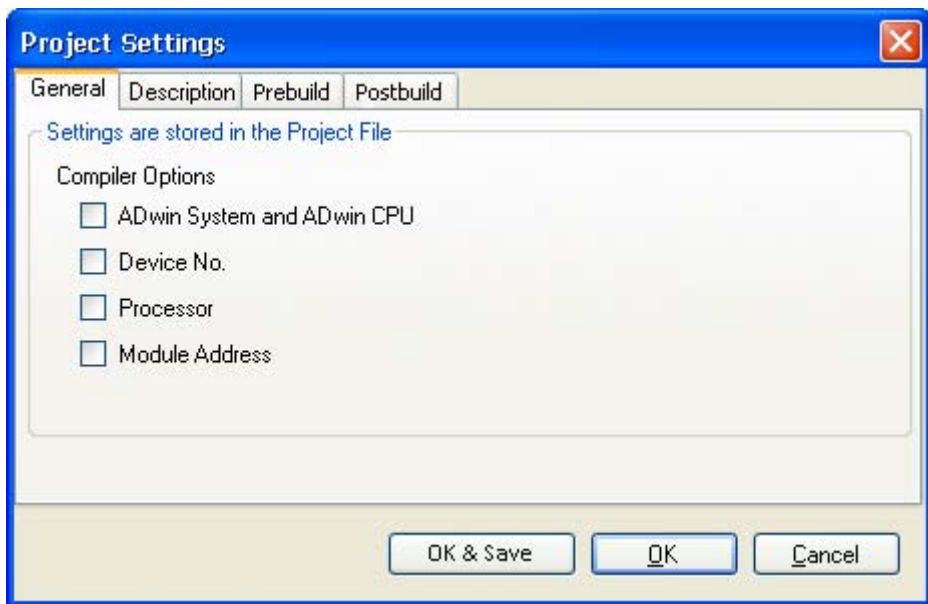
ADtools

Die Hilfsprogramme [ADtools](#) (Beschreibung siehe [Kapitel 3.12 auf Seite 89](#)) können über eine eigene Symbolleiste aufgerufen werden. Bei aktiver Option erscheint das jeweilige Hilfsprogramm in der *ADtools*-Leiste.

Dialogfenster „Project Settings“

Sie legen im Dialogfenster Einstellungen zum aktuellen Projekt fest. Die Einstellungen werden mit der Projektdatei gespeichert. Eine gespeicherte Einstellung wird beim Laden der Projektdatei wieder hergestellt; vorherige Einstellungen gehen verloren.

- [Reiter General](#)
- [Reiter Description](#)
- [Reiter Prebuild / Postbuild](#)

**Reiter** General

Die markierten Einstellungen werden mit der Projektdatei gespeichert.

- Dialogfenster Compiler Options
 - ADwin System and ADwin CPU
 - Device No.
 - Processor
 - Module Address
- Menü Debug
 - Debug Mode

Reiter Description

Sie können eine Beschreibung erstellen und mit der Projektdatei speichern. Die Beschreibung hat keine weitere Funktion in .

Reiter Prebuild / Postbuild

Sie können Kommandozeilen-Aufrufe eingeben, die automatisch vor jedem Übersetzen (Prebuild) bzw. danach (Postbuild) ausgeführt

werden. Die Kommandozeilen werden mit der Projektdatei gespeichert.

Jeder Compiler-Aufruf (siehe unten, `$BUILDACTION`) löst die Ausführung der Kommandozeilen-Aufrufe aus; jedoch nur, wenn die zu kompilierende Quelltext-Datei in die Projektdatei eingebunden ist.

Die Kommandozeilen werden nacheinander dem Betriebssystem übergeben, wobei das Projektverzeichnis als Arbeitsverzeichnis dient. Anschließend wartet jeweils auf die erfolgreiche Abarbeitung der Kommandozeile. Wenn die Abarbeitung fehlschlägt und nicht mehr reagiert, müssen Sie die Abarbeitung über den Taskmanager beenden.

In einer Kommandozeile können folgende Variablen verwendet werden:

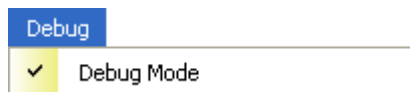
- `$PROJECTNAME`: Projektdatei ohne Pfad und Dateierweiterung.
- `$PROJECTPATH`: Pfad des Projektverzeichnisses (ohne Projektdatei).
- `$PROJECT`: Projektdatei (ohne Pfad).
- `$BUILDACTION`: Zeichenkette, die den auslösenden Compiler-Aufruf angibt.
 - C: Menüeintrag Build ► Compile
 - MB: Menüeintrag Build ► Make Bin File
 - ML: Menüeintrag Build ► Make Lib File

Pfadangaben sollten zur Sicherheit mit Anführungszeichen umgeben werden (wegen Leerzeichen).

Wenn eine Zeile mit `#` beginnt, wird sie als Kommentar gewertet und nicht weiter beachtet.

3.9.6 Das Menü „Debug“

Im Menü `Debug` können Sie den `Debug-Modus` aktivieren, der Ihnen beim Auffinden von Laufzeit-Fehlern hilft. Beachten Sie bitte, dass die Einstellung erst nach dem nächsten Kompilieren wirksam wird.



Option Debug mode

Wenn Sie die Compiler-Option **Debug mode** im Menü **Debug** aktivieren und anschließend einen Quelltext kompilieren, werden zusätzliche Sicherheitsabfragen in den Prozess eingebaut (siehe auch [Kapitel 5.3.1 auf Seite 124](#)).

Die Compiler-Einstellung wird auch in der **Statusleiste** angezeigt, die Einstellung für einen laufenden Prozess dagegen im **Prozessfenster**.

Das Einschalten der Option verlängert die Programm-Ausführungszeit und vergrößert den Speicherbedarf. Sie sollten daher die Option nur während der Programmentwicklung nutzen.

Mit **#Begin_/#End_Debug_Mode_Disable** können Sie den Modus zeitweise deaktivieren.

Wenn ein Laufzeitfehler im **ADwin**-System auftritt, wird die Fehlermeldung im Fenster „**Debug Errors**“ angezeigt (unten im **Info-Bereich**), und zwar mit dem Zeitpunkt des ersten Auftretens. Beim ersten Auftreten eines Fehlers wird das Fenster automatisch in den Vordergrund gesetzt.

Ein Neustarten des Prozesses setzt die Fehlermeldungen nicht zurück, das Rücksetzen geschieht jedoch bei einem Neuladen des Prozesses.

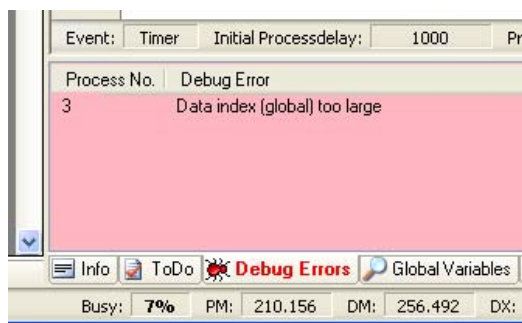


Abb. 5 – Das Fenster **Debug Errors**

Die folgende Tabelle zeigt, welche Fehler angezeigt werden und welche Korrektur dazu jeweils erfolgt.

Fehlermeldung, Ursache	Korrekturmaßnahme
Data index zu groß / <1 Array index zu groß / <1 Schreibzugriff auf nicht deklarierte Elemente eines lokalen oder globalen Felds, d.h. auf eine zu große oder zu kleine Elementnummer.	Der Zugriff wird nicht ausgeführt.

Es wird für jeden Prozess nur ein einzelner Fehler angezeigt (in der Regel der zuletzt aufgetretene), auch wenn der Prozess mehrere Laufzeitfehler erzeugt.

3.9.7 Das Menü „Tools“

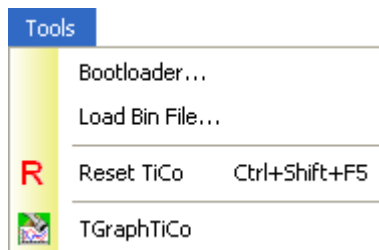
Im Menü `Tools` rufen Sie Hilfsprogramme auf.

Der Menüeintrag `Bootloader...` überträgt eine Binärdatei zum *TiCo*-Prozessor und/oder aktiviert den *TiCo*-Bootloader. Der *TiCo*-Bootloader kann einen Prozess beim Einschalten der *ADwin*-Hardware automatisch starten. Näheres ist in [Kapitel 3.7.2 „TiCo-Bootloader programmieren“](#), [Seite 50](#) beschrieben.

Der Menüeintrag `Load Bin File...` überträgt eine vorhandene Binärdatei zum *TiCo*-Prozessor (siehe [TiCo-Binärdatei übertragen](#), [Seite 49](#)).

Der Menüeintrag `Reset TiCo` stoppt den *TiCo*-Prozessor und setzt ihn zurück; alle globalen *TiCo*-Variablen werden auf Null gesetzt und der Prozess wird gelöscht.

Die Menüeinträge `Bootloader...` und `Load Bin File...` stehen nur zur Verfügung, wenn im [Dialogfenster „Compiler Options“](#) die Option „Do not access the device“ deaktiviert ist (siehe [Seite 58](#)).



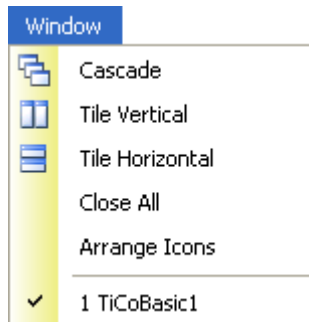
Der Menüeintrag `TGraphTiCo` startet ein Hilfsprogramm; Kurzbeschreibung siehe [Kapitel 3.12 auf Seite 89](#).

3.9.8 Das Menü „Window“

Mit dem Menü `Window` können Sie zwischen verschiedenen Quelltext-Fenstern umschalten und diese am Bildschirm arrangieren.

Der Menüpunkt `Arrange Icons` ordnet die Symbole verkleinerter Dateien neu an, was Sie z. B. nach einer Änderung der Bildschirmauflösung verwenden können.

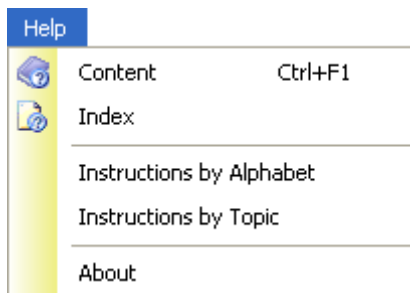
Unten im Menü können Sie über die Dateinamen einen der geöffneten Quelltexte zum aktiven Fenster machen. Der aktive Quelltext ist mit einem Haken gekennzeichnet; im Beispiel rechts ist dies `TiCoBasic1.bas`.




3.9.9 Das Menü „Help“

Mit dem Menü `Help` rufen Sie die Online-Hilfe der Entwicklungsumgebung auf:

- Content: Inhaltsverzeichnis
- Index: Indexverzeichnis
- Instructions by
 - Alphabet: Befehlsliste alphabetisch
 - Topic: Befehlsliste nach Themen.



Die Befehlslisten beziehen sich auf das *ADwin*-System, das im [Dialogfenster „Compiler Options“ \(Seite 58\)](#) eingestellt ist.

Alternativ können Sie auch die Schaltfläche  verwenden. Mit der Taste [F1] erhalten Sie Hilfe zu einem geöffneten Fenster oder zu einem markierten Befehlswort.

About öffnet ein Fenster, das die Version der Entwicklungsumgebung und den verwendeten License key angibt. Wenn Sie die Schaltfläche Change License wählen, können Sie den License key ändern (siehe auch [Kapitel 3.1.1 auf Seite 11](#)).

Ohne Eingabe des gültigen License key befindet sich *TiCoBasic* im Demo-Modus. In diesem Modus ist das Arbeiten mit der Entwicklungsumgebung nur zu Prüf-, Demonstrations- und Bewertungszwecken erlaubt. Sie können beispielsweise keine Binärdateien erzeugen.

3.10 Fenster

Der [Info-Bereich](#) wird separat behandelt, siehe [Seite 83](#).

3.10.1 Toolbox

Die Toolbox ist der Bereich in der Oberfläche links, in dem das [Projektfenster](#), das [Parameterfenster](#), das [Registerfenster](#) und das [Prozessfenster](#) angezeigt werden.

Die Toolbox teilt sich in einen oberen und einen unteren Anzeigebereich. Die zugehörigen Fenster können den beiden Anzeigebereichen frei zugeordnet werden. Ein verdecktes Fenster wird durch einen Klick auf den zugehörigen Reiter in den Vordergrund geholt.

Um ein Fenster einem Bereich zuzuordnen, gehen Sie vor wie folgt:

- Öffnen Sie mit einem Rechtsklick auf die Kopfleiste des Fensters das Kontextmenü.
- Wählen Sie den Anzeigebereich oben (top) oder unten (bottom).



- Es ist möglich, alle Fenster dem gleichen Anzeigebereich zuzuordnen. Dadurch ist nur eines der Fenster im Vordergrund.

Die Standardeinstellung kann durch den Menüeintrag **View ▶ Restore default layout** wieder eingestellt werden.

Die Toolbox kann über die Symbole in der Kopfzeile als frei verschiebbares Fenster angezeigt oder ganz ausgeblendet werden.

3.10.2 Projektfenster

Das Projektfenster zeigt an, ob ein Projekt geöffnet und welche Quelltext- oder Include-Dateien eingebunden sind.

Das Projektfenster ist Teil der [Toolbox](#) (siehe [Seite 76](#)).

Sie können im Projektfenster folgende Aktionen ausführen (siehe auch [Projekte verwalten](#), [Seite 51](#)):

- Alle offenen Dateien in das Projekt einbinden:
Wählen Sie `Add Open Files to Project` aus dem Kontextmenü im Projektfenster.
- Eine andere Datei zum Projekt hinzufügen:
Wählen Sie `Add Other File to Project` aus dem Kontextmenü im Quelltextfenster.
- Eine oder mehrere Dateien aus dem Projekt löschen:
Markieren Sie die Dateien per Mausklick im Projektfenster und
 - drücken die Taste `[ENTF]` oder
 - wählen `Remove from Project` aus dem Kontextmenü.
- Eine Datei öffnen und zum aktiven Quelltext machen:
 - Klicken Sie doppelt (linke Maustaste) auf die Datei oder
 - Markieren Sie eine Datei im Projektfenster und wählen `Open` aus dem Kontextmenü.
- Alle Dateien des Projekts speichern:
Wählen Sie `Save all Files of Project` aus dem Kontextmenü im Projektfenster oder die Schaltfläche in der Projektleiste.
- Windows Explorer mit dem Pfad der markierten Datei öffnen:
Wählen Sie `Open Path in Window Explorer` aus dem Kontextmenü im Projektfenster.

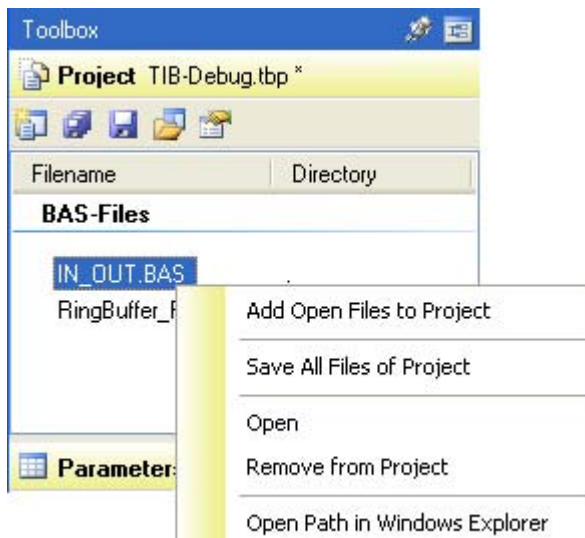


Abb. 6 – Das Projekt-Fenster mit Kontextmenü

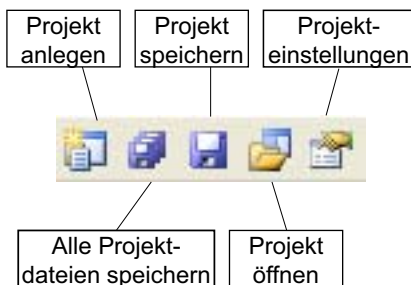



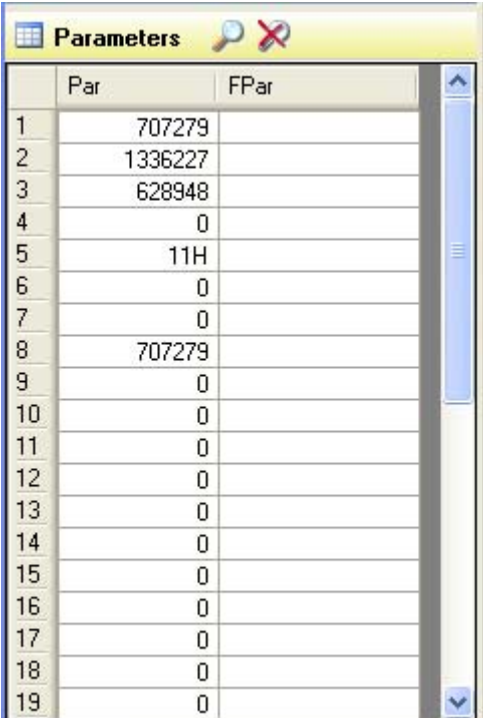
Abb. 7 – Die Projektleiste

3.10.3 Parameterfenster

Das Parameterfenster zeigt die globalen Parameter **Par_1...Par_80** an. Mit dem Schieberegler am rechten Rand können Sie den angezeigten Parameter-Bereich auswählen.

Das Parameterfenster ist Teil der [Toolbox](#) (siehe [Seite 76](#)).

Wenn die Kommunikation zwischen PC und ADwin-System aktiv ist (Schaltfläche **Enable Cyclic Update**  in der Werkzeugleiste), sind die Tabellenfelder weiß hinterlegt und zeigen die Werte der globalen Parameter an. Die Werte werden kontinuierlich vom System ausgelesen und angezeigt. Grau hinterlegte Felder zeigen an, dass keine Kommunikation stattfindet.




	Par	FPar
1	707279	
2	1336227	
3	628948	
4	0	
5	11H	
6	0	
7	0	
8	707279	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	
16	0	
17	0	
18	0	
19	0	

Parameters Processes Registers


Abb. 8 – Das Parameter-Fenster

Sie können die Werteanzeige zwischen dezimal und hexadezimal umstellen (siehe **Par_5** in [Abb. 8](#)), indem Sie mit der Maus auf die Nummer der betreffenden Variable klicken (links vom Tabellenfeld). Mit einem Klick auf den Spaltentitel **Par** wird die Anzeige aller Parameter **Par_1...Par_80** auf einmal geändert.

Die Fließkommawerte `FPar_1` ... `FPar_80` werden mit 7 Stellen Genauigkeit angezeigt.

Mit der Schaltfläche `Scan Global Variables`  können Sie [Verwendete globale Variablen und Felder anzeigen](#) (siehe [Seite 48](#)).

3.10.4 Prozessfenster

Das Prozessfenster zeigt Informationen über die Prozesse auf dem *TiCo*-Prozessor an, wenn die Kommunikation zwischen PC und System aktiv ist (Schaltfläche  in der Werkzeugleiste). Anderenfalls ist das Fenster grau hinterlegt.

Das Prozessfenster ist Teil der [Toolbox](#) (siehe [Seite 76](#)). Sie öffnen das Prozessfenster mit einem Klick auf den Reiter `Processes`.

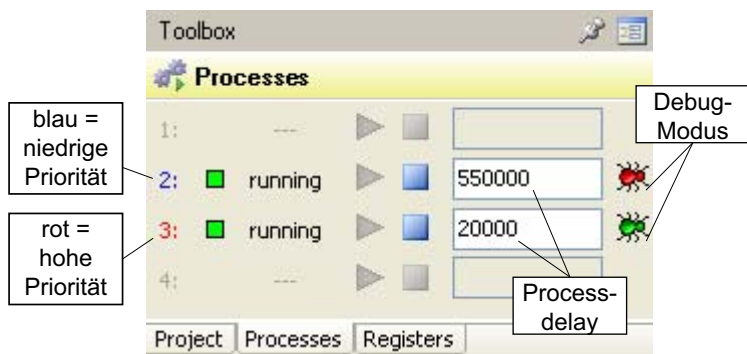


Abb. 9 – Das Prozess-Fenster

Für jeden Prozess werden folgende Informationen angezeigt:



– Prozess-Priorität

Die Farbe der Prozessnummer gibt die Priorität an.

- rot = hohe Priorität
- blau = niedrige Priorität.

Die Bedeutung und die Zeiteinheiten des Processdelay sind in [Kapitel 6.2.1](#), [Seite 131](#) erläutert.


- Prozess-Status
 - `running`: Prozess läuft.
 - `stopped`: Prozess wurde angehalten.
 - `---`: Prozess ist nicht vorhanden.

Sie können einen Prozess mit der Schaltfläche `Stop`  beenden und mit `Start`  wieder starten. Die Schaltflächen in der Werkzeugleiste haben die gleiche Funktion; sie beziehen sich auf den zum aktiven Quelltext gehörigen Prozess.

- Processdelay (Prozess-Zykluszeit)

Das zum aktiven Quelltext gehörige Processdelay ist auch oben in der Werkzeugleiste zu sehen.


Sie können die Zykluszeit ändern, indem Sie einen neuen Wert in das Eingabefeld schreiben. Sobald Sie das Feld verlassen, wird der Wert zum *TiCo*-Prozessor übertragen. Achten Sie darauf, den *TiCo*-Prozessor nicht durch zu kleine Werte überlasten.

- Prozess arbeitet im Debug-Modus 

Das Icon wird angezeigt, wenn der Prozess im Debug-Modus arbeitet. Näheres zum Debug-Modus ist unter [Option Debug mode](#) beschrieben. Das Icon wird rot, wenn im Prozess ein Debug-Fehler erkannt wurde.

Die Compiler-Einstellung zum Debug-Modus wird in der [Statusleiste](#) angezeigt.

3.10.5 Registerfenster

Das Registerfenster zeigt die Registerinhalte des *TiCo*-Prozessors an, wenn die Kommunikation zwischen PC und System aktiv ist (Schaltfläche  in der Werkzeugleiste). Anderenfalls sind die Felder grau hinterlegt.

Das Registerfenster ist Teil der [Toolbox](#) (siehe [Seite 76](#)). Sie öffnen das Registerfenster mit einem Klick auf den Reiter `Registers`.

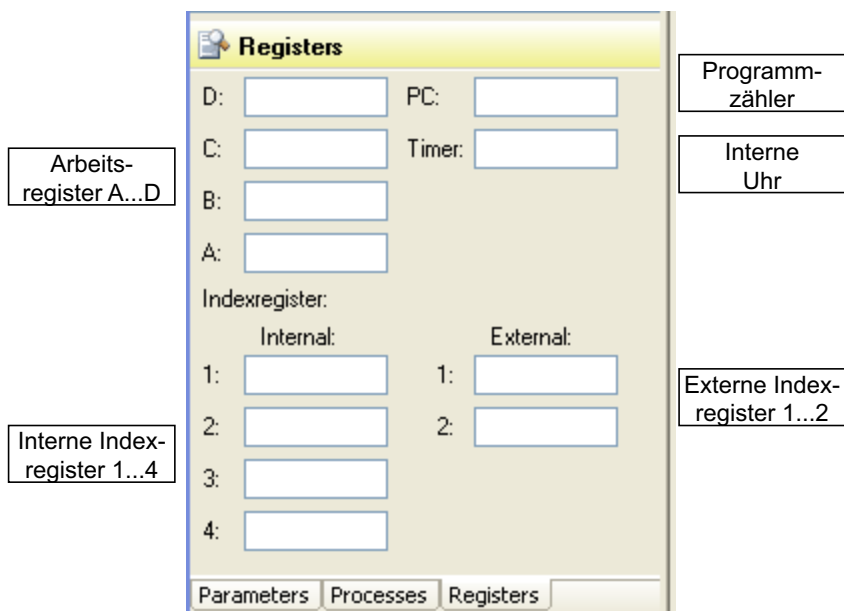


Abb. 10 – Das Register-Fenster

Die Registerinhalte sind hilfreich, wenn Sie in Ihrem Programm Assembler-Befehle verwenden. Eine Dokumentation der Assembler-Befehle ist derzeit noch nicht verfügbar.

Das Quelltextfenster ist das Hauptfenster der Entwicklungsumgebung, in dem Sie die Quelltexte von *TiCoBasic*-Programmen editieren.

Die verschiedenen Funktionen im Quelltextfenster sind in [Kapitel 3](#) beschrieben. Wichtige Abschnitte sind folgende:

- [Grundelemente der Entwicklungsumgebung](#), Seite 15.
- [Quelltexte erstellen](#), Seite 19.
- [Kontextmenü im Quelltextfenster](#), Seite 21.

3.10.6 Statusleiste

Die Statusleiste befindet sich am unteren Rand der Bedienoberfläche.



- Aktion in
 - Prozessorauslastung und Speichergröße
 - Cursor-Position
Compiler-Einstellung
- Links: Informationen zur zuletzt ausgeführten Aktion.
 - Mitte: Die aktuelle Auslastung (bei aktiver Verbindung zwischen PC und *TiCo*) und die Speichergröße des *TiCo*-Prozessors. aktiver
 - Rechts: Die aktuelle Cursor-Position im Quelltextfenster (Zeile und Spalte); daneben die Einstellungen für den Compiler: Debug-Modus, Device no., Prozessor, *ADwin*-Hardware.

Ein Doppelklick auf die Compiler-Einstellungen öffnet das [Dialogfenster „Compiler Options“](#) ([Seite 58](#)).

Die Angaben zu Prozessorauslastung und Speicherplatz bedeuten:

- Busy:** Zeitliche Auslastung des Prozessors in Prozent, berechnet als:
 $\text{Rechenzeit} / (\text{Rechenzeit} + \text{Leerlaufzeit})$.
Siehe auch [Kapitel 6.2.2](#).
- PM:** Verfügbarer Programmspeicher in Bytes.
- DM:** Verfügbarer interner Datenspeicher in Bytes.
- DX / SX:** Verfügbarer externer Datenspeicher in Bytes.

3.11 Info-Bereich

Der Info-Bereich ist der Bereich in der Oberfläche ganz unten, in dem folgende Fenster angezeigt werden:

- [Infofenster](#) (siehe unten)
- [ToDo-Liste](#) (siehe [ToDo-Liste](#))
- [Das Fenster Debug Errors](#) (siehe [Seite 72](#))
- [Fenster „Global Variables“](#) (siehe [Seite 86](#))
- [Fenster „Declarations“](#) (siehe [Seite 87](#))

Weitere Fenster siehe [Kapitel 3.10](#), [Seite 76](#).

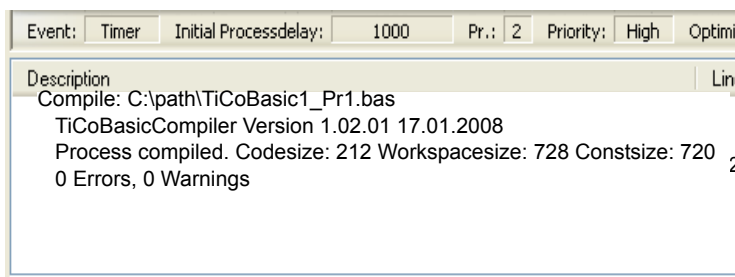
3.11.1 Infofenster

Im Info-Fenster werden Meldungen des Compilers zum jeweils aktiven Quelltext dargestellt:

- Statusmeldung nach dem Kompilieren
- Fehlermeldungen
- Warnungen

Warnungen und Fehlermeldungen werden mit dem Ort des Auftretens (Zeile, Dateiname und Pfad) angegeben. Ein Doppelklick auf die Meldung färbt die betreffende Quelltextzeile rot und setzt den Cursor in die Zeile.

Eine erfolgreiche Statusmeldung nach dem Kompilieren sieht z. B. folgendermaßen aus:



Nur für Prozessoren bis T11: Die Werte der Statusmeldung geben Hinweise, wieviel Speicherplatz der Prozess benötigen wird.

- **Codesize**: Größe der erzeugten Binärdatei in Bytes; die Datei wird als Prozess im Programmspeicher (PM) abgelegt.
- **Workspacesize**: Benötigter Speicherplatz in Bytes im lokalen Datenspeicher (DM).

Der Platz wird benötigt für:

- Compiler-Tabelle: 720 Byte
- interne Zwecke: 8 Byte
- temporäre Compiler-Variablen: Bedarf abhängig vom Programm
- Variablen und Felder in [DM_Local](#):
jede lokale Variable: 4 Byte

jedes lokale Feld: $(\text{Feldgröße} + 1) * 4 \text{ Byte}$

jedes globale Feld `Data_x`: $(\text{Feldgröße} + 6) * 4 \text{ Byte}$

- Auch Variablen und Felder in `DRAM_Extern` benötigen Platz im lokalen Datenspeicher:

jedes lokale Feld: 0 Byte

jedes globale Feld `Data_x`: 16 Byte

- `Constsize`: Benötigter Speicherplatz (DM) für Konstanten in Bytes.
- `Stacksize`: Größe des internen Stapels (DM), der für Libraries verwendet wird.

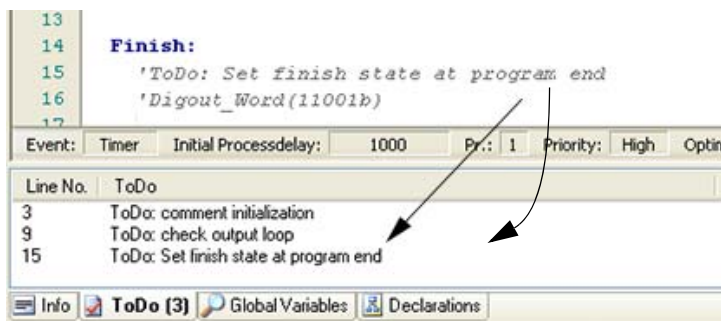
Es wird nicht angezeigt, wieviel Speicherplatz im externen Datenspeicher (DX) benötigt wird.

3.11.2 ToDo-Liste

Das Fenster `ToDo` dient als einfache ToDo-Liste: es werden Zeilen der aktuellen Quelltextdatei angezeigt, in denen der Text `ToDo`: als Kommentar enthalten ist. Durch entsprechende Kommentarzeilen können Sie noch nicht erledigte Arbeiten am Quelltext markieren und übersichtlich anzeigen.

Wenn eine Aufgabe erledigt ist, löschen Sie einfach die entsprechende Kommentarzeile.


Das Fenster `ToDo` ist Teil des [Info-Bereichs](#) (siehe [Seite 83](#)).



Ein Doppelklick auf einen `ToDo`-Eintrag setzt den Cursor in die betreffende Zeile im Quelltext.

3.11.3 Fenster „Global Variables“

Das Fenster Global Variables zeigt an, welche globalen Variablen (**Par_1 ... Par_80**) und Felder (**Data_1 ... Data_16**) in einem Projekt oder in einem Quelltext verwendet werden.

Um die Anzeige zu starten oder zu aktualisieren, klicken Sie auf die Schaltfläche **Scan Global Variables**  im **Parameterfenster** (siehe [Verwendete globale Variablen und Felder anzeigen](#), Seite 48).

Das Fenster ist Teil des **Info-Bereichs** (siehe Seite 83).


All		Global Variable	Process file	Line No.	Source file
Par	Par_1	Par_1	ADB-SCR-WIN-GlobaVars1.bas	4	adb-scr-win-globalvars.inc
	Par_1	Par_1	ADB-SCR-WIN-GlobaVars1.bas	9	
	Par_2	Par_1	ADB-SCR-WIN-GlobaVars1.bas	13	
	Par_3	Par_1	ADB-SCR-WIN-GlobaVars1.bas	15	used 2 times
	Par_4	Par_1	ADB-SCR-WIN-GlobaVars1.bas	16	used 2 times
	Par_10	Par_1	ADB-SCR-WIN-GlobaVars2.bas	8	
	FPar	Par_2	ADB-SCR-WIN-GlobaVars1.bas	14	
	Data	Par_3	ADB-SCR-WIN-GlobaVars2.bas	5	
	Data_5	Par_3	ADB-SCR-WIN-GlobaVars2.bas	7	
	Data_8	Par_4	ADB-SCR-WIN-GlobaVars1.bas	2	adb-scr-win-globalvars.inc
Data	Data_5	Par_4	ADB-SCR-WIN-GlobaVars1.bas	3	adb-scr-win-globalvars.inc
	Data_5	Par_4	ADB-SCR-WIN-GlobaVars2.bas	6	
	Data_5	Par_4	ADB-SCR-WIN-GlobaVars2.bas	7	
	Data_5	Par_10	ADB-SCR-WIN-GlobaVars1.bas	3	adb-scr-win-globalvars.inc
	Data_5	Par_10	ADB-SCR-WIN-GlobaVars2.bas	7	
	Data_5	Data_5	ADB-SCR-WIN-GlobaVars1.bas	5	
	Data_5	Data_5	ADB-SCR-WIN-GlobaVars1.bas	15	
	Data_5	Data_5	ADB-SCR-WIN-GlobaVars2.bas	2	
	Data_5	Data_8	ADB-SCR-WIN-GlobaVars1.bas	4	
	Data_5				

Das markierte Element in der Baumanzeige links legt fest, welche Variablen im Fenster rechts angezeigt werden. So zeigt die Liste nach einem Klick auf **Par** nur noch die Variablen **Par_1...Par_80** an (soweit verwendet).

Sie können die Zeilen im rechten Fenster mit einem Klick auf einen Spaltentitel alphabetisch sortieren.

Die Liste zeigt an:

- **Global Variable:** Name der verwendeten globalen Variablen oder des globalen Felds.
- **Process file:** Name der Hauptdatei des durchsuchten Prozesses.
- **Line no.:** Zeilennummer, wo die Variable genutzt oder aufgerufen wird.
- **Source file:** Dateiname, wo die Variable genutzt oder aufgerufen wird. Ein leerer Eintrag heißt, dass die Zeile sich auf die Hauptdatei des Prozesses bezieht (siehe Spalte **Process file**).

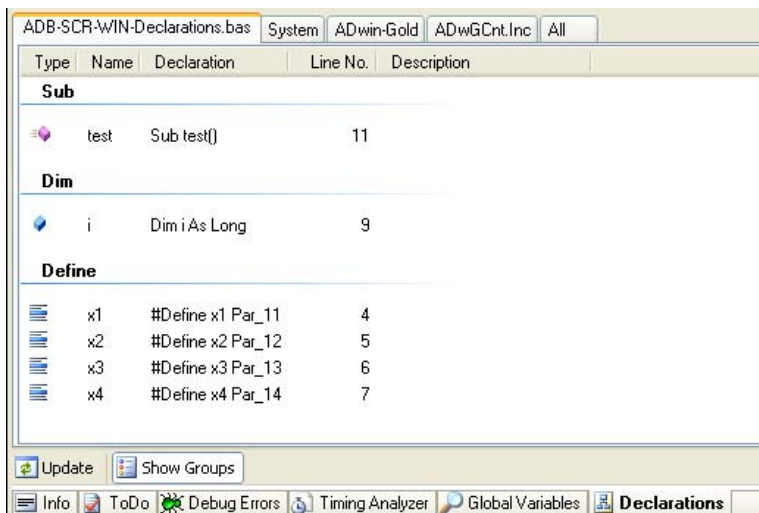
Wenn Sie den Quelltext ändern, wird das Fenster nicht automatisch aktualisiert. Verwenden Sie dazu Schaltfläche **Scan Global Variables**  im Parameterfenster.

3.11.4 Fenster „Declarations“

Das Fenster **Declarations** zeigt alle zu einer Quelltextdatei und zu den verbundenen Include- und Library-Dateien gehörenden Deklarationen an. Zum Aktualisieren der Anzeige drücken Sie die Schaltfläche **Update**.

Deklarationen aus anderen als der aktiven Quelltextdatei – auch innerhalb eines Projekts – werden nicht berücksichtigt.

Das Fenster **Declarations** ist Teil des [Info-Bereichs](#) (siehe [Seite 83](#)).



Die Deklarationen werden unter verschiedenen Reitern, die den Ort der Deklaration repräsentieren, angezeigt:

- [file].bas: In der Quelltextdatei erstellte Deklarationen: lokale Variablen, Felder, Befehle (**Sub**, **Function**) und symbolische Namen (**#Define**).
- System: Systemvariablen und Befehle, die in *TiCoBasic* implementiert sind und die zu den aktuellen Compiler-Einstellungen passen.

Die globalen Variablen `Par` werden nicht angezeigt. Beachten Sie hierzu auch das [Fenster „Global Variables“ \(Seite 86\)](#) und die Funktion [„Verwendete globale Variablen und Felder anzeigen“ \(Seite 48\)](#).

- `ADwin-Gold, ADwin-light-16`: Befehle für Hardware-Zugriffe, die in *TiCoBasic* implementiert sind und die zu den aktuellen Compiler-Einstellungen passen.
- `[file].inc`: Variablen und Befehle, die in dieser Include-Datei deklariert sind. Ein solcher Reiter taucht nur auf, wenn im Quelltext eine Include-Datei mit `#Include` eingebunden ist.
- `[file].lib`: Variablen und Befehle, die in dieser Bibliotheksdatei deklariert sind. Ein solcher Reiter taucht nur auf, wenn im Quelltext eine Bibliotheksdatei mit `Import` eingebunden ist.
- `All`: Alle gültigen Deklarationen aus den oben aufgeführten Quellen.

Sie können die Deklarationen mit einem Klick auf einen Spaltentitel sortieren. Wenn Sie die Option `Show Groups` aktivieren, werden die Deklarationen nach Gruppen sortiert.

Wenn Sie den Quelltext ändern, wird das Fenster nicht automatisch aktualisiert. Verwenden Sie dazu Schaltfläche `Update`.

Die Deklarationen können nur angezeigt werden, wenn die Option `Parse Declarations` unter [Editor - General](#) (siehe [Seite 64](#)) aktiv ist.

3.12 ADtools

ADtools sind kleine Hilfsprogramme, die Daten und Betriebszustände eines *ADwin*-Systems oder eines *TiCo*-Prozessors anzeigen. Sie starten *ADtools* mit einem Klick auf eine Schaltfläche in der senkrechten Leiste am rechten Rand.

Für den *TiCo*-Prozessor gibt es derzeit nur das Programm `TGraphTiCo`, mit dem Sie die Werte globaler Felder (`Data`) grafisch anzeigen können.

Bitte beachten Sie: Sie können in die -Leiste auch [Aufrufkürzel zu Dateien / Ordnern einfügen \(Seite 49\)](#).

Jedes *ADtool* ist ein eigenständiges Windows-Programm, das Sie auch mehrfach starten können: Lassen Sie sich alle interessanten Parameter auf dem Bildschirm anzeigen. Wenn Sie eine passende

Bildschirmanzeige zusammengestellt haben, können Sie die Gesamt-Konfiguration speichern und später erneut verwenden.

Folgende *ADtools* stehen Ihnen zur Verfügung:



ADtools

speichert und lädt eine von Ihnen erstellte Gesamt-Konfigurationen aus mehreren *ADtools*.



TGraphTiCo

stellt den Inhalt globaler Felder eines *TiCo*-Prozessors in Kurvenform dar.

Alle weiteren Informationen zu den Hilfsprogrammen entnehmen Sie bitte der Online-Hilfe, die Sie im jeweiligen Hilfsprogramm aufrufen.

4 Prozesse programmieren


In diesem Kapitel zeigen wir Ihnen, wie Sie ein *TiCoBasic*-Programm aufbauen, strukturieren und welche Variablen Ihnen dabei zur Verfügung stehen.

4.1 Programmaufbau

Sie geben ein *TiCoBasic*-Programm als ASCII-Text mit dem Editor der Entwicklungsumgebung ein; dabei verwenden Sie eine erweiterte Basic-Syntax. Diesen Quelltext übersetzt der Compiler in einen ausführbaren Prozess für den *Tico*-Prozessor, siehe [Quelltext übersetzen](#) auf [Seite 23](#).

Ein Quelltext besteht aus einer beliebigen Anzahl von Befehlszeilen, die jeweils einen Befehl oder eine Zuweisung enthalten; Ausnahme siehe : ([Doppelpunkt](#)). Eine Quelltextzeile darf bis zu 2048 (ASCII-) Zeichen enthalten; Ausnahme siehe [#Include](#).

TiCoBasic akzeptiert bei Befehlen und Variablennamen Groß- und Kleinschreibung. In unseren Beispielen verwenden wir allerdings zur besseren Unterscheidung eine einheitliche Schreibweise.

Ein Programm besteht aus bis zu 3 Abschnitten, die bei der Ausführung auf dem *Tico*-Prozessor unterschiedliche Aufgaben übernehmen, sowie aus den erforderlichen Deklarationen. Halten Sie die Reihenfolge der [Abb. 11](#) beim Programmaufbau ein. 

Jedes Programm muss zumindest den Abschnitt **Event:** enthalten.

Die Ausnahme im Programmaufbau ist der [Prozess ohne Ansteuerung \(None\)](#), in dem es keine fest definierten Abschnitte gibt. [Näheres siehe Seite 130](#).

Optional können Sie Funktionen und Unterprogramme definieren, und „Include“-Dateien und Libraries einbinden.

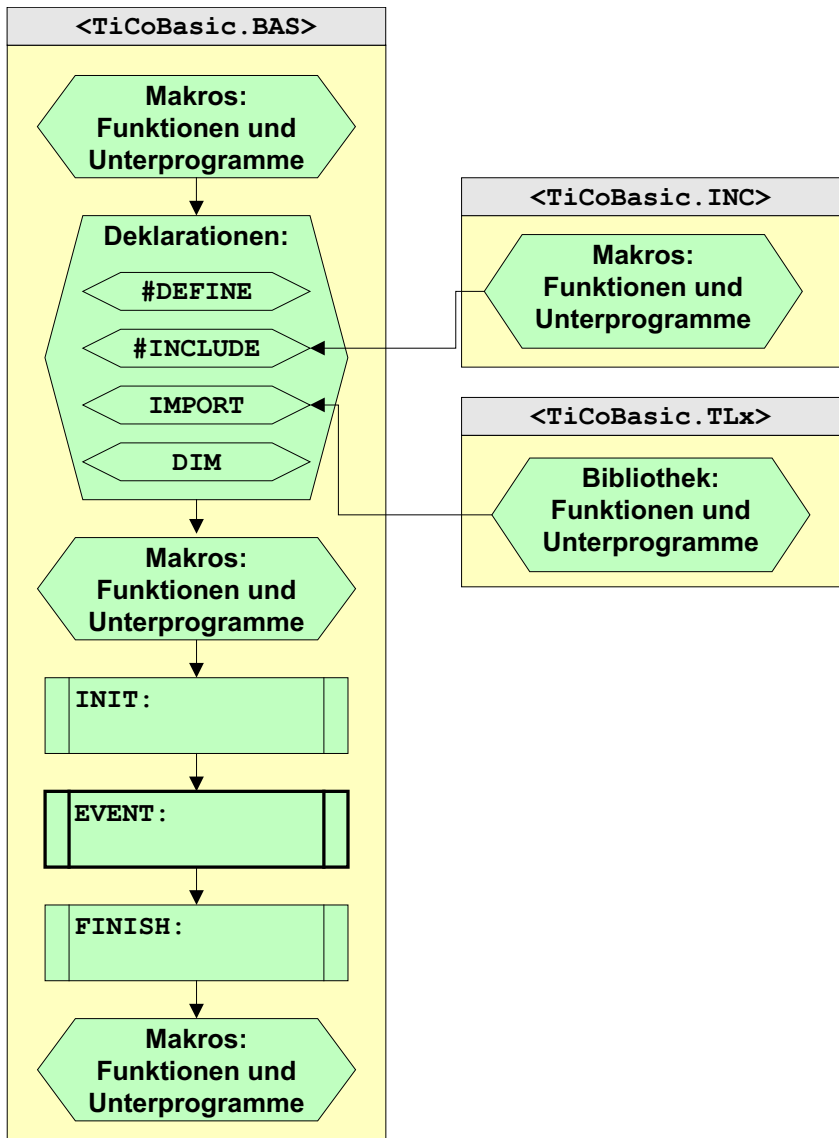


Abb. 11 – Aufbau eines *TiCoBasic*-Programms

4.1.1 Die Programmabschnitte

Die Programmabschnitte (siehe Abb. 11) beginnen jeweils mit einem Kennwort. Alle Abschnitte haben die für den Prozess eingestellte Priorität ([Dialogfenster „Process Options“](#), [Seite 60](#)).

- **Init:** ist der Programmabschnitt, der bei jedem Start des Prozesses einmalig durchlaufen wird. Er dient zur Initialisierung, z.B. von Variablen oder Datenleitungen.
- **Event:** ist der zentrale Funktionsabschnitt, der (typischerweise) in regelmäßigen Abständen aufgerufen wird, bis er gestoppt wird. Je nach Einstellung wird der Aufruf durch einen zyklischen Timer-Event oder durch einen externen Event ausgelöst.
- **Finish:** wird nach dem Stoppen eines Prozesses einmalig durchlaufen und ist daher das „Gegenstück“ zur Initialisierung.

Die Abschnitte **Init:** und **Finish:** sind optional, der Abschnitt **Event:** muss immer vorhanden sein. Im Unterschied zu *ADbasic* existiert kein Abschnitt **LowInit:**.

4.1.2 Befehle für den Hardware-Zugriff

Befehle für den Zugriff auf *ADwin*-Hardware gehören zum Lieferumfang und werden in [Include-Dateien](#) (und [Bibliotheken \(Libraries\)](#)) bereit gestellt.

Die folgenden Include-Dateien enthalten die Befehle zur Ansteuerung der Hardware-I/Os. Sie können die Dateien mit einem Textbaustein (siehe [Textbausteine einfügen](#), [Seite 46](#)) leicht in den Quelltext einbinden:

GoldIITiCo.inc

ADwin-Gold II

Textbaustein einfügen mit IG2[TAB].

AInTiCo.inc
AOut_TiCo.inc
ArincTiCo.inc
CAN_TiCo.inc
Cnt_TiCo.inc
DIO32TiCo.inc
Fieldbus_TiCo.inc
Lsbus_TiCo.inc
MIO_TiCo.inc
MIO-D12_TiCo.inc
RS_LIN_TiCo.inc
SPI_TiCo.inc

ADwin-Pro II: Befehle für mehrere Modultypen; Textbaustein einfügen mit IP[TAB].

Beachten Sie: Greifen Sie aus *TiCoBasic* nur dann auf *ADwin*-Hardware (bzw. Schnittstellen) zu, wenn sichergestellt ist, dass kein Zugriff aus *ADbasic* stattfindet.

4.1.3 Benutzerdefinierte Befehle und Variablen

Sie können [Eigene Befehle und Variablen dokumentieren](#) (Seite 27), so dass die eingefügten Hinweistexte beim Programmieren automatisch als Tooltip angezeigt werden.

Das [Fenster „Declarations“](#) (Seite 87) zeigt eine Übersicht aller definierten Befehle und Variablen an.

Alle benutzerdefinierten Namen (symbolische Namen ausgenommen) müssen mit einem Buchstaben beginnen und dürfen nur aus Buchstaben (a-z, A-Z), Ziffern (0-9) und dem Zeichen „_“ (Underscore) bestehen. Umlaute (ä, ö, ü) sind nicht erlaubt, Groß- und Kleinschreibung wird nicht unterschieden. Die Länge von Namen ist nur begrenzt durch die max. Zeilenlänge (2048 Zeichen).

Symbolische Namen

Mit der Anweisung **#Define** können Sie symbolische Namen definieren (siehe). Gruppieren Sie alle diese Definitionen am Beginn der Datei und vor dem Start der Programmabschnitte.

Symbolische Namen werden häufig für Konstanten, globale Variablen und globale Felder verwendet, aber auch für Berechnungsausdrücke.

Felder und lokale Variablen

In *TiCoBasic* müssen Sie nur lokale Variablen und alle Felder vor der Benutzung mit `Dim` deklarieren (siehe [Seite 158](#)). Die globalen Variablen `Par_n` sind bereits vordefiniert und müssen nicht deklariert werden. Nach der Deklaration haben Variablen und Felder keinen definierten Inhalt, sollten also von Ihnen initialisiert werden.



Alle diese Variablen und Felder sind innerhalb des Prozesses in allen Programmabschnitten verfügbar. Auf die globalen Variablen und Felder können Sie darüber hinaus auch aus anderen Prozessen und von der *ADwin* CPU aus zugreifen, z.B. um Daten auszutauschen.

Im Fenster „Global Variables“ ([Seite 86](#)) können alle verwendeten globalen Variablen und Felder angezeigt werden.

Makros

Makro-Funktionen `Function ... EndFunction` und -Unterprogramme `Sub ... EndSub` werden bei einem Aufruf im Programmtext an der aufrufenden Stelle eingefügt (siehe auch [Kapitel 4.5.1 auf Seite 115](#)). Makros müssen in jedem Fall außerhalb der Programmabschnitte definiert werden (siehe [Abb. 11 auf Seite 92](#)).

Die Gruppe der Makros und Libraries werden unter dem Oberbegriff Prozeduren zusammengefasst.

Libraries

Das Einbinden von Libraries muss vor dem Beginn der Programmabschnitte erfolgen. Library-Funktionen `Lib_Function ... Lib_EndFunction` und -Unterprogramme `Lib_Sub ... Lib_EndSub` sind Programm-Module mit geringerem Speicherbedarf (bei mehrfachem Aufruf) als die o.g. Makro-Funktionen und -Unterprogramme (siehe auch [Kapitel 4.5.3 auf Seite 116](#)).

Die Gruppe der Makros und Libraries werden unter dem Oberbegriff Prozeduren zusammengefasst.

4.2 Variablen und Felder

4.2.1 Übersicht

Datenstruktur	Name	Datentyp	Bemerkung
Globale Variablen und Felder			
Variable (Skalar)	Par_1...Par_80	Long	Vordefiniert, nicht deklarierbar
System-Variable	Processdelay	Long	
	Processn_running	Long	
Eindimensionales Feld (Vektor)	Data_1 [] ... Data_16 []	Long, RingBuffer	Name Data_ nicht änderbar, nur Deklaration von Feldnummer und Dimension.
Lokale Variablen und Felder			
Variable (Skalar)	frei wählbar	Long	muss deklariert werden
Eindimensionales Feld (Vektor)	frei wählbar	Long	muss deklariert werden

Wenn Sie bei der Deklaration den Speicherbereich nicht explizit festlegen, werden Variablen und Felder standardmäßig im internen Speicher DM angelegt (Speicherbelegung siehe [Kapitel 4.3.1](#)).

Der Datentyp **Long** hat eine Länge von 32 Bit.

4.2.2 Datenstrukturen

In *TiCoBasic* stehen Ihnen vor allem 2 Datenstrukturen zur Verfügung:

- Variablen (Skalare)

VAR

Eine Variable kann einen einzelnen Wert enthalten.

- Eindimensionale Felder.

ARRAY

Ein Feld besteht aus einer frei definierbaren Zahl von Feldelementen, die je einen Wert enthalten können.

Sie können eindimensionale globale Felder **Data_n** auch als Ringbuffer verwenden (Ringspeicher nach dem Prinzip: First in, first out, siehe [Kapitel 4.3.3 auf Seite 105](#)).

Die maximale Anzahl an Variablen bzw. die Größe eines Felds ist nur durch die Speichergröße des *TiCo*-Prozessors begrenzt.

Der Compiler unterscheidet

- **Globale Variablen (Parameter)** und **Globale Felder (Arrays)**:

Auf globale Datenstrukturen können sowohl alle *TiCo*-Prozesse als auch die *ADwin* CPU zugreifen, z.B. zum Austausch von Daten.

System-Variablen zählen zu den globalen Variablen (siehe Seite 101).

- **Lokale Variablen und Felder** (siehe Seite 102):

Lokale Datenstrukturen sind nur innerhalb des Prozesses verfügbar, in dem sie deklariert wurden.

Innerhalb einer Funktion oder eines Unterprogramms deklarierte Datenstrukturen sind nur lokal innerhalb der Funktion oder des Unterprogramms verfügbar.

Sie deklarieren Variablen und Felder mit der Anweisung **Dim**. Dadurch wird der Datentyp bestimmt, der erforderliche Platz im Speicher belegt und der Speicherplatz dem Variablennamen fest zugeordnet.

Zur Vereinfachung für Sie sind die globalen Variablen **Par_1 ... Par_80** bereits vordefiniert; Sie müssen (und können) diese Variablen also nicht deklarieren.

Die Deklaration globaler Felder erkennt der Compiler am Namen „**Data_n**“, wobei „**Data_**“ ein festgelegter Text ist und „**n**“ die von Ihnen festgelegte Nummer des Felds (1...16)).

Variablen und Feldelemente haben nach der Deklaration keinen definierten Wert und sollten deshalb mit einem sinnvollen Wert (z.B. Null) initialisiert werden. Ausnahme: Mit dem Übertragen eines Prozesses auf den *TiCo*-Prozessor werden die globalen Variablen **Par_1 ... Par_80** automatisch mit Null initialisiert.



4.2.3 Datentypen

Bei der Deklaration von Variablen und Feldern muss der Datentyp angegeben werden.

Der Compiler verarbeitet nur den Datentyp **Long**. Das sind ganzzahlige 32 Bit-Werte mit dem Wertebereich:

$$-2147483648 \dots +2147483647 = -2^{31} \dots +2^{31}-1$$

Im nächsten Abschnitt ist dargestellt, mit welchen Schreibweisen Sie einen Zahlenwert eingeben können.

4.2.4 Zahlenwerte eingeben

Sie können 4 verschiedene Schreibweisen benutzen, wenn Sie einen Zahlenwert angeben möchten. Die folgenden Beispiele weisen einer Variablen `x` den Wert 930 zu.

1. Dezimale Schreibweise: `x = 930`
2. Exponential-Schreibweise: `x = 93E1`

Hierbei steht „93E1“ für $93 \cdot 10^1$, d. h. nach dem „E“ folgt der (max. 2-stellige) Exponent zur Basis 10.

3. Binäre Schreibweise (angehängtes „b“): `x = 111010001b`.
4. Hexadezimale Schreibweise (angehängtes „h“): `x = 3A2h`.

Wenn der hexadezimale Wert mit einem Buchstaben (A - F) beginnt, müssen Sie eine Null voranstellen: Anstelle von „F6h“ also „0F6h“. Anderenfalls interpretiert der Compiler ihren Wert als den Namen einer lokalen Variablen.

Bei einer Zahlenwert im Quelltext können Sie die Schreibweise zwischen dezimal, binär und hexadezimal einfach mit [CTRL-SPACE] ändern.

4.2.5 Globale Variablen (Parameter)

Auf globale Variablen (und Felder) können alle laufenden *TiCo*-Prozesse und die *ADwin* CPU zugreifen; daher eignen sie sich gut zum Datenaustausch zwischen den Prozessen oder zwischen den Prozessen und der *ADwin* CPU (siehe auch [Kapitel 6.3.1 „Datenaustausch zwischen Prozessen“](#)). Ihnen stehen 80 ganzzahlige Variablen sowie bis zu 16 Felder (Arrays) vom Datentyp `Long` zur Verfügung. Alle Variablen und Feldelemente haben eine Länge von 32 Bit.

Die ebenfalls global verfügbaren [System-Variablen](#) sind auf [Seite 101](#) beschrieben.

Sie können die globalen Variablen in Ihren Programmen an beliebiger Stelle verwenden, ohne sie zu deklarieren. Die Variablen haben jedoch keinen definierten Wert und sollten deshalb mit einem sinnvollen Wert (z.B. Null) initialisiert werden. Ausnahme: Mit dem Übertragen eines Prozesses auf den *TiCo*-Prozessor werden die globalen Variablen `Par_1` ... `Par_80` automatisch mit Null initialisiert.

Die in einem Programm verwendeten globalen Variablen und Felder können im [Fenster „Global Variables“](#) ([Seite 86](#)) angezeigt werden.

Die globalen Variablen werden auch als Parameter bezeichnet und haben die Namen `Par_1`, `Par_2`, ..., `Par_80` mit dem Datentyp `Long` für ganzzahlige 32Bit-Werte.

Beispiel

```
Rem Der Parameter 5 erhält den Wert 700.
```

```
PAR_5 = 700
```

```
Rem Die Spannung am analogen Eingang 1 wird gemessen und  
Rem in Parameter 72 abgelegt.
```

```
PAR_72 = ADC(1)
```



Im Gegensatz zu den sonstigen Variablen dürfen Sie die globalen Variablen `Par_n` nicht deklarieren, da sie vordefiniert und dem Compiler bereits bekannt sind.



4.2.6 Globale Felder (Arrays)

Die globalen Felder ermöglichen Ihnen, große Datenmengen zwischen den Prozessen auf dem *ADwin*-System oder der *ADwin* CPU auszutauschen (siehe auch [Kapitel 6.3.1 „Datenaustausch zwischen Prozessen“](#)). Ihnen stehen bis zu 16 globale Felder (Arrays) vom Datentyp `Long` zur Verfügung.

Da Größe und Datentyp wählbar sind, müssen Sie globale Felder am Anfang Ihres Programms deklarieren (siehe [Dim](#)) und möglichst auch initialisieren (die Feldelemente haben sonst keinen definierten Wert).



Die in einem Programm verwendeten globalen Felder und Variablen können im [Fenster „Global Variables“](#) ([Seite 86](#)) angezeigt werden.

Die Deklaration eines globalen Felds erkennt der Compiler am Namen „`Data_n`“, wobei „`Data_`“ ein festgelegter Text ist und „`n`“ die von Ihnen festgelegte Nummer des Felds (1...16)). Die Namen für `Data`-Felder sind also:

Data_1, Data_2, ..., Data_16.

Andere Feldnummern sind unzulässig. Sie können jedoch die Feldnummern frei wählen, auch die Deklaration von z.B. **Data_5** (ohne **Data_1 ... Data_4**) ist gültig. In Ihrem Programm werden die Felder vom Compiler anhand ihrer Nummer unterschieden.



Beispiel

```
REM Feld 5 mit 20000 Elementen vom Typ Long deklarieren.
Dim DATA_5[20000] As Long
```

Die maximale Größe der Felder richtet sich nur nach dem verfügbaren Speicherplatz. Beispielsweise kann auf einem *TiCo*-Prozessor mit 256 MiB Speicher ein Feld mit über 67 Millionen Elementen vom Typ **Long** deklariert werden.

Nachdem das Feld deklariert ist, können Sie auf jedes einzelne Element zugreifen. Das erste Element eines Felds besitzt den Index 1.



Weisen Sie *auf keinen Fall* dem Element 0 eines Felds einen Wert zu, z.B. mit **Data_1[0] = ...**



Beispiel

```
Rem Der globalen, ganzzahligen Variablen Par_1 wird der
Rem Wert des 200. Elements aus dem Feld 5 zugewiesen.
PAR_1 = DATA_5[200]

Rem Durch diese Anweisung erhält das 345. Element aus
Rem dem Feld Data_5 den Wert 4000.
DATA_5[345] = 4000
```

Sie können den Index eines *Feldelements* auch über eine Variable übergeben:

```
Rem Auch hier wird, wie im Beispiel davor, dem 345.
Rem Element des
Rem Felds Data_5 der Wert 4000 zugewiesen.
nummer1 = 345
DATA_5[nummer1] = 4000
```

Dagegen darf die Nummer eines Felds nicht durch eine Variable übergeben werden. Die folgende Anweisung führt zu einer Fehlermeldung des *TiCoBasic*-Compilers:



```
num = 2
DATA_num[300] = 20      'FALSCH !!
DATA_2[300] = 20        'RICHTIG
```

Der Compiler interpretiert `Data_num` als Namen eines lokalen Felds, das (wahrscheinlich) nicht deklariert wurde und daher nicht verfügbar ist. Verwenden Sie statt dessen die Schreibweise `Data_2`. Beachten Sie die unterschiedliche Syntax-Hervorhebung bei den Variablen.

4.2.7 System-Variablen

Um Informationen über den Status des *TiCo*-Prozessors zu erhalten, stehen Ihnen die folgenden System-Variablen zur Verfügung. Diese Variablen sind global, d.h. für alle *TiCo*-Prozesse und von der *ADwin* CPU aus verfügbar. Weitere Informationen finden Sie bei den Befehlsbeschreibungen.

Process_n_Running

Zeigt den Status des Prozesses `n` an (mit `n = 1...10`), d.h. ob der Prozess läuft, gerade angehalten wird oder gestoppt ist ([siehe Seite 202](#)). Die Variable kann nur gelesen werden.

Process_Error

Zeigt bei aktiviertem Debug-Modus den zuletzt aufgetretenen Fehler des Prozesses `n` an (mit `n = 1...4`, [siehe Seite 201](#)). Die Variable kann nur gelesen werden.

Processdelay

Der Soll-Zeitabstand, in dem zeitgesteuerte Prozesse vom Zähler aufgerufen werden, ist das `Processdelay`, auch Zykluszeit genannt. Mit der Systemvariablen `Processdelay` ([siehe auch Seite 199](#)) können Sie die Zykluszeit abfragen oder einstellen. Die Zykluszeit wird in Taktzyklen des Zählers gemessen.

Sie können die Variable `Processdelay` nur innerhalb der Abschnitte **Init:** und **Event:** lesen und beschreiben. Das Beschreiben der Variablen ist jedoch nur 1mal pro Abschnitt erlaubt.

Achten Sie darauf, dass die Auslastung des Prozessors möglichst weniger als 90% beträgt, keinesfalls aber 100% übersteigen darf.



4.2.8 Lokale Variablen und Felder



Alle lokalen Variablen und Felder, die Sie für Ihren Prozess benötigen, müssen Sie vor dem Beginn des ersten Abschnitts in Ihrem *TiCo-Basic*-Programm deklarieren und möglichst auch initialisieren (sie haben sonst keinen definierten Wert).

Namen müssen mit einem Buchstaben beginnen und dürfen nur aus Buchstaben (a-z, A-Z), Ziffern (0-9) und dem Zeichen „_“ (Underscore) bestehen.¹ Umlaute (ä, ö, ü) sind nicht erlaubt, Groß- und Kleinschreibung wird nicht unterschieden. Die Länge von Variablennamen ist nur begrenzt durch die max. Zeilenlänge (2048 Zeichen).

Bei (skalaren) Variablen sind als Datentypen ganzzahlige Werte in 32 Bit ([Long](#)) verfügbar.



Beispiel

```
REM Definiere die Variable 'wert' mit dem Datentyp Long
Dim wert As Long
```

Variablen können Sie nicht nur als skalare Größe, sondern auch als eindimensionales Feld deklarieren, das heißt, Sie können Felder von Variablen erzeugen und verarbeiten. Die Anzahl der zu dimensionierenden Elemente im Feld wird in eckigen Klammern nach dem Namen eingegeben.



Beispiel

```
Dim wert[100] As Long 'Definiert ein Feld der
                      'Länge 100 mit Namen 'wert'
                      'und dem Datentyp Long
```



Das erste Element eines Felds besitzt den Index 1, im Beispiel: `wert[1]`. Auf das Element mit dem Index 0 dürfen Sie nicht zugreifen.

1. Beachten Sie: Namen lokaler Felder dürfen nicht mit der Zeichenfolge `Data_` beginnen, anderenfalls entsteht ggf. ein Compiler-Fehler.

4.3 Variablen und Felder – Details

4.3.1 Variablen und Felder im Datenspeicher

Sie können für Felder und lokale Variablen explizit festlegen, in welchem Speicherbereich (siehe unten) sie angelegt werden. Diese Festlegung geschieht bei der Deklaration mit `Dim` im Quelltext durch die Zusätze `At Dm_Local` oder `At Dram_Extern`. Wenn Sie bei der Deklaration keinen Zusatz verwenden, werden Variablen und Felder im internen Speicher DM angelegt

Wie empfehlen Ihnen die Verwendung des internen Speichers für Variablen und (kleine) Felder, auf die Sie sehr schnell zugreifen möchten. Der langsamere externe Speicher ist – falls vorhanden – wegen seiner Größe vorwiegend für Felder geeignet.

In [Abb. 12](#) sehen Sie Beispiele für Deklarationen, um Variablen und Felder in den verschiedenen Speicherbereichen anzulegen.

Variable / Feld	Speicher- bereich	Deklaration im Quelltext
Lokale Variable	intern (DM)	<code>Dim var As Long</code> oder <code>Dim var As Long At DM_Local</code>
	extern (DX)	<code>Dim var As Long At Dram_Extern</code>
Feld (global oder lokal)	intern (DM)	<code>Dim array[5] As Long At DM_Local</code>
	extern (DX)	<code>Dim array[5] As Long</code> oder <code>Dim array[5] As Long At Dram_Extern</code>

Abb. 12 – Festlegung des Speicherbereichs bei Deklarationen

Die globalen Variablen `Par_1...Par_80` sind vordefiniert und stehen immer im internen Speicher DM zur Verfügung. Sie brauchen und können diese daher nicht neu (z.B. für den externen Speicher) deklarieren.

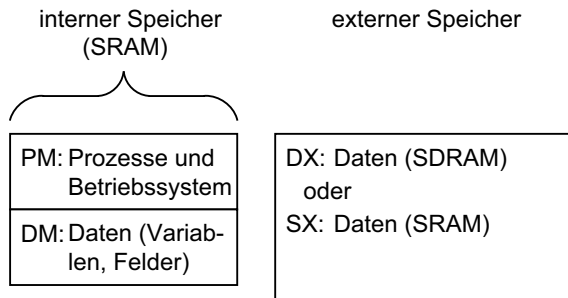


4.3.2 Speicherbereiche

Der *TiCo*-Prozessor verwendet einen schnellen internen Speicher (SRAM) und – falls vorhanden – einen großen externen Speicher

(SDRAM). Manche Pro II-Module besitzen anstelle des externen SDRAM ein schnelleres SRAM (SX).

Je die Hälfte des internen Speichers steht als Programmspeicher PM und als Datenspeicher DM zur Verfügung.



– Programmspeicher (PM)

Der Programmspeicher belegt die Hälfte des internen SRAM und nimmt das Betriebssystem und Ihre Prozesse auf.

– Datenspeicher intern (DM)

Der interne Datenspeicher belegt die Hälfte des internen SRAM und nimmt die globalen und lokalen Variablen und Felder auf).

– Externer Speicher (DX, SX)

Der externe Speicher belegt das externe SDRAM.

Bei manchen Pro II-Modulen (z.B. Pro II-MIO-TiCo) ist anstelle des SDRAM ein SRAM-Speicher eingebaut.



Der Zugriff auf den externen Speicher ist immer mit einer – noch dazu schwankenden – Wartezeit (Jitter) verbunden; ausgenommen davon ist der Zugriff über [Die Datenstruktur RingBuffer \(siehe Seite 105\)](#). Beachten Sie hierzu [Kapitel 5.2.6 auf Seite 124](#).

Sie können auf Daten im internen Speicher DM deutlich schneller zugreifen als auf Daten im externen Speicher DX. Bei externem SRAM und internem Speicher ist der Zugriff etwa gleich schnell. Die Zugriffszeiten auf die Speicherbereiche PM und DM im internen SRAM sind gleich.


Die Speichergröße (SRAM, SDRAM) ist eine Bestelloption und kann nicht nachträglich vergrößert werden.

Die Größe der Speicherbereiche ist der einzige Faktor, der die Größe von Prozessen und die Zahl der deklarierbaren Variablen und Felder begrenzt (indirekt auch die Größe der Quelltext-Dateien). Sie sehen in der Statusleiste der Entwicklungsumgebung, wieviel Speicher Ihnen in PM, DM und DX insgesamt zur Verfügung steht (angegeben in Bytes).

4.3.3 Die Datenstruktur RingBuffer


Um große Datenmengen kontinuierlich und schnell zu übertragen, empfehlen wir globale Felder `Data_n` mit der Datenstruktur `RingBuffer`: Ein Ringspeicher, der nach dem Prinzip „First In, First Out“ verwaltet wird.

Die grundsätzlichen Eigenschaften von globalen Feldern sind in [Kapitel 4.2.6 „Globale Felder \(Arrays\)“ auf Seite 99](#) beschrieben.

Ein `Ringbuffer` ist nicht zu verwechseln mit der Datenstruktur FIFO der *ADwin* CPU. Der FIFO-Ringspeicher ist im Handbuch *ADbasic* beschrieben. 

Ringspeicher sind für verschiedene Anwendungen sinnvoll; allerdings schließen sich die Anwendungen gegenseitig aus:

- Der *TiCo*-Prozess greift auf Daten im externen DRAM zu und zwar lesend wie schreibend über den gleichen Ringspeicher.
- *ADbasic*-Prozesse (auf der *ADwin* CPU) und *TiCoBasic*-Prozesse tauschen über einen Ringspeicher miteinander Daten aus. Für jede Übertragungsrichtung ist ein separater Ringspeicher erforderlich.
- Mehrere Prozesse auf dem *TiCo*-Prozessor tauschen über einen Ringspeicher miteinander Daten aus. Es sind 2 Ringspeicher erforderlich, einer zum Schreiben und einer zum Lesen.

Der Umgang mit der Datenstruktur `RingBuffer` ist nicht einfach. Bei falscher Anwendung können schwer auffindbare Fehler entstehen. Die Verwendung der Datenstruktur `RingBuffer` ist daher erfahrenen Benutzern von *ADbasic* und *TiCoBasic* vorbehalten. 

Wie arbeitet der Ringspeicher?

In einem Ringspeicher werden die Daten auf besondere Weise verwaltet. Sie können sich die Daten als eine Kette vorstellen, an deren

Ende Sie neue Daten einzeln anhängen und an deren Spitze Sie einzelne Daten abholen können. Sie greifen also – im Unterschied zu einem „einfachen“ Feld – nicht auf beliebige Feldelemente zu, sondern immer nur auf das erste oder letzte (über je einen Datenzeiger). Dadurch lesen Sie die Daten in der gleichen Reihenfolge aus, wie sie in das Feld geschrieben wurden (=First In, First Out).

Da ein RingBuffer-Feld eine endliche (von Ihnen deklarierte) Zahl von Elementen besitzt, bildet die Kette aus benutzten und unbenutzten Feldelementen einen Ring, den Ringspeicher. Die Datenzeiger auf das erste und das letzte benutzte Feldelement werden automatisch verwaltet, wenn Sie dem Feld einen neuen Wert zuweisen oder einen Wert auslesen.



Aus der Ringstruktur des RingBuffer-Felds ist ersichtlich, dass die Spitze der Datenkette das Datenende „überholen“ kann. Dies ist möglich, wenn Sie Daten schneller in den RingBuffer schreiben als Sie auslesen. Dadurch werden alte gespeicherte Daten überschrieben und gehen somit verloren.

Ringspeicher deklarieren und anwenden

Ein Ringspeicher wird mit `Dim` deklariert:

Beispiel

```

Rem Lese-Ringspeicher mit 103 Elementen im externen
  Speicher
Dim DATA_1[103] As Long As RingBuffer_For_Read At
  DRAM_Extern
Rem Schreib-Ringspeicher mit 1000 Elementen
Rem im internen Speicher
Dim DATA_2[1000] As Long As RingBuffer_For_Write At
  DM_Local
Rem Lese- und Schreib-Ringspeicher mit 199 Elementen im
Rem externen Speicher
Dim DATA_3[199] As Long As RingBuffer_For_Read At
  DRAM_Extern
Dim DATA_3[199] As Long As RingBuffer_For_Write At
  DRAM_Extern

```

Zu Feldgrößen im externen Speicher beachten Sie bitte den Abschnitt [RingBuffer](#).

Wenn kein Speicherbereich angegeben wird, verwendet der Compiler die Voreinstellung `DM_Local`. Wir empfehlen als Erinnerungsstütze, den Speicherbereich bei der Deklaration immer anzugeben.

Beachten Sie bitte, dass Sie ein globales Feld Data nicht gleichzeitig als „einfaches“ Feld und als Ringspeicher verwenden können.



Auf ein RingBuffer-Feld greifen Sie zu, indem Sie dessen Feldnamen (mit der entsprechenden Feldnummer) angeben.

Beispiel



```
Dim DATA_5[1000] As Long As RingBuffer_For_Read At
    DM_Local
Dim DATA_5[1000] As Long As RingBuffer_For_Write At
    DM_Local
REM Schreibt in den RingBuffer mit der Nummer 5 den Wert 95.
DATA_5 = 95
PAR_7 = DATA_5
REM Liest einen Wert aus dem RingBuffer und speichert ihn in
REM der globalen Variablen Par_7.
```

Um sicherzustellen, dass noch Platz im RingBuffer ist, sollten Sie vor dem Schreiben die Funktion `RingBuffer_Empty` verwenden. In gleicher Weise prüfen Sie mit der Funktion `RingBuffer_Full` vor dem Lesen, ob noch nicht gelesene Werte vorhanden sind.

In Bezug auf die folgenden Regeln bildet der externe Speicher `SRAM_Extern` (SX) mit dem internen Speicher `DM_Local` einen gemeinsamen Speicherbereich. Auf bestimmten Pro II-Modulen ersetzt `SRAM_Extern` den externen Speicher `DRAM_Extern`.

Allgemeine Regeln zur Deklaration von Ringspeichern:

- Für jeden Speicherbereich sind 2 Ringspeicher-Deklarationen erlaubt.
- Im externen Speicher `DRAM_Extern` ist je ein Ringspeicher zum Lesen und ein Ringspeicher zum Schreiben erlaubt.

Im Speicherbereich `DM_Local` + `SRAM_Extern` sind Kombinationen von Lese- und Schreib-Ringspeichern möglich. Sie können also auch 2 Lese-Ringspeicher oder 2 Schreib-Ringspeicher deklarieren.

- Es ist nicht erlaubt, im externen Speicher `DRAM_Extern` neben einem Ringspeicher auch normale Felder zu deklarieren.

**Beispiel**

```
Rem 2 Ringspeicher im externen Speicher; damit sind  
normale  
Rem Felder im externen Speicher nicht mehr erlaubt!  
Dim DATA_5[199] As Long As Ringbuffer_For_Read At  
    DRam_Extern  
Dim DATA_5[199] As Long As Ringbuffer_For_Write At  
    DRam_Extern  
  
Rem 2 Ringspeicher im internen Speicher  
Rem Normale Felder sind außerdem möglich  
Dim DATA_1[200] As Long As Ringbuffer_For_Read At  
    DM_Local  
Dim DATA_2[200] As Long As Ringbuffer_For_Read At  
    DM_Local  
Dim DATA_3[200] As Long At DM_Local
```

Auf externes DRAM zugreifen

Der Zugriff auf globale und lokale Felder im externen Speicher ist relativ langsam. Ein schneller Datenaustausch ist dagegen mit einem Ringspeicher möglich. Hierbei schreibt und liest der *TiCo*-Prozess die Daten über den gleichen Ringspeicher.

Beispiel



*Rem Schreib- und Lese-Ringspeicher im externen Speicher
Rem Damit sind normale Felder im DRAM_Extern nicht mehr
Rem erlaubt!*

```
Dim DATA_5[199] As Long As Ringbuffer_For_Read At
    DRAM_Extern
Dim DATA_5[199] As Long As Ringbuffer_For_Write At
    DRAM_Extern
Dim free,used,value1 As Long
```

Init:

Rem Lese-Ringspeicher Data_5 initialisieren
RingBuffer_Clear(5, Par_1)

Event:

Rem Sind noch Elemente zum Beschreiben frei?
free = **Ringbuffer_Empty**(5,0)
If (**free** > 0) **Then**
 DATA_5 = **value1**
EndIf
Rem Können noch benutzte Elemente gelesen werden?
used = **Ringbuffer_Full**(5,0)
If (**used** > 0) **Then**
 PAR_7 = **DATA_5**
EndIf

Nach der Deklaration eines Lese-Ringspeichers im externen Speicher sollten Sie den Ringspeicher mit dem Befehl **RingBuffer_Clear** initialisieren.

Datenaustausch zwischen TiCo-Prozessen

Zwei TiCo-Prozesse in einem Projekt (siehe auch [Kapitel 6.3.1 auf Seite 134](#)) können über einen Ringspeicher kontinuierlich und schnell miteinander Daten austauschen. Der Ringspeicher kann dazu im (kleineren) internen Speicher oder im (langsameren) externen Speicher liegen.

Der Datenaustausch arbeitet nur korrekt, wenn der Datenfluss eindeutig ist, also nur der eine Prozess in den Ringspeicher schreibt und nur der andere Prozess daraus liest. Dabei ist auch eine Umkehr des Datenflusses möglich, solange der Datenfluss eindeutig bleibt.

**Beispiel****Prozess 1 schreibt Daten:**

```

Rem Schreib-Ringspeicher im internen Speicher
Dim DATA_5[500] As Long As Ringbuffer_for_Write At
  DM_Local
Dim free,value1 As Long

```

Init:

```

Rem Lese-Ringspeicher Data_5 initialisieren
RingBuffer_Clear(5, Par_1)

```

Event:

```

Rem Sind noch Elemente zum Beschreiben frei?
free = Ringbuffer_Empty(5, 0)
If (free > 0) Then
  DATA_5 = value1
EndIf

```

Prozess 2, der Daten liest:

```

Rem gleiches Feld, aber als Lese-Ringspeicher
Dim DATA_5[500] As Long As Ringbuffer_for_Read At
  DM_Local
Dim used As Long

```

Event:

```

Rem Können noch benutzte Elemente gelesen werden?
used = Ringbuffer_Full(5, 0)
If (used > 0) Then
  PAR_7 = DATA_5
EndIf

```

Datenaustausch mit ADbasic-Prozessen

ADbasic-Prozesse (auf der ADwin CPU) und TiCoBasic-Prozesse können über einen Ringspeicher kontinuierlich und schnell miteinander Daten austauschen. Der Ringspeicher kann dazu im (kleineren) internen Speicher oder im (langsameren) externen Speicher liegen.

Für jede Übertragungsrichtung ist jeweils ein eigener Ringspeicher erforderlich, der nur in TiCoBasic deklariert wird. Der Datenaustausch arbeitet nur korrekt, wenn der Datenfluss eindeutig ist, also nur der eine Prozess in den Ringspeicher schreibt und nur der andere Prozess daraus liest. Eine Umkehr des Datenflusses ist nicht möglich.

Die Datenübertragung verwendet eine globale Variable `Par_n` des *TiCo*-Prozessors zur Synchronisation. Hierzu trägt der *ADbasic*-Befehl den aktuellen Wert des Schreib- oder Lesezeigers – je nach Richtung des Datenflusses – in die Variable ein, so dass in *TiCoBasic* bekannt ist, wieviele Daten gelesen oder geschrieben werden können.

Beispiel



TiCoBasic-Prozess, der Daten schreibt

```
Rem Schreib-Ringspeicher im internen Speicher
#Define free Par_1
Dim DATA_1[500] As Long As Ringbuffer_for_Write At
    DM_Local

Init:
    Rem Schreib-Ringspeicher Data_1 initialisieren
    RingBuffer_Clear(1, Par_5)

Event:
    Rem Sind noch Elemente im Data_1 zum Beschreiben frei?
    Rem In ADbasic wurde der Lesezeiger im TiCo-Parameter Par_5
    Rem gesetzt. Damit kann hier die Anzahl der freien Elemente
    Rem gelesen werden.
    free = Ringbuffer_Empty(1, Par_5)
    If (free > 0) Then
        DATA_1 = Par_3
    EndIf
```

ADbasic-Prozess, der Daten liest (hier für ADwin-Gold II)

```
#Include ADwinGoldII.inc
Dim DATA_10[300] As Long 'Feld für gelesene Daten
Dim tset[150] As Long 'Feld für TiCo-Einstellungen
Dim val As Long 'Fehlerwert

Init:
    Rem Datenübertragung zum TiCo-Prozessor 1 initialisieren
    TDrv_Init(1, tset)

Event:
    Rem Bis zu 250 Werte aus dem TiCo-Feld Data_1 lesen und
    Rem in Data_10 speichern. Der Lesezeiger wird in den
    Rem TiCo-Parameter Par_5 geschrieben.
    val = Get_TiCo_RingBuffer(tset, 1, DATA_10, 1, 250, 1,
    5, 0)
    If (val > 0) Then
        Rem Daten wurden gelesen, Daten weiter verarbeiten
    EndIf
```


4.4 Berechnungsausdrücke

Ein Berechnungsausdruck ist das, was Sie einer Variablen zuweisen oder einem Befehl als Argument übergeben. Er besteht aus einer beliebigen Kombination von:

- einfachen Daten: Konstante, Variable oder Feldelement.
- Operatoren, die auf Argumente angewendet werden, die wieder Berechnungsausdrücke sind.
- Befehlen aus dem *TiCoBasic*-Befehlssatz oder benutzerdefinierte Befehlen.

4.4.1 Auswertung von Operatoren

Für die Auswertung eines Berechnungsausdrucks (Definition siehe [Kapitel 4.4](#)) ist es wesentlich, in welcher Reihenfolge die Operatoren angewendet werden. Hierzu werden die Operatoren in Kategorien eingeteilt, die nach Prioritäten geordnet sind: Eine Kategorie höherer Priorität wird vor einer Kategorie niedriger Priorität bearbeitet (siehe).

Operator	Kategorie
" "	Begrenzer von Zeichenketten
Kennwort in <i>TiCoBasic</i>	Befehl, Funktion, Variable, etc.
=	Zuweisung
()	Klammern
-	Vorzeichen einer <i>Konstanten</i>
* /	Punkt-Operatoren
+ -	Strich-Operatoren
And Or XOr	Binär-Operatoren
< > =	Vergleichs-Operatoren
And Or	Boolesche Operatoren

Abb. 13 –

Abb. 14 – Prioritäten von Operatoren-Kategorien
(von oben nach unten absteigende Priorität)

**Beispiel**

```
var = PAR_1 + PAR_2 * PAR_1^3 / 4
```

entspricht

```
var = PAR_1 + (PAR_2 * (PAR_1^3) / 4)
```

Wenn sich 2 Operatoren in der gleichen Kategorie befinden (oder gleiche Operatoren vorhanden sind), dann verarbeitet der Compiler diese wie sie erscheinen, von links nach rechts.



Wenn Sie Variablen mit negativem Vorzeichen verwenden, kann dies in manchen Fällen zu unerwarteten Ergebnissen führen, die Sie durch Klammersetzung vermeiden.

**Beispiel**

```
var = 1/-x
```

'nicht empfohlene Schreibweise

```
var = 1 / (-x)
```

'Korrekt: Negativer Umkehrwert

4.5 Bedingungen, Schleifen und Module

Wenn Sie Programme schreiben, strukturieren Sie diese in *TiCoBasic* mit folgenden Elementen:

- Kontrollstrukturen verkürzen aufwändige Abschnitte.
 - Schleifen für oft wiederholte Abschnitte:
 Do ... Until oder
 For ... Next.
 - Abfragen für fallweise Unterscheidungen:
 If ... EndIf, #If ... #EndIf oder
 SelectCase ... EndSelect.
- Unterprogramm- und Funktions-Makros ermöglichen Ihnen, häufig benutzte Programmabschnitte zu definieren als
 - Unterprogramm-Makros mit Sub ... EndSub
 - Funktions-Makros mit Function ... EndFunction
- Bibliotheken (Libraries) von kompilierten Funktionen und Unterprogrammen, die Sie mit Import in den Quelltext einbinden:
 - Library-Unterprogramme: Lib_Sub ... Lib_EndSub
 - Library-Funktionen: Lib_Function ... Lib_EndFunction
- Sammlungen von Quelltext-Abschnitten und Programm-Makros in Include-Dateien, die Sie komplett in den Quelltext einbinden mit
 #Include filename.Inc

Sie finden nähere Erklärungen und Beispiele der Befehle in [Kapitel 7 „Befehlsreferenz“](#).

4.5.1 Unterprogramm- und Funktions-Makros

Unterprogramme und Funktionen definieren Makros, d.h. deren vollständiger Anweisungsblock wird (noch vor dem Kompilieren) an der aufrufenden Stelle in den Quelltext eingefügt.

Die Syntax von Unterprogramm- und Funktions-Makros ist sehr einfach, Sie müssen lediglich die Begriffe `Sub ... EndSub` und `Function ... EndFunction` wie eine Klammer um die jeweiligen Programabschnitte legen. Funktionen geben – im Unterschied zu Unterprogrammen – einen Wert zurück.

Makros erhöhen die Übersichtlichkeit Ihres Quelltextes. Beachten Sie aber auch, dass jeder Aufruf die erzeugte Binärdatei vergrößert. Sie können alternativ auch Library-Funktionen oder -Unterprogramme verwenden (siehe unten).

Sie finden nähere Informationen zum Aufbau von Makros in der Befehlsreferenz ([Seite 167: Function ... EndFunction](#); [Seite 223: Sub ... EndSub](#)).

4.5.2 Include-Dateien

Sie können eine Sammlung von Quelltext-Abschnitten erstellen und in einer sogenannten „Include-Datei“ speichern. Solche Dateien (bzw. den darin enthaltenen Quelltext) können Sie sehr einfach mit dem Befehl `#Include` in Ihren aktuellen Quelltext einbinden.

Der Inhalt von Include-Dateien unterliegt den gleichen Regeln wie der von normalen Quelltext-Dateien, vorwiegend enthalten sie jedoch nur Unterprogramm- und Funktions-Makros.

Zum Erstellen einer Include-Datei geben Sie, wie bei einer „normalen“ *TiCoBasic*-Datei, den gewünschten Quelltext ein und speichern diesen mit „File / Save as“ als Dateityp „Include file *.Inc“.

Je nach enthaltenem Quelltext müssen Sie darauf achten, an welcher Stelle Sie die Include-Datei in Ihren aktuellen Quelltext einbinden, damit die korrekte Programmstruktur gewahrt bleibt. Wenn die Include-Datei Unterprogramm- und Funktions-Makros enthält, muss sie beispielsweise vor dem Abschnitt `Init:` oder nach dem Abschnitt

Finish: eingebunden werden.

Sie können Include-Dateien auch in Quelltexte von Library-Dateien oder anderen Include-Dateien einbinden.



Die Include-Dateien, die mit *TiCoBasic* geliefert werden, enthalten Unterprogramm- und Funktions-Makros, die Befehle für den Hardware-Zugriff definieren; siehe [Befehle für den Hardware-Zugriff auf Seite 93](#). Aus diesem Grund ist die korrekte Stelle für das Einbinden dieser Dateien der Anfang des Quelltexts (siehe [Seite 92](#)).

Sobald eine Include-Datei mit **#Include** in einen Quelltext eingebunden ist, werden die darin definierten Befehle im Quelltext hervorgehoben (siehe [Syntax hervorheben](#)).

4.5.3 Bibliotheken (Libraries)

In einer Bibliothek können Sie kompilierte Library-Unterprogramme und -Funktionen (Module) zusammenfassen. Mit dem Befehl **Import** binden Sie diejenigen Module einer Library in einen Prozess ein, die dort tatsächlich aufgerufen werden.

Die Library-Module sind den Funktions- und Unterprogramm-Makros ähnlich. Sie erstellen diese in einem Quelltext mit den Befehlen **Lib_Sub ... Lib_EndSub** und **Lib_Function ... Lib_EndFunction** und kompilieren daraus die Library-Datei mit „Build / Make lib file“.

Wenn Sie einen Quelltext kompilieren, in dem eine Library importiert wird, werden nur die im Quelltext aufgerufenen Library-Module zu der Binärdatei hinzugefügt. Ein mehrfacher Aufruf im Quelltext vergrößert die Binärdatei nicht (im Gegensatz dazu siehe auch [Kapitel 4.5.1 „Unterprogramm- und Funktions-Makros“](#)), jedoch benötigt jeder einzelne Aufruf auch zusätzliche Ausführungszeit.



Beachten Sie bitte, dass ein Library-Modul kein Library-Modul innerhalb der gleichen Library-Datei aufrufen kann. Wir empfehlen Ihnen, statt dessen Funktions- und Unterprogramm-Makros zu verwenden. Alternativ können Sie auch eine zusätzliche Library erstellen (oder mehrere).

Wenn Sie Libraries verschachtelt einbinden (d.h. in einer Library eine weitere Library einbinden), müssen Sie im aufrufenden Quelltext die

Libraries aller Schachtelungsstufen einbinden (siehe Abb. 15), sonst erhalten Sie eine Fehlermeldung des Compilers.

Rekursive Aufrufe von Library-Funktionen oder Unterprogrammen sind nicht erlaubt.



Sie finden nähere Informationen zum Aufbau von Library-Modulen in der Befehlsreferenz (Seite 182: [Lib_Function ... Lib_EndFunction](#); Seite 187: [Lib_Sub ... Lib_EndSub](#)).

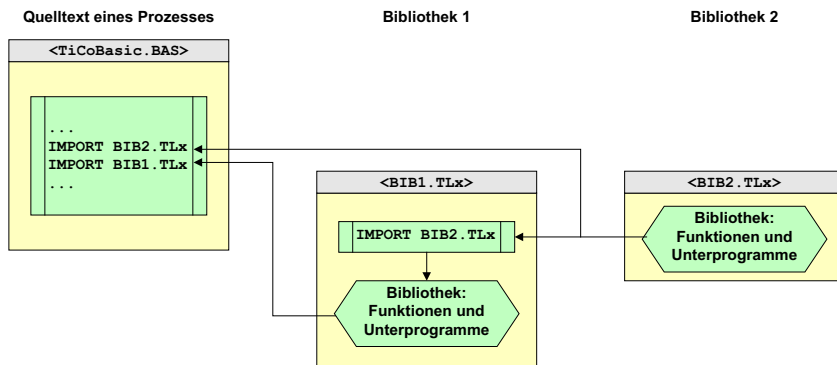


Abb. 15 – Verschachteltes Einbinden von Bibliotheken

Makros kontra Bibliotheken

Was tun Sie, wenn Sie sowohl ein Makro als auch eine Bibliothek zur Lösung einer Aufgabe verwenden können? In diesem Fall müssen Sie zwischen Programmgeschwindigkeit und Programmgröße abwägen.

Ein Makro wird im Code so oft expandiert, wie es im Quelltext aufgerufen wird. Wenn Ihr Programm eine Makro-Funktion 100-mal aufruft, gibt es 100 Kopien dieses Makros im endgültigen Programm. Im Gegensatz dazu gibt es den Code einer Library-Funktion nur einmal. Deshalb wäre es hinsichtlich einer geringen Programmgröße besser, eine Library-Funktion zu wählen.

Wenn ein Programm eine Library-Funktion aufruft, erfordert das einen bestimmten Verarbeitungsaufwand, um die Programmausführung an die Bibliothek zu übergeben und später wieder in das aufrufende Programm zurückzukehren. Beim „Aufrufen“ eines Makros gibt es diesen Verarbeitungsaufwand nicht, da der Code bereits im Programm steht.

Hinsichtlich einer höheren Ausführungsgeschwindigkeit liegen die Vorteile also bei den Makros.

Das Abwägen zwischen Programm-Größe und -Geschwindigkeit ist für den Programmieranfänger eher unerheblich. Wichtig werden diese Betrachtungen erst bei großen oder zeitkritischen Anwendungen.

5 Prozesse optimieren


Der *Tico*-Prozessor ist dafür ausgelegt, Regel-, Steuer- und Messaufgabe schnell und präzise auszuführen. Je nach Anforderung kann es erforderlich werden, dass Sie Ihr *TiCoBasic*-Programm für eine schnellere Bearbeitungszeit optimieren.

Im folgenden zeigen wir beispielhaft, mit welchen Mitteln und an welchen Stellen Sie bei einer Optimierung ansetzen können. Die Vorgehensweise hängt von vielen Faktoren ab und ist daher auf den Einzelfall abzustimmen.

5.1 Bearbeitungszeit messen

Als Grundlage für eine Optimierung ist es wichtig, die Bearbeitungszeit eines Prozesszyklus oder von Programmabschnitten zu messen. Sie verwenden hierzu den internen Zähler des *Tico*-Prozessors.

Der *Tico*-Prozessor verfügt über einen internen Zähler, der in Zeittakten von 10 ns oder 20 ns hochgezählt wird. Mit dem Befehl `Read_Timer()` können Sie den aktuellen Zählerstand feststellen.

Nach dem Einschalten der Spannungsversorgung wird der Zähler auf den Wert 0 (Null) gesetzt und anschließend in festen Zeittakten kontinuierlich hochgezählt. 

Sie messen die Bearbeitungszeit von Programmen als Zeitdifferenz. Im folgenden Beispiel wird die Bearbeitungszeit eines zeitkritischen Abschnitts (abzüglich eines Offsets) in der globalen Variablen `Par_1` gespeichert.

Sie erhalten den Offset, wenn Sie die beiden `Read_Timer()`-Zeilen nacheinander – ohne dazwischen liegende Anweisungen – ausführen und die Differenz dieser Werte bilden. Der Offset muss für das betrachtete Programm nur ein Mal ermittelt werden.

**Beispiel**

```
Dim t1, t2 As Long
```

Event:

```
...
t1 = Read_Timer()
...
t2 = Read_Timer()
PAR_1 = t2 - t1 - 4
```

*'zeitkritischer Abschnitt
'Bearb.zeit des zeitkritischen
'Abschnitts in Zeittakten
'(Offset = 4 Zeittakte)*

Wenn `Par_1` im obigen Beispiel den Wert 37 erhält, hat der zeitkritische Abschnitt mit dem Prozessor TiCo1 $37 \times 20\text{ns} = 740\text{ns}$ benötigt.

Sie können die Zeitmessung auch benutzen, um beispielsweise die Zeitdifferenz zwischen zwei externen Event-Signalen zu messen. Im Beispiel wird diese bei jedem Aufruf in der globalen Variablen `Par_1` gespeichert.

**Beispiel**

```
Dim oldtime, time As Long
```

Init:

```
oldtime = Read_Timer()
```

Event:

```
time = Read_Timer()
PAR_1 = time - oldtime
oldtime = time
```

5.2 Verschiedene Tipps

5.2.1 Zugriff auf Hardware-Adressen

Viele Funktionen des *Tico*-Prozessors werden über dessen Steuer- und Datenregister kontrolliert. Diese Funktionen können Sie sehr schnell ausführen lassen, wenn Sie mit den Befehlen **In** und **Out** *direkt* auf die entsprechenden Register zugreifen. Direkt bedeutet, dass Sie im Prozesszyklus die Adressen nicht berechnen, sondern als konstante Werte übergeben: Sie sparen die Berechnungszeit ein.

Die Adressen der Steuer- und Datenregister finden Sie im entsprechenden Hardware-Handbuch.

5.2.2 Konstanten anstelle von Variablen

Eine Berechnung kann deutlich schneller ausgeführt werden, wenn Sie Werte als Konstanten und nicht mit Variablen angeben.

Beispiel

```
PAR_1 = PAR_2 * PAR_2  'mit Par_2=17  
PAR_1 = 17 * 17
```



Für die erste Berechnung muss zur Laufzeit der Wert der Variablen `Par_2` ermittelt, das Quadrat berechnet und `Par_1` zugewiesen werden.

In der zweiten Berechnung kann schon der Compiler den Wert ermitteln. Zur Laufzeit wird der Wert nur noch zugewiesen.

5.2.3 Schnellere Messfunktion

Mit dem Befehl `ADC` wird eine A/D-Wandlung für einen Kanal mit bestimmter Verstärkung vorgenommen. Der Befehl ist sehr einfach gehalten, um Ihnen die Anwendung zu erleichtern, denn er fasst mehrere Ablaufschritte zusammen ([siehe Hardware-Handbuch zum ADwin-System](#)).

Es gibt verschiedene Situationen, in denen Sie mit den einzelnen Ablaufschritten einen schnelleren Ablauf erzielen können als mit dem Befehl `ADC`.

Beispielsweise wird mit dem Befehl `ADC` nicht ausgenutzt, dass sich auf einem *ADwin-Gold II*-System zwei ADC befinden, die gleichzeitig zwei verschiedene Kanäle konvertieren können. Dies geschieht im folgenden Beispiel:

**Beispiel**

Rem Beispiel für Gold II

Event:

```

Rem Multiplexer der ADC auf Kanäle 1 und 2 setzen
Set_Mux1(00000b)
Set_Mux2(00000b)
...
Start_Conv(11b)
Wait_EOC(11b)
PAR_1 = Read_ADC(1)
PAR_2 = Read_ADC(2)

```

'Einschwingzeit abwarten
'Wandlung an beiden ADC
'starten
'Wandlungsende abwarten
'Auslesen von ADC1
'Auslesen von ADC2

5.2.4 Wartezeit genau einstellen

Mit einer Wartezeit kann man leicht einen genauen Zeitabstand zwischen 2 Befehlen einstellen, z.B. um eine feste Bearbeitungszeit eines Hardware-Bausteins zu überbrücken.

Der Befehl **Sleep** stellt die genaue Wartezeit ein: Der Prozessor stoppt für die eingestellte Zeit, so dass der folgende Befehl entsprechend später startet.

5.2.5 Wartezeiten nutzen

Manche Befehle erfordern nach ihrem Aufruf eine bestimmte Wartezeit, ohne dabei den Prozessor zu nutzen. Diese Zeit können Sie für andere Berechnungen nutzen.

Solche Befehle sind **Set_Mux1/2** und **Start_Conv**, nach denen Wartezeiten nötig sind, um das Einschwingen des Multiplexers und die Konvertierung der ADC abzuwarten. Während dieser Wartezeit ist aber der Prozessor nicht beschäftigt, könnte also andere Aufgaben übernehmen.

Genauere Angaben über die erforderlichen Wartezeiten bei der Datenwandlung finden Sie in Ihrem Hardware-Handbuch.

Als praktische Anwendung wird das Beispiel aus dem Abschnitt „**Schnellere Messfunktion**“ nun so erweitert, dass in einem Prozesszyklus 2 Messungen an je 2 ADC durchgeführt werden. Dadurch können Sie im Vergleich zum Befehl ADC in der gleichen Zeit die 4fache Zahl an Messungen durchführen.

Der wesentliche Effekt beruht darauf, dass die einzelnen Schritte der 2 Messungen nicht nacheinander folgen, sondern das Setzen des Multiplexers in die Wartezeit der jeweils anderen Messung verschoben wird. Die Abläufe der beiden Messungen überlagern sich: Auf den Wandlungsstart (Dauer 2µs) für die Kanäle 1+2 folgt das Setzen des Multiplexers (Dauer 2µs) für die Kanäle 3+4.

Beispiel



Rem Beispiel für Gold II
#Include GoldIITiCo.inc

Init:

```
Set_Mux1(000b)      'Mux1 auf Kanal 1 (PGA-Faktor 1)
Set_Mux2(000b)      'Mux2 auf Kanal 2 (PGA-Faktor 1)
Sleep(20)           '2 µs warten
```

Event:

```
Start_Conv(11b)      'Wandlung starten (Kanäle 1+2)
Set_Mux1(001b)      'Mux1 auf Kanal 3
Set_Mux2(001b)      'Mux2 auf Kanal 4
Wait_EOC(11b)       'Wandlungsende abwarten (Kanäle 1+2)
PAR_1 = Read_ADC(1)  'Auslesen von ADC1, Kanal 1
PAR_2 = Read_ADC(2)  'Auslesen von ADC2, Kanal 2

Start_Conv(11b)      'Wandlung starten (Kanäle 3+4)
Set_Mux1(000b)      'Mux1 auf Kanal 1
Set_Mux2(000b)      'Mux2 auf Kanal 2
Wait_EOC(11b)       'Wandlungsende abwarten (Kanäle 3+4)
PAR_3 = Read_ADC(1)  'Auslesen von ADC1, Kanal 3
PAR_4 = Read_ADC(2)  'Auslesen von ADC2, Kanal 4
```

Für die erste Messung im Abschnitt **Event:** ist es erforderlich, den Multiplexer bereits im Abschnitt **Init:** zu setzen, damit der erste Start der Wandlung definiert ist.

Achten Sie streng darauf, dass Sie die Mindest-Wartezeiten für das Einschwingen des Multiplexers und für die Konvertierung der ADC nicht unterschreiten, da sonst die A/D-Wandlung nicht funktioniert und falsche Ergebnisse liefert. Hinweise bietet das [Kapitel 5.2.4 „Wartezeit genau einstellen“](#).



5.2.6 Optimierung des Speicherzugriffs

Der Zugriff auf den externen Speicher ist relativ langsam, insbesondere beim Zugriff auf Einzeldaten. Bei einem Prozess mit niedriger Priorität kann ein Einzelzugriff auf den externen Speicher sogar dazu führen, dass die Reaktionszeit eines Prozesses mit hoher Priorität verlangsamt wird.

Zusätzlich ergibt sich bei jedem Zugriff auf den externen Speicher des *TiCo*-Prozessors eine Wartezeit, die variieren kann („Jitter“). Der Grund ist, dass der *TiCo*-Prozessor davon ausgeht, dass man auf zufällige Speicherstellen zugreift, und deswegen jeden Zugriff neu organisiert – mit entsprechender Wartezeit.

Vermeiden Sie die oben genannten Nachteile, indem Sie die Datenstruktur [RingBuffer](#) für den Zugriff auf Daten im externen Speicher verwenden. Beachten Sie: Die Nachteile werden erst nach der Initialisierung und dem ersten Zugriff mit der Datenstruktur [RingBuffer](#) vermieden.

Informationen zur Anwendung der Datenstruktur [RingBuffer](#) finden Sie in [Kapitel 4.3.3 auf Seite 105](#).

5.3 Debugging

Die Bedienoberfläche beinhaltet als Hilfsmittel zur Fehlersuche den Debug-Modus. Der Debug-Modus wird über das Menü „Debug“ aktiviert (siehe [Kapitel 3.9.6 auf Seite 71](#)) und entfaltet seine Hilfstätigkeit für solche Prozesse, die bei aktiviertem Modus kompiliert werden.



Beachten Sie: Das Aktivieren des Debug-Modus erzeugt zusätzlichen Programm-Code. Dadurch verlängert sich die Ausführungszeit des Programms und der Speicherbedarf wird erhöht – zum Teil beträchtlich. Nutzen Sie das Hilfsmittel daher nur zum Entwickeln und Testen von Programmen.

5.3.1 Laufzeitfehler erkennen (Debug-Modus)

Der Debug-Modus ist ein Hilfsmittel zum Aufspüren folgender Laufzeitfehler in *TiCoBasic*-Programmen:

- Zugriff auf zu große / zu kleine Elementnummern eines Felds

Ohne Debug-Modus werden diese Laufzeitfehler ignoriert, d.h. das Ergebnis der entsprechenden Zeile ist undefiniert, wird aber dennoch für den weiteren Programmablauf verwendet. Dies kann, je nach Programm, unerwünschte Folgen haben, im schlimmsten Fall bis zum Absturz des ADwin-Systems.

Sie aktivieren die Option „Debug mode“ im Menü „Debug“; kompilieren Sie anschließend den zu prüfenden Quelltext. Sobald ein Laufzeitfehler auftritt, wird er automatisch im Fenster „Debug Errors“ (siehe [Option Debug mode](#)) angezeigt. Außerdem wird der Fehler so behandelt, dass ein stabiler Betriebszustand erhalten bleibt.

Gefundene Fehler müssen in jedem Fall bereinigt werden; die automatische Fehlerbehandlung des Debug-Modus ist nur ein Hilfsmittel für die Fehlersuche, nicht für den Dauerbetrieb.



Die Einzelheiten zum Aktivieren und zur Anzeige der Laufzeitfehler sind im Abschnitt „[Option Debug mode](#)“ auf [Seite 72](#) beschrieben.

6 Prozesse im Betriebssystem

Das *ADwin*-System stellt alle Möglichkeiten zur Verfügung, um komplexe Anlagen zu regeln, zu steuern und Messungen durchzuführen. Mit *TiCoBasic* programmieren Sie einen Prozess, der diese Möglichkeiten nutzt: Sie legen darin fest, wie und wann der *Tico*-Prozessor analoge und digitale Daten verarbeitet und nach außen gibt, entweder um der *ADwin* CPU zuzuarbeiten oder auch ganz eigenständig.

Nach dem Start des Prozesses wird das Programm¹ im *Tico*-Prozessor (typischerweise) zyklisch, also in regelmäßigen Abständen neu aufgerufen und abgearbeitet. Ein solcher Aufruf eines Prozesszyklus wird durch eines der folgenden Startsignale, sogenannte „Events“, ausgelöst:

1. Timer-Event: Ein Impuls des internen Zählers. Sie können für jeden Prozess separat festlegen, in welchem Zeitabstand (Processdelay) ein neuer Event ausgelöst wird: das ist ein zeitgesteuerter Prozess.
2. Externer Event: Ein externes Signal, das am „Event“-Eingang Ihres *ADwin*-Systems eingeht. Dies könnte beispielsweise ein Impuls eines Inkrementalgebers sein: das ist ein extern gesteuerter Prozess.
3. Der Prozessstyp None (ohne Event-Signal) wird nur für – meist in Assembler programmierte – Spezialanwendungen benötigt und schließt andere Prozessstypen aus. Wenn nicht anders programmiert, reagiert der Prozess nicht auf Event-Signale und wird nur einmal durchlaufen.

Die exakte Funktion eines Prozesses definieren Sie im *TiCoBasic*-Quelltext:

- Die Initialisierung im Abschnitt **Init:**.
- Die eigentliche Funktion des Prozesszyklus im zentralen Abschnitt **Event:**.
- Die Schlussbearbeitung im Abschnitt **Finish:**.

Von der *ADwin* CPU aus können Sie die Prozesse eines Systems steuern, d.h. Sie können die Prozesse starten, stoppen oder deren Processdelay ändern. Vom PC aus ist dies nur mit der Entwicklungsumgebung *TiCoBasic* möglich. Mit der Bootloader-Option können

1. genauer: der Programmabschnitt **Event:**.

Prozesse außerdem beim Starten der *ADwin*-Hardware automatisch gestartet werden. Näheres zum Programmieren des Bootloaders siehe [Kapitel 3.7.2 „TiCo-Bootloader programmieren“](#), [Seite 50](#).

6.1 Prozessverwaltung

Auf dem *TiCo*-Prozessor sollte nur ein einziger Prozess (mit hoher Priorität) laufen. Je nach Aufgabenstellung wählen Sie dafür einen der folgenden Prozesstypen:

- [Zeitgesteuerter Prozess \(Timer\)](#)

Neben dem hochprioren zeitgesteuerten Prozess ist zusätzlich auch ein niederpriorer zeitgesteuerter Prozess möglich. Der Prozess mit niedriger Priorität kann nicht alleine ausgeführt werden.

- [Extern gesteuerter Prozess \(External\)](#)

Der extern gesteuerte Prozess hat immer hohe Priorität.

- [Prozess ohne Ansteuerung \(None\)](#)

Bei dem Prozess ohne Ansteuerung spielt die Priorität keine Rolle.

Es ist auch möglich, einen zeitgesteuerten und einen extern gesteuerten Prozess miteinander zu kombinieren. Wenden Sie sich in diesem Fall bitte an unseren Support (support@adwin.de), damit wir Sie über die erforderlichen Vorkehrungen informieren können.

Wenn Sie mehrere Prozesse gleichzeitig nutzen wollen, müssen Sie die Quelldateien in ein Projekt einbinden (siehe [Kapitel 3.10.2 auf Seite 76](#)).

6.1.1 Zeitgesteuerter Prozess (Timer)

Beim zeitgesteuerten Prozess startet ein regelmäßiger Impuls des internen Zählers einen Prozesszyklus. Der Zeitabstand zwischen zwei Impulsen wird auch als Zykluszeit (Processdelay) bezeichnet und kann in Schritten zu 10ns oder 20ns eingestellt werden (siehe auch [Kapitel 4.2.7 auf Seite 101](#)).

Neben dem zeitgesteuerten Prozess mit hoher Priorität ist zusätzlich auch ein zeitgesteuerter Prozess mit niedriger Priorität möglich. Weisen Sie einem Prozess seine Priorität über das Menü „Options \ Process Options“ zu.

Der Prozess mit hoher Priorität wird bevorzugt behandelt:

- Signal können in Intervallen von 10ns / 20ns ohne Jitter ausgegeben werden.
- Vom Aufruf des Prozesszyklus durch den internen Zähler bis zur Bearbeitung des ersten Befehls vergehen maximal 120ns, mit *TiCo2* sogar nur 100ns.
- Der hochpriore Prozesszyklus ist nicht unterbrechbar und wird immer vollständig abgearbeitet.

Auch ein Stopp-Befehl kann einen laufenden, hochprioren Prozesszyklus nicht unterbrechen: es muss auf das Ende der Bearbeitung gewartet werden.

Ein Prozesszyklus mit niedriger Priorität wird sofort unterbrochen, wenn ein Prozesszyklus mit hoher Priorität aufgerufen wird und zwar so lange, bis dieser fertig bearbeitet ist.

Programmieren Sie zeitkritische Vorgänge in den Prozess mit hoher Priorität und andere Vorgänge mit niedriger Priorität, damit der Prozessor die zeitkritischen Prozesszyklen ungestört bearbeiten kann.



6.1.2 Extern gesteuerter Prozess (External)

Beim extern gesteuerten Prozess startet ein bestimmtes externes Signal einen Prozesszyklus.

Der Prozess läuft immer mit hoher Priorität:

- Vom Aufruf des Prozesszyklus durch ein externes Signal bis zur Bearbeitung des ersten Befehls vergehen maximal 300ns.
- Der hochpriore Prozesszyklus ist nicht unterbrechbar und wird immer vollständig abgearbeitet.

Auch ein Stopp-Befehl kann einen laufenden, hochprioren Prozesszyklus nicht unterbrechen: es muss auf das Ende der Bearbeitung gewartet werden.

Das aufrufende externe Signal wird sehr flexibel über eine Hardware-Adresse festgelegt: Der Wert der Hardware-Adresse wird mit einer Bitmaske Und-verknüpft und anschließend mit einem festen Wert über einen Operator (<, >, =) verglichen. Wenn der Vergleich wahr ist, wird ein Event-Signal ausgelöst. Zum Einstellen der Werte siehe [Dialogfenster „Process Options“ auf Seite 60](#).

Die Hardware-Adressen sind für jede ADwin-Hardware unterschiedlich.

External Event	
Address:	H/D 70H
Mask:	H/D 12H
Value:	H/D 0
Operation:	> ▼

Beispiel: Die Einstellung oben maskiert den Wert der Adresse 70h mit 12h, so dass nur die Bits 2 und 5 erhalten bleiben. Wenn das Ergebnis > 0 ist (operation und value), wenn also eines der beiden Bits gesetzt ist, wird ein Event-Signal aus gelöst und damit ein Prozesszyklus gestartet. Gesetzt den Fall, die Hardware-Adresse ist ein Register für Digitaleingänge, löst also jedes Setzen eines der beiden – den Bits 2 und 5 zugeordneten – Digitalkanäle einen Prozesszyklus aus.

6.1.3 Prozess ohne Ansteuerung (None)

Der Prozesstyp None (ohne Event-Signal) wird nur für – meist in Assembler programmierte – Spezialanwendungen benötigt und schließt alle anderen Prozesstypen aus. Wir empfehlen die Anwendung nur für sehr erfahrene Nutzer.

Der Prozess reagiert nicht auf Event-Signale, sondern startet, sobald der Prozess auf den TiCo-Prozessor übertragen ist oder durch die Bootloader-Funktion. Das Programm wird nur einmal durchlaufen, es beginnt *nicht automatisch* von vorne.

Um das Programm mehr als einmal zu durchlaufen, können beispielsweise Schleifen eingesetzt werden.

Der Prozesstyp None beeinflusst das Betriebssystem: Prozesse können von außerhalb weder gestartet noch gestoppt werden, noch kann die Zykluszeit von Prozessen verändert werden.

Beachten Sie die Besonderheiten bei der Programmierung:



- Im Programm gibt es keine Abschnitte, die Kennwörter **Init:**, **Event:** und **Finish:** sind daher ungültig und erzeugen eine Fehlermeldung.
- Der Befehl **End** hat keine Funktion.
- Programmieren Sie an das Ende des Programms eine Endlosschleife. Anderenfalls können unvorhergesehene Probleme auftreten. Eine Endlosschleife kann so aussehen:

```
Do  
Until (1 = 2)
```

Die Programmierung mit Assembler ist in einem separaten Handbuch beschrieben.

6.2 Zeitverhalten von Prozessen

6.2.1 Processdelay

Der Zeitabstand, in dem *zeitgesteuerte* Prozesszyklen vom Zähler aufgerufen werden, ist die Zykluszeit des Abschnitts **Event:**. Die Zykluszeit wird in Taktzyklen des Zählers gemessen und dann als *Processdelay* bezeichnet. Sie können das Processdelay jedes Prozesses über den Wert der Systemvariablen **Processdelay** festlegen (siehe auch [Seite 199](#)).

Auch der Zeitabstand zwischen dem Ende des letzten Befehls im Abschnitt **Init:** und dem Beginn des Abschnitts **Event:** beträgt ein *Processdelay* (plus einige wenige Taktzyklen).

Die Dauer eines Zähler-Taktzyklus ist abhängig vom Prozessortyp:

Prozessor	Taktzyklus
TiCo1	20ns
TiCo2	10ns

Abb. 16 – Dauer eines Zähler-Taktzyklus (Einheit des Processdelay)

Ein Processdelay mit dem Wert 1250 beispielsweise bedeutet beim Prozessor TiCo1 einen regelmäßigen Aufruf im Zeitabstand von

$1250 \times 20\text{ns} = 25\mu\text{s}$. Dies kann z.B. eingestellt werden mit der Programmzeile:

```
Processdelay = 1250
```

Damit jeder (zeitgesteuerte) Prozesszyklus zu der (mit **Processdelay**) festgelegten Zeit aufgerufen wird, darf die Bearbeitungszeit eines Prozesszyklus die Zykluszeit auch im ungünstigsten Fall nicht überschreiten. Unterschiede bei der Berechnungszeit ergeben sich beispielsweise bei fallweisen Unterscheidungen (If, Case).

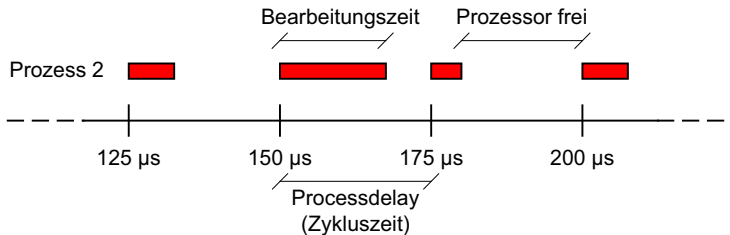


Abb. 17 – Processdelay und Bearbeitungszeit



Beispiel

Wenn eine umfangreiche Berechnung nur alle 1000 Messungen auftritt, dann muss auch die lange Bearbeitungszeit dieses Prozesszyklus kürzer sein als die Zykluszeit. Um dennoch kurze Prozesszyklen zu erreichen, ist es eine gute Alternative, die Berechnung in kleine Schritte aufzuteilen und in jedem Prozesszyklus jeweils einen Schritt zu bearbeiten. Die Prozesszyklen erhalten dadurch eine durchweg gleichmäßige und kurze Bearbeitungszeit.

6.2.2 Auslastung des TiCo-Prozessors

Die Auslastung des *TiCo*-Prozessors ist das Verhältnis von genutzter Rechenzeit zur insgesamt verfügbaren Rechenzeit, angegeben in Prozent.

Sie können die Auslastung des Prozessors an der Anzeige „Busy“ in der Statusleiste der Entwicklungsumgebung beobachten (siehe [Kapitel 3.10.6](#)). Dieser Wert gibt Ihnen einen Anhaltspunkt, ob der Prozessor noch genügend Rechenzeit frei hat, um alle Prozesszyklen abarbeiten zu können.

Die Auslastung des Prozessors sollte 90% nur in Ausnahmefällen überschreiten, den Wert 100% jedoch niemals.

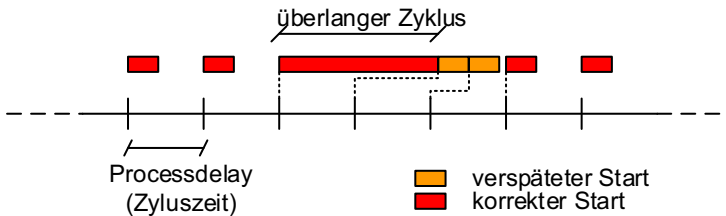
6.2.3 Verschiedene Betriebszustände im Betriebssystem

Das Betriebssystem behandelt den zeitgesteuerten und den extern gesteuerten Prozess beim Zeitverhalten unterschiedlich: Bei dem extern gesteuerten Prozess können Event-Signale verloren gehen, bei zeitgesteuerten Prozessen in der Regel nicht.

Zeitgesteuerter Prozess

Bei einem zeitgesteuerten Prozess wird in der Regel jeder Prozesszyklus zu der (mit **Processdelay**, Seite 131) festgelegten Zeit aufgerufen. Manchmal ist dies nicht möglich, z.B. weil ein Prozesszyklus länger dauerte als die Zykluszeit; dann laufen Event-Signale des internen Zählers auf.

Das Betriebssystem holt aufgelaufene Event-Signale nach, d.h. Prozesszyklen werden nacheinander ohne Pause aufgerufen, bis das ursprüngliche Zeitraster wieder erreicht ist. Beim niederpriorigen Prozess gilt dies auch, soweit der hochpriorige Prozess nicht aktiv ist.



Wenn aufgelaufene Event-Signale länger als 2^{31} Takte – bei TiCo1: 42,9 Sekunden, bei TiCo2 21,5 Sekunden – auf die Abarbeitung warten müssen, werden die Event-Signale nicht mehr nachgeholt. Wenn eine derart große Verzögerung auftritt, müssen Sie das Zeitverhalten des Prozesses überprüfen: Es ist wahrscheinlich, dass der Prozesszyklus regelmäßig länger dauert als die Zykluszeit. In diesem Fall vergrößern Sie die Zykluszeit oder verkürzen die Bearbeitungszeit für den Prozesszyklus durch geeignete Programmierung.



Extern gesteuerter Prozess

Bei dem externen Prozess werden eintreffende Event-Signale sehr schnell bearbeitet, jedoch können auch Event-Signale verloren gehen.

Das Betriebssystem verwendet ein Hardware-Register, um externe Event-Signale zu verarbeiten. Ist ein Event-Signal eingetroffen, startet das Betriebssystem sofort einen Prozesszyklus, wenn nicht gerade ein hochpriorer Prozesszyklus bearbeitet wird. In diesem Fall dient das Register als Zwischenspeicher für das Event-Signal, und das Betriebssystem startet den nächsten Prozesszyklus sofort nach dem aktuell bearbeiteten Prozesszyklus.



Wenn mehrere Event-Signale während eines Prozesszyklus eintreffen, werden anschließend nicht entsprechend viele Prozesszyklen aufgerufen, sondern nur ein einziger; es gehen also Event-Signale verloren.

Ein externes Event-Signal ist eine besonders wichtige Information – schon weil sie vom *ADwin*-System nicht vorherbestimmbar ist – und darf auf keinen Fall verloren gehen. Achten Sie deshalb in diesem Prozess ganz besonders auf kurze Prozesszyklen (im Abschnitt **Event:**).

6.3 Kommunikation

6.3.1 Datenaustausch zwischen Prozessen

Sie können Daten zwischen *TiCoBasic*-Prozessen über globale Variablen (`Par_1 ... Par_80`) oder über globale Felder (`Data_n`) austauschen.



Wenn Sie globale Felder in mehreren Prozessen verwenden, müssen Sie diese in jedem Prozess in absolut gleicher Weise deklarieren. In diesem Fall ist es praktisch, wenn Sie die Deklaration der globalen Felder in einer Include-Datei speichern und diese in allen Prozessen einbinden (siehe auch [Kapitel 4.5.2 „Include-Dateien“](#)).

Dies gilt nicht für [Die Datenstruktur RingBuffer](#) (siehe [Kapitel 4.3.3 auf Seite 105](#)); hier darf die Deklaration innerhalb eines Projekts nur ein einziges Mal vorkommen.

Je nach Programmierung können Sie (jeweils gleiche) globale Variablen verwenden, um aus einem Prozess heraus einen anderen, gleichzeitig laufenden Prozess zu steuern.

Beispiel



Prozess 1 arbeitet als Funktionsgenerator und Prozess 2 als Regler. Der Funktionsgenerator schreibt regelmäßig den jeweils erzeugten Wert in die globale Variable `Par_10`. Der Regler liest bei jedem Prozesszyklus diese globale Variable `Par_10` aus und verwendet deren Inhalt als Sollwert des Regelkreises.

Damit steuert der Funktionsgenerator auf einfache Weise den Sollwertverlauf des Reglers. Alle *lokalen* Variablen und Felder des Prozesses 1 bleiben hierbei dem Prozess 2 verborgen (und umgekehrt). Beachten Sie bitte, dass bei der Zusammenarbeit der beiden Prozesse auch deren Zeitverhalten von Bedeutung ist.

6.3.2 Kommunikation zwischen PC und TiCo-Prozessor

Vom PC aus haben Sie keinen direkten Zugriff auf den *TiCo*-Prozessor, der Zugriff ist nur von der *ADwin* CPU aus möglich. Die *ADwin* CPU kann Prozesse im *TiCo*-Prozessor steuern, Daten von dort lesen oder dorthin schreiben (siehe [Kapitel 6.3.3 auf Seite 136](#)). Der *TiCo*-Prozessor selbst kommuniziert nicht aktiv.

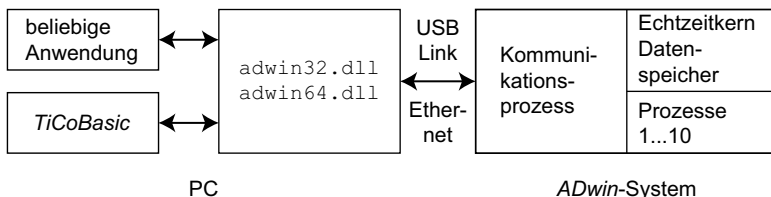
Um Daten zwischen PC und *TiCo*-Prozessor austauschen zu können, muss die *ADwin*-CPU als Zwischenstation eingerichtet werden. Hierbei überträgt ein Prozess – den Sie selbst erstellen müssen – auf der *ADwin*-CPU die Daten und Steuersignale. Auf diese Weise arbeitet auch der Datenfluss bei der Entwicklungsumgebung *TiCoBasic*.

Daten werden immer über globale Variablen (`Par_n`, `FPar_n`) oder globale Felder (`Data_n`) ausgetauscht (auch beim [Datenaustausch zwischen Prozessen](#), siehe oben).

Kommunikation zwischen PC und ADwin CPU

Die Kommunikation zur *ADwin* CPU läuft unter Windows über die sogenannte „*ADwin32.dll*“ bzw. „*ADwin64.dll*“ (dynamic-link library), in der *ADwin* CPU übernimmt diese Aufgabe ein Kommunikationsprozess.

Wenn Sie mit der ActiveX-Schnittstelle arbeiten, übernimmt diese – auf den ersten Blick ähnlich zentral – jede Kommunikation mit der *ADwin* CPU. Intern gibt die ActiveX-Schnittstelle die kommunizierten Daten weiter an die „ADwin32/64.dll“ oder erhält sie von dieser.



Die „ADwin32/64.dll“ übernimmt folgende Aufgaben:

- Kommunikation mit dem angesprochenen *ADwin*-System über Ethernet (TCP/IP).
- Erkennen und Behandeln von Fehlern.
- Verriegeln mehrerer PC-Anwendungen gegeneinander, wenn diese gleichzeitig auf dasselbe System zugreifen möchten.

Durch die Verriegelung können mehrere Anwendungen unabhängig voneinander und quasi gleichzeitig auf ein oder mehrere *ADwin*-Systeme zugreifen.

Wenn eine PC-Anwendung die Kommunikation zu einem bestimmten System aufnimmt, übergibt es neben der gewünschten Anweisung auch eine Gerätenummer, die sogenannte „Device No“. Anhand dieser „Device No“ unterscheidet die „ADwin32/64.dll“ die verschiedenen *ADwin*-Systeme und ordnet die entsprechenden Einstellungen zu.

6.3.3 Kommunikation zwischen ADwin CPU und TiCo-Prozessor

Die *ADwin* CPU kann Prozesse im *TiCo*-Prozessor steuern, Daten von dort lesen oder dorthin schreiben. Der *TiCo*-Prozessor selbst kommuniziert nicht aktiv.

Mit *ADbasic*-Befehlen kann die *ADwin* CPU auf den *TiCo*-Prozessor zugreifen und folgende Aktionen ausführen:

- Datenzugriff Initialisieren.
- Globale Variablen *Par_1*...*Par_80* lesen und schreiben.
- Globale Felder *Data_1*...*Data_16* lesen und schreiben.
- Ringspeicher lesen und schreiben sowie Zustand abfragen.
- Processdelay eines *TiCo*-Prozesses einstellen und abfragen.
- *TiCo*-Prozesse starten und stoppen.
- Prozessor starten, stoppen und zurücksetzen.
- Systeminformation abfragen.
- Binärdatei übertragen.

Eine detaillierte Beschreibung der Befehle finden Sie [hier](#):

- *ADwin-Gold II*: [Kapitel 7.4 auf Seite 229](#).
- *ADwin-Pro II*: [Kapitel 7.5 auf Seite 275](#).

6.3.4 Die Device No

Jedes *ADwin*-System, das an einen PC angeschlossen ist, wird über eine (in diesem PC) eindeutige Gerätenummer, die *Device No* angesprochen.

Sie stellen die Device-No. mit dem Programm *ADconfig* ein: Programs ▶ *ADwin* ▶ *ADconfig*.

In *ADconfig* verknüpfen Sie eine „Device No“ mit den Verbindungsparametern, mit denen ein System erreichbar ist (über TCP/IP). Auf diese Informationen greift die „*ADwin32.dll*“ zurück, um mit dem System kommunizieren zu können.

7 Befehlsreferenz

Im folgenden sind die in *TiCoBasic* verfügbaren Befehle für *TiCo*-Prozessoren aufgeführt. Befehle zur Steuerung der Ein- und Ausgänge finden Sie in der Hardware-Dokumentation.

Die Befehle sind alphabetisch sortiert aufgeführt. Im Anhang gibt es eine Befehlsübersicht.

In [Kapitel 7.4](#) und [Kapitel 7.5](#) sind die *ADbasic*-Befehle aufgeführt, mit denen die *ADwin* CPU auf den *TiCo*-Prozessor zugreifen kann; die Befehle sind für *ADwin-Gold II* und für *ADwin-Pro II* separat aufgeführt.

7.1 Befehlssyntax

Beachten Sie bitte:

- Als Argument ist ein beliebiger [Berechnungsausdruck möglich](#).
- Bei einigen Argumenten ist die Datenstruktur vorgeschrieben, die wie folgt gekennzeichnet ist:

CONST Konstante Zahlen wie [35](#) sowie Berechnungsausdrücke ohne Variablen.

VAR Variable oder Feldelement.

ARRAY Feld, in der Syntaxzeile auch erkennbar an den Klammern [] nach dem Feldnamen.

- Neben einem Argument oder dem Rückgabewert einer Funktion ist der erwartete Datentyp angegeben:

LONG ganze Zahl

LOGIC logischer Ausdruck in einer Bedingung

- Manche Befehle können nur benutzt werden, wenn eine bestimmte Library- oder Include-Datei eingebunden wird. Unter **Syntax** ist der jeweilige Befehl zum Einbinden angegeben (setzen Sie diese Befehlszeile bitte an den Anfang des Quelltextes).

Es wird davon ausgegangen, dass die erforderliche Library- oder Include-Datei in dem Verzeichnis liegt, das unter „Options ▶ Set -

tings“ unter dem Reiter „Directory“ eingestellt ist (siehe auch die Befehle `#Include` oder `Import`).

7.2 Basis-Befehlssatz *TiCoBasic*

Die Befehle in diesem Abschnitt sind für alle *TiCo*-Prozessoren gültig.

+ (Addition)

Der Operator „+“ addiert je zwei Werte.

Syntax

```
val = val_1 + val_2
```

Parameter

val_1 Summand 1

LONG

val_2 Summand 2

LONG

Bemerkungen

- / -

Siehe auch

- (Subtraktion), * (Multiplikation), / (Division), ^ (Potenz)

Beispiel

```
Par_1 = 9 + 4                      'Par_1 = 13
```

- (Subtraktion)

Der Operator „-“ subtrahiert je zwei Werte.

Syntax

```
val = val_1 - val_2
```

Parameter

val_1	Minuend
val_2	Subtrahend

LONG

LONG

Bemerkungen

- / -

Siehe auch

+ (Addition), * (Multiplikation), / (Division), ^ (Potenz)

Beispiel

```
Par_1 = 9 - 4                    'Par_1 = 5
```

* (Multiplikation)

Der Operator „*“ multipliziert je zwei Werte.

Syntax

```
val = val_1 * val_2
```

Parameter

val_1 Multiplikator 1

LONG

val_2 Multiplikator 2

LONG

Bemerkungen

- / -

Siehe auch

+ (Addition), - (Subtraktion), / (Division), ^ (Potenz)

Beispiel

```
Par_1 = 9 * 4                    'Par_1 = 36
```

/ (Division)

Der Operator „/“ dividiert je zwei Werte.

Syntax

```
val = val_1 / val_2
```

Parameter

val_1 Dividend

LONG

val_2 Divisor

LONG

Bemerkungen

Beachten Sie, dass eine Division immer ohne Rest durchgeführt wird.

Wenn Sie durch eine Variable mit negativem Vorzeichen teilen, müssen Sie diese in Klammern setzen, damit das erwartete Ergebnis berechnet wird (siehe auch [Kapitel 4.4.1 „Auswertung von Operatoren“](#)).

Siehe auch

[+ \(Addition\)](#), [- \(Subtraktion\)](#), [* \(Multiplikation\)](#), [^ \(Potenz\)](#)

Beispiel

```
Par_1 = 36 / 4                      'Par_1 = 9
REM Par_2: ganzzahlige Rechnung
Par_2 = 2 / 4 * 5                  'Par_2 = 0
Par_3 = 27 / (-Par_1)              'Par_3 = -3
REM Beachten Sie die Klammersetzung in der letzten Zeile
```


^ (Potenz)

Der Operator „^“ berechnet eine beliebige Potenz eines Wertes.

Syntax

```
val = val_1 ^ val_2
```

Parameter

val_1 Basis

LONG

val_2 Exponent

LONG

Bemerkungen

Wenn die Basis und/oder der Exponent eine Variable mit negativem Vorzeichen ist, müssen Sie diese in Klammern setzen, damit das Vorzeichen bei der Potenzierung berücksichtigt wird (siehe auch [Kapitel 4.4.1 „Auswertung von Operatoren“](#)). Bei Konstanten ist dies nicht der Fall.

```
var1 = -2^2           'var1 = 4
var2 = -var1^2        'var2 = -16
var3 = (-var1)^2      'var3 = 16
```

Polynome werden schneller berechnet, wenn Sie die Potenzen mittels Ausklammerung durch wenige Multiplikationen ersetzen:

```
y = a + b*x + c*x^2 + d*x^3 + e*x^4 'langsame Variante
y = a + x*(b + x*(c + x*(d + x*e))) 'schnelle Variante
```

Siehe auch

+ (Addition), - (Subtraktion), * (Multiplikation), / (Division)

Beispiel

```
Par_1 = 9 ^ 4           'Par_1 = 6561
```

#... (Compiler-Anweisung)

Ein *TiCoBasic*-Befehl, der mit dem Zeichen „#“ beginnt, ist eine Anweisung für den Compiler, den Quelltext vor dem Erzeugen des Binärcodes auf eine bestimmte Weise zu bearbeiten.

Folgende Compiler-Anweisungen stehen Ihnen zur Verfügung:

#Define	Definition symbolischer Konstanten: Zeichenfolgen im Quelltext werden durch andere Zeichenfolgen ersetzt.
#Include	Datei einfügen: Eine Datei (mit Quelltext) wird in den Quelltext eingefügt.
#If...#EndIf	Bedingte Kompilierung: Bei erfüllter Bedingung werden die entsprechenden Quelltextzeilen kompiliert, anderenfalls gelöscht
#Begin_ Debug_ Mode_ Disable	Unterbricht in einem laufenden Prozess den Debug-Modus.
#End_ Debug_ Mode_ Disable	Hebt die Unterbrechung des Debug-Modus wieder auf.

: (Doppelpunkt)

Das Zeichen „:“ trennt Programmschritte innerhalb einer einzelnen Programmzeile.

Syntax

```
[Schritt_1] : [Schritt_2] { : [Schritt_3] ... }
```

Bemerkungen

[Schritt_n] bezeichnet einen beliebigen Programmschritt, wie er sonst in einer einzelnen Programmzeile angegeben wird.

Eine Programmzeile darf nicht mehr als 2048 Zeichen beinhalten.

Verwenden Sie den Befehl nur, wenn dadurch der Quelltext übersichtlicher wird.

Beispiel

```
Inc Par_1 : Inc Par_2  
Rem Par_1 und Par_2 erhöhen in *einer* Zeile
```

= (Zuweisung)

Der Operator „=“ weist der Variablen oder dem Feldelement links vom Operator das Ergebnis des Ausdrucks rechts vom Operator zu.

Syntax

```
var = expr
```

Parameter

`var` Variable oder Feld

VAR

LONG

`expr` Berechnungsausdruck

LONG

Bemerkungen

- / -

Beispiel

```
Dim val_1, val_2 As Long 'Deklaration
```

Init:

```
val_1 = 69 'Zuweisung einer  
           'Konstanten
```

Event:

```
val_2 = val_1 * 2 'Zuweisung eines  
                  'Ausdrucks
```

< = > (Vergleich)

Die Operatoren „<“, „=“ und „>“ dienen zum Vergleich zweier Werte. In *TiCoBasic* kommen diese Operatoren nur in Bedingungen vor.

Syntax

```
If (val_1 > val_2) Then
```

Parameter

val_1 Operand

LONG

val_2 Operand

LONG

Bemerkungen

Folgende Vergleiche sind möglich:

Operator	Bedeutung
<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich
=	gleich
<>	ungleich

Siehe auch

```
If ... Then ... {Else ... } EndIf, #If ... Then ... {#Else ... } #EndIf
```

Beispiel

```
Dim value As Long
```

```
Event:
```

```
value = -5
```

```
If (value < 0) Then value = 0
```

```
Rem Ergebnis: value = 0
```

AbsI

AbsI liefert den Betrag einer Long-Variablen.

Syntax

```
ret_val = AbsI(value)
```

Parameter

value Argument: $-(2^{31}-1) \dots +2^{31}-1$.

LONG

ret_val Betrag des Arguments ($0 \dots +2^{31}-1$).

LONG

Bemerkungen

Für den kleinsten negativen, ganzzahligen Wert -2^{31} gibt es in *TiCo-Basic* keine positive Entsprechung; der Betrag dieses Werts ist daher undefiniert.

Siehe auch

[Mod](#)

Beispiel

```
Dim val_1, val_2 As Long
```

```
Event:
```

```
val_1 = -5
```

```
val_2 = AbsI(val_1) 'Ergebnis: val_2 = 5
```

And

Der Operator **And** verknüpft zwei ganzzahlige Werte bitweise oder zwei Boolesche Ausdrücke als Boolescher Operator.

Syntax

```
ret_val = val_1 And val_2
        'Bitweiser Operator

If ((expr1) And (expr2)) Then
        'Boolescher Operator
```

Parameter

val_1, val_2	Ganzzahliger Wert	LONG
expr1, expr2	Boolescher Ausdruck mit dem Wert „wahr“ oder „falsch“	LOGIC

Bemerkungen

Sie können mit **And** nur gleichartige Ausdrücke verknüpfen (ganzzahlige *oder* Boolesche), ein Mischen ist nicht möglich.

Sie können Boolesche Ausdrücke nur mit den Anweisungen **If ... Then ... Else** oder **Do ... Until** verwenden (Variablen können keine Booleschen Werte annehmen).

Wenn Sie in einer Zeile mehrere Boolesche Operatoren verwenden, müssen Sie jede Verknüpfung separat in Klammern setzen. Bei der Verknüpfung ganzzahliger Werte ist dies nicht erforderlich.

Siehe auch

Not, Or, XOr

Beispiel

Rem Bitweise Verknüpfung von Long-Variablen

```
Dim val_1, val_2, val3 As Long
```

```
val_1 = 0100b           '= 4
```

```
val_2 = 0110b           '= 6
```

```
val3 = val_1 And val_2 'Bitweise Verknüpfung
```

Rem Ergebnis: val3 = 0100b = 4

Rem ODER:

Rem Boolesche Verknüpfung von Booleschen Ausdrücken

```
Dim val_1, val4 As Long
```

```
val_1 = 314
```

Rem Boolesche Verknüpfung: (wahr) And (wahr) = wahr

```
If ((val_1 < 910) And (val_1 > 310)) Then
```

```
    val4 = 1
```

```
Else
```

```
    val4 = 0
```

```
EndIf                                     'Ergebnis: val4 = 1
```


#Begin_Debug_Mode_Disable

#Begin_Debug_Mode_Disable verhindert bei eingeschaltetem Debug-Modus das Erzeugen von Debug-Informationen.

Syntax

```
#Begin_Debug_Mode_Disable
```

Parameter

- / -

Bemerkungen

#Begin_Debug_Mode_Disable ist eine Präprozessor-Anweisung und hat nur eine Funktion, wenn der Prozess mit der [Option Debug mode](#) kompiliert wird (siehe [Seite 72](#)). Der Debug-Modus dient dazu, Laufzeitfehler zu erkennen, siehe [Kapitel 5.3.1 auf Seite 124](#).

Die Unterbrechung des Debug-Modus wird mit **#End_Debug_Mode_Disable** wieder aufgehoben.

Sie können Unterbrechungen des Debug-Modus nicht verschachteln, d.h. jedes **#End_Debug_Mode_Disable** hebt die Unterbrechung in jedem Fall wieder auf. Achten Sie dabei besonders auf Unterbrechungen in Makros oder Bibliotheken.

Siehe auch

[#End_Debug_Mode_Disable](#), [#Include](#)

Beispiel

Event:

```
#Begin_Debug_Mode_Disable  
Rem unkritischer Quellcode, der nicht überwacht wird  
Rem ...  
#End_Debug_Mode_Disable  
Rem kritischer Quellcode mit Überwachung auf  
Rem Laufzeitfehler  
Rem ...
```

Data_n

Mit `Dim Data_n [...]` `As ...` wird ein globales Data-Feld dimensioniert. Weitere Informationen zur Dimensionierung siehe [Dim](#) auf [Seite 158](#).

Syntax

```
Dim Data_n[dim1] As Long
    {At <mem_type>}
```

Parameter

Data_n	Name des deklarierten Data-Felds mit n : 1...16.	
dim1, dim2	Feldgröße: Anzahl (1...2 ³¹) der Elemente vom Typ <code>arr_type</code> im Feld.	CONST LONG
<mem_type>	Speicher, in dem die Variablen abgelegt werden: <code>DRAM_Extern</code> : externer Datenspeicher. <code>SRAM_Extern</code> : externer SRAM-Speicher bei Pro II-Modulen (anstelle des DRAM). <code>DM_Local</code> :interner Datenspeicher (Default).	

Bemerkungen

Bei einem Feld können Sie auf die Elemente 1...**dim** zugreifen. Das Feldelement [0] dürfen Sie nicht verwenden, denn es wird für interne Zwecke benutzt.

Die maximale Feldgröße ist abhängig vom verfügbaren physikalischen Speicher auf dem *TiCo*-Prozessor.

Siehe auch

[Dim](#), [RingBuffer](#), „Globale Felder (Arrays)“ auf [Seite 99](#), „Variablen und Felder im Datenspeicher“ auf [Seite 103](#)

Beispiel

```
Rem Dimensioniere das globale Feld Data_15 mit
Rem 1000 Long-Elementen
Dim Data_15[1000] As Long
```

Dec

Dec verringert den Wert einer Long-Variablen um 1.

Syntax

Dec (*var*)

Parameter

var

Name einer lokalen oder globalen Long-Variablen

VAR

CONST

LONG

Bemerkungen

Die Anweisung **Dec** (*var*) führt zum gleichen Ergebnis wie die Programmzeile: *val=val-1*. Außerdem kann die Anweisung **Dec** eine geringere Ausführungszeit haben.

Siehe auch

[Inc, - \(Subtraktion\)](#)

Beispiel

```
Dim index As Long
Dim Data_1[1000] As Long

Init:
index=1000

Event:
  DAC(1,Data_1[index]) 'Wert auf DAC1 ausgeben
  Dec(index)           'index um 1 verringern
  If (index<1) Then
    index=1000         'Nach 1000 Ausgaben von
                        'vorne beginnen
  EndIf
```

Die Konstante kann abhängig von **#If PRIORITAET** definiert werden.

#Define

#Define ersetzt im Quelltext einen symbolischen Namen durch einen frei definierbaren Ausdruck, z.B. eine Konstante.

Syntax

```
#Define name expression
```

Parameter

name	Symbolischer Name, <i>ohne</i> Hochkommata. Sonderzeichen sind nicht erlaubt, nur alphanumerische Zeichen (a...z, A...Z, 0...9) und der Unterstrich (_).	CONST STRING
expression n	Ausdruck, für den der symbolische Name steht; <i>ohne</i> Hochkommata. Alle Zeichen sind erlaubt.	CONST STRING

Bemerkungen

Stellen Sie diese Anweisung an den Beginn eines Quelltextes. Die Ersetzung wird im Quelltext ab der darauf folgenden Zeile durchgeführt.

Die Funktion **#Define** ist ein Präprozessor-Befehl, d.h. die Ersetzung findet statt, wenn Sie den Quelltext kompilieren lassen (noch bevor der Compiler das lauffähige Programm erzeugt). Verwenden Sie die Funktion, um im Quelltext aussagekräftige Namen anstelle von Konstanten, Parametern oder Berechnungsausdrücken zu verwenden.

Die erste Zeichenfolge bis zu einem Leerzeichen wird als symbolischer Name interpretiert, der nachfolgende Zeileninhalt bis zum Zeilenumbruch als einzufügender Ausdruck¹. Der Ausdruck wird exakt so eingefügt, wie Sie ihn definiert haben; Variablenamen im Ausdruck werden also nicht durch deren aktuellen Wert, sondern als Zeichenfolge ersetzt.

Groß- und Kleinschreibung wird beim Suchen und Ersetzen nicht unterschieden.

1. Text hinter einem Kommentarzeichen „*/**“ wird vom Compiler ignoriert.

Wenn Sie für **expression** einen Rechenausdruck einsetzen, empfehlen wir, diesen mit Klammern zu umgeben. Sie vermeiden damit eventuelle Fehler im Zusammenhang mit weiteren Rechenausdrücken.

Siehe auch

[#Include](#)

Beispiel

```
#Define setpoint Par_1 'Kommentar, wird nicht ersetzt  
#Define measured Data_1  
#Define const 12441223
```

Mit diesen Anweisungen können Sie im Quelltext anstelle von **Par_1**, **Data_1** und der Ziffernfolge die Namen **setpoint**, **measured** und **const** verwenden.

```
#Define Sollwert (13 + 4^3)  
Par_1 = 2 * Sollwert      '= 2 * (13 + 4^3)
```

Ohne die Klammern im **#Define**-Ausdruck würden Sie statt des erwarteten Ergebnisses „154“ den Wert „90“ erhalten.

Dim

Dim deklariert ein oder mehrere

- *lokale* Variablen
- *lokale* eindimensionale Felder
- *globale* eindimensionale Felder **Data_n**[n] (auch RingBuffer-Felder)

Data_n[n] [m] Grundlagen zu Variablen und Datentypen finden Sie in [Kapitel 4.2.3 auf Seite 97](#) sowie Informationen zu RingBuffer-Feldern unter dem Stichwort [RingBuffer](#) auf [Seite 206](#).

Syntax

```
Dim var1 {, var2, ...} As Long
```

```
Dim array1[dim1] As Long
```

```
{At <mem_type>}
```

```
Dim Data_n[dim1] As Long
```

```
{As RingBuffer_For_Read /  
RingBuffer_For_Write} {At <mem_type>}
```

Parameter

<code>var1, var2</code>	Namen der deklarierten Variablen	
<code>array1,</code> <code>array2,</code> <code>Data_n</code>	Namen der deklarierten Felder. Für <code>Data_n</code> kann <code>n</code> aus 1...16 gewählt werden.	
<code>dim1, dim2</code>	Feldgröße: Anzahl ($1 \dots 2^{31}$) der Feldelemente vom Typ <code>Long</code> .	CONST <u>LONG</u>
<code><mem_type></code>	Speicherbereich, in dem die Variablen abgelegt werden: <code>DRAM_Extern</code> : externer Datenspeicher <code>SRAM_Extern</code> : externer SRAM-Speicher bei Pro II-Modulen (anstelle von DRAM) <code>DM_Local</code> : interner Datenspeicher (Default)	

Bemerkungen

Die globalen Variablen `Par_n` dürfen nicht deklariert werden, weil sie vordefiniert sind.

Wenn Sie von der ADwin CPU oder aus mehreren Prozessen auf Daten zugreifen wollen, ist dies nur über *globale* Variablen und Felder möglich.

Die Datenstruktur `RingBuffer` ist geeignet, um große Datenmengen kontinuierlich und schnell zu übertragen, zwischen ADwin CPU und Ti-Co-Prozessor, zwischen TiCo-Prozessor und externem Speicher oder zwischen zwei TiCo-Prozessen.

Der Umgang mit Ringspeichern ist nicht einfach. Bei falscher Anwendung können schwer auffindbare Fehler entstehen. Die Verwendung des Datentyps `RingBuffer` ist daher erfahrenen Benutzern von *Ti-CoBasic* und *ADbasic* vorbehalten.

Bitte beachten Sie die Hinweise in [Kapitel 4.3.3 auf Seite 105](#).

Bei einem Feld können Sie auf die Elemente $1 \dots \text{dim}$ zugreifen. Das Feldelement [0] dürfen Sie nicht verwenden, weil es für interne Zwecke benutzt wird.

Die maximale Feldgröße ist abhängig vom verfügbaren physikalischen Speicher auf dem TiCo-Prozessor.

Hinweise zum Umgang mit den verschiedenen Speicherbereichen `<mem_type>` finden Sie über die unten angegebenen Querverweise.



Siehe auch

[Data_n](#), [Event](#)., [RingBuffer](#), [Finish](#)., [Init](#)., „Variablen und Felder“ auf Seite 96, „Variablen und Felder im Datenspeicher“ auf Seite 103, „Speicherbereiche“ auf Seite 103

Beispiel

```
Rem Dimensioniere var1 als Long-Variable  
Dim var1 As Long
```

```
Rem Dimensioniere das Feld array1 mit  
Rem 1000 Long-Elementen  
Dim array1[1000] As Long
```

```
Rem Dimensioniere das globale Feld Data_15 mit  
Rem 1007 Long-Elementen als Lese-Ringspeicher  
Dim Data_15[1007] As Long As RingBuffer_For_Read
```


Do ... Until

Do...Until definiert eine Schleife, deren Anweisungsblock mindestens einmal durchlaufen werden. Die Schleife wird abgebrochen, wenn die Abbruchbedingung den Wert „Wahr“ hat.

Syntax

```
Do
    ... 'Anweisungsblock
Until (condition)
```

Parameter

condition Boolesche Abbruchbedingung mit den Operatoren <, >, =, **And** und **Or**.

LOGIC |

Siehe auch

< = > (Vergleich), **And**, **Or**, **For ... To ... {Step ... } Next**, **SelectCase**

Bemerkungen

Sie können **Do...Until**-Schleifen beliebig tief verschachteln.

Vermeiden Sie Schleifen mit langer Ausführungszeit in hochprioren Prozessen, weil diese nicht unterbrochen werden können.

Beispiel

```
Dim count As Long
Dim Data_1[103] As Long As RingBuffer_For_Write

Init:
    count = 1

Event:
    Do 'Schleife beginnen
        Data_1 = ADC(1,4) 'Messwert auslesen
        Inc count 'Zählvariable erhöhen
    Until (count > 100) '100 Messungen durchgeführt?
```

End

End beendet einen Prozess

Syntax

End

Bemerkungen

Der Befehl End beendet die Ausführung des Abschnitts sofort. End ist in allen Programmabschnitten gültig.

Wenn der Befehl im Abschnitt **Event:** benutzt wird, beginnt danach die Ausführung des Abschnitts **Finish:** (sofern vorhanden). Wenn im Abschnitt **Event:** nach der Anweisung End weitere Programmzeilen folgen, werden sie nicht mehr ausgeführt.

Siehe auch

ProcessN_Running,

Beispiel

```
Event:
  If (ADC(1) > 3000) Then 'Messen und Vergleichen
    End 'Prozess beenden, aber Finish: noch ausführen
  EndIf

Finish:
  Set_Digout(1) 'Dig. Ausgang 1 setzen
```

#End_Debug_Mode_Disable

#End_Debug_Mode_Disable hebt die Unterbrechung des Debug-Modus wieder auf.

Syntax

```
#End_Debug_Mode_Disable
```

Parameter

- / -

Bemerkungen

#End_Debug_Mode_Disable ist eine Präprozessor-Anweisung und hat nur eine Funktion, wenn der Prozess mit der [Option Debug mode](#) kompiliert wurde (siehe [Seite 72](#)) und der Debug-Modus mit **#Begin_Debug_Mode_Disable** unterbrochen wurde. Der Debug-Modus dient dazu, Laufzeitfehler zu erkennen, siehe [Kapitel 5.3.1 auf Seite 124](#).

Der Debug-Modus wird mit **#Begin_Debug_Mode_Disable** unterbrochen.

Sie können Unterbrechungen des Debug-Modus nicht verschachteln. Achten Sie dabei besonders auf Unterbrechungen in Makros oder Bibliotheken.

Siehe auch

[#Begin_Debug_Mode_Disable](#), [#Include](#)

Beispiel

Event:

```
#Begin_Debug_Mode_Disable
Rem unkritischer Quellcode, der nicht überwacht wird
Rem ...
#End_Debug_Mode_Disable
Rem kritischer Quellcode mit Überwachung auf
Rem Laufzeitfehler
Rem ...
```

Event:

Das Kennwort **Event:** bezeichnet den Anfang des Haupt-Programmabschnitts, der bei jedem Event-Signal aufgerufen wird.

Syntax

Event:

Parameter

- / -

Bemerkungen

Zur Übersicht der Programmabschnitte siehe [Kapitel 4.1.1 auf Seite 93](#).

Der Programmabschnitt **Event:** ist der zentrale Funktionsabschnitt, der im Prozess (typischerweise) in regelmäßigen Abständen aufgerufen wird, bis er gestoppt wird. Je nach Einstellung wird der Aufruf durch einen zyklischen Timer-Event oder durch einen externen Event ausgelöst. Näheres ist in [Kapitel 6 „Prozesse im Betriebssystem“](#) beschrieben.

Siehe auch

[Dim](#), [Init:](#), [Finish:](#)

Beispiel

```
Dim val_1 As Long
```

```
Event:
```

```
    val_1 = -5
```

Finish:

Das Kennwort **Finish:** bezeichnet den Anfang des Programmabschnitts zur Schlussbearbeitung.

Syntax

Finish:

Parameter - / -

Bemerkungen

Zur Übersicht der Programmabschnitte siehe [Kapitel 4.1.1 auf Seite 93](#).

Der Programmabschnitt **Finish:** wird einmalig durchlaufen, sobald der Prozess gestoppt wird.

Wenn der letzte Befehl im Abschnitt **Finish:** abgearbeitet ist, vergeht noch eine bestimmte Zeit, bis der Prozessstatus „gestoppt“ erreicht ist.

Im Unterschied zu *ADbasic* hat der Abschnitt **Finish:** die Priorität, die für den Prozess gewählt ist.

Siehe auch

[Dim](#), [Init:](#), [Event:](#), [ProcessN_Running](#)

Beispiel

```
Dim val_1 As Long
```

```
Finish:
```

```
    val_1 = -5
```

For ... To ... {Step ... } Next

For...Next definiert eine Schleife, die eine bestimmte Zahl an Durchläufen haben sollen.

Syntax

```
For i = X To Y {Step Z}
    ...
Next i
```

'Anweisungsblock

Parameter

i	lokale Zählvariable	LONG
X	Startwert der Laufvariablen	LONG
Y	Endwert der Laufvariablen	LONG
Z	Schrittweite (≥ 1) der Laufvariablen; Vorgabewert: 1	LONG

Bemerkungen

Der Anweisungsblock wird in jedem Fall einmal ausgeführt, auch wenn der Startwert **X** größer als der Endwert **Y** ist.

Deklarieren Sie die Zählvariable als lokale Variable (Datentyp **Long**).

Ein hochpriorer Prozess kann von keinem anderen Prozess unterbrochen werden, auch wenn gerade eine zeitaufwändige Schleife bearbeitet wird. Während dieser Zeit kann der *TiCo*-Prozessor nicht auf andere Events reagieren. Die Schleife darf deshalb in hochpriorien Prozessen nur verwendet werden, wenn die Anzahl der Schleifendurchläufe niedrig gehalten wird.

Siehe auch

Do ... Until, If ... Then ... {Else ... } EndIf, SelectCase

Beispiel

- / -

Function ... EndFunction

`Function...EndFunction` definiert ein Funktions-Makro mit Übergabeparametern und einem Rückgabewert.

Syntax

```
Function macro_name ({val_1, val_2, ...}) As Long

    {Dim var As Long}

    ...                               'Anweisungsblock

    macro_name = ... 'Rückgabewert zuweisen

EndFunction
```

Parameter

<code>macro_</code>	Name der Funktion und Rückgabewert, Datentyp
<code>name</code>	<code>Long</code>
<code>val_1, val_2</code>	Namen der Übergabeparameter; <code>LONG</code>
	für Felder ist die Syntax mit Dimensionsklammern erforderlich: <code>array[]</code> oder <code>Data_n[]</code> .

Bemerkungen

Allgemeine Informationen über Makros finden Sie in [Kapitel 4.5.1 auf Seite 115](#).

Diese Anweisung definiert ein Funktions-Makro, d.h. der vollständige Anweisungsblock zwischen `Function` und `EndFunction` wird an der aufrufenden Stelle eingefügt.

Funktionen erhöhen die Übersichtlichkeit Ihres Quelltextes. Beachten Sie aber, dass jeder Funktionsaufruf die kompilierte Datei vergrößert.

Sie können Funktionen an 3 Stellen einfügen:

1. Vor dem Abschnitt `Init:`
2. Nach dem Abschnitt `Finish:`
3. In einer separaten Datei, die Sie mit `#Include` einbinden (aber nur an einer der Stellen, die unter 1. und 2. angegeben sind).

Beachten Sie bitte, dass Sie in Funktionen:

- keine Prozess-Abschnitte wie **Init:**, **Event:**, oder **Finish:** definieren.
- am Anfang lokale Variablen definieren können, die nur innerhalb der Funktion und für die Dauer der Abarbeitung verfügbar sind.
Eine lokale Variable kann den gleichen Namen haben wie eine Variable, die außerhalb der Funktion definiert wurde.
- dem Funktionsnamen einen Wert zuweisen, damit dieser zum Rückgabewert an die aufrufende Stelle wird.

Eine Funktion wird mit ihrem Namen und allen definierten Argumenten aufgerufen; die Funktion muss in der aufrufenden Programmzeile als Argument verwendet werden, z. B. in einer Zuweisung (siehe Beispiel). Als Argument ist jeder Berechnungsausdruck (auch Felder) zulässig, solange er den passenden Datentyp hat.

Wenn Sie keine Argumente definieren, müssen Sie dennoch beim Aufruf der Funktion die Leerklammern verwenden: `name()`.

Wenn ein Feld als Übergabeparameter einer Funktion verwendet wird, ist die Syntax für Aufruf und Definition unterschiedlich:

- Funktions-Aufruf *ohne* Dimensions-Klammern:
`ret_val = name(array_pass)`
- Funktions-Definition *mit* Dimensions-Klammern:
`Function name(array_def[])`

Werte werden an Feldelemente (eines Felds als Übergabeparameter) zugewiesen wie gewöhnlich:

```
array_def[2] = value
```

Wenn Sie einem Übergabeparameter `x` in der Funktion einen Wert zuweisen, darf beim Funktionsaufruf für `x` nur eine Variable oder ein einzelnes Feld-Element angegeben werden, aber keine Konstante. Auf diese Weise können Übergabeparameter auch einen Rückgabewert enthalten.

Übergabeparameter in symbolischen Namen (**#Define**) führen zu Compiler-Problemen. Nutzen Sie deswegen Definitionen, die ohne Übergabeparameter auskommen.

```
Rem Ungeeignet: Übergabeparameter 'array_def' im Define
#Define pointer1 array_def[2]
Rem Geeignete Alternative
#Define pointer2 2
Function name(array_def[])
  If (pointer1 > 0) Then 'Zeile erzeugt Compiler-Fehler
  If (array_def[pointer2] > 0) Then 'so geht's
  ...
EndFunction
```

Bei Berechnungsausdrücken in einer Funktion sollten die Übergabeparameter in Klammern stehen. Auf diese Weise vermeiden Sie Probleme mit der Rangfolge von Operatoren (z.B. Punkt- vor Strich-Rechnung).

Siehe auch

#Include, Sub ... EndSub

Beispiel

```
Function sumsquare(w1, w2, w3) As Long
Rem Die Funktion berechnet das Summenquadrat aus den
  Werten
Rem w1, w2 und w3
  Dim sum As Long
  sum = w1 + w2 + w3
  sumsquare = sum * sum
EndFunction
```

Ein Aufruf der Funktion erfolgt z.B. mit den Programmzeilen:

```
Event:
  x = sumsquare(x1, x2, x3)
  DAC (1, sumsquare(x1, x2, x3))
```

Aufruf mit Feld

Die gleiche Funktion mit einem Feld als Übergabe-Parameter:

```
Function sumsquare_array(array[]) As Long
  Dim sum As Long
  sum = array[1] + array[2] + array[3]
  sumsquare_array = sum * sum
EndFunction
```

Der Aufruf dieser Funktion erfolgt wieder in ähnlicher Weise (allerdings *ohne* die Dimensionsklammern):

```
Dim my_array[3] As Long
Dim Data_1[3] As Long
Dim x As Long
Event:
x = sumsquare_array(my_array)
DAC(1, sumsquare_array(Data_1))
```

Beim Aufruf können Sie ein globales Feld (wie **Data_1**) oder ein lokales Feld (wie **my_array**) angeben. Tragen Sie nur den Feldnamen ein, ohne Elementnummer und eckige Klammern.

Prozessor	Priorität	
	Hoch	Niedrig
T9	25ns	100µs
T10	25ns	50µs
T11	3,3ns	3,3ns = 0,003µs
T12	1,0ns	1,0ns = 0,001µs

If ... Then ... {Else ... } EndIf

Die Kontrollstruktur bewirkt in Abhängigkeit von einer Bedingung die Ausführung einer Anweisung (**If...Then...**) oder eines Anweisungsblocks (**If...Then...Else...EndIf**).

Syntax

```
If (condition) Then
    ...                               'Anweisungsblock
{Else                               'Der else-Teil ist optional
    ...                               'Anweisungsblock }
EndIf
oder
If (condition) Then instr
```

Parameter

condition Boolesche Bedingung mit den Operatoren <, <=, >, >=, **_LOGIC** |
 =, <>, **And** und **Or**.
 Wenn die Bedingung „Wahr“ ist, werden die Anweisungen nach **Then** ausgeführt.

instr Anweisung (entspricht einer Befehlszeile).

Bemerkungen

Sie können **If**-Strukturen beliebig tief verschachteln; nur die Speichergröße begrenzt Sie hierbei.

Der Anweisungsblock nach **Else** wird (falls vorhanden) schneller ausgeführt als der nach **If...Then**. Dies beschleunigt die Gesamt-Ausführungszeit des **Event**:-Abschnitts, weil die Bedingung meistens den Wert „Falsch“ hat, z. B. bei der Prüfung auf Überschreiten von Grenzwerten.

In der einzeiligen Variante darf die Anweisung weder ein Unterprogramm-Makro (**Sub**) noch ein Funktion-Makro (**Function**) aufrufen.

Siehe auch

[< = > \(Vergleich\)](#), [And](#), [Or](#), [Do ... Until](#), [SelectCase](#)

Beispiel

```
Dim val As Long           'Deklaration

Event:
    val = ADC(1)           'Messwert erfassen

If (val > 3000) Then       'Grenzwert überschritten:
    Digout_Reset(10b)     'Rücksetzen Digout 1
    Digout_Set(01b)       'Setzen Digout 0
Else                       'Grenzwert nicht
                           'überschritten:
    Digout_Reset(01b)     'Rücksetzen Digout 0
    Digout_Set(10b)       'Setzen Digout 1
EndIf                     'Kontrollstruktur Ende
```

#If ... Then ... {#Else ... } #EndIf

Die Präprozessor-Anweisung bewirkt in Abhängigkeit von einer Bedingung die Kompilierung eines Anweisungsblocks (**#If...Then...#Else...#EndIf**).

Syntax

```
#If <SYSPAR> = value Then
...                               'Anweisungsblock
{#Else                           'Der else-Teil ist optional
...                               'Anweisungsblock}
#EndIf
```

Parameter

<SYSPAR> = Boolesche Bedingung (ohne Klammern und **LOGIC** | **value** Hochkommas) mit dem Gleich-Operator =.

Wenn die Bedingung erfüllt ist, werden die folgenden Anweisungen ausgeführt.

Die Systemparameter **<SYSPAR>** und die zugehörigen Werte **value** sind in der Tabelle unten aufgeführt.

<SYSPAR>	value	Bedeutung
ADWIN_ SYSTEM	ADWIN_GOLDII ADWIN_PROII	Einstellung „System“ im Fenster „Compiler Options“.
PROCESSOR	TICO1 TICO2	Einstellung „Processor“ im Fenster „Compiler Options“.

Bemerkungen

In der Bedingung darf nur der Operator „=“ verwendet werden; weder Boolesche Verknüpfungen mit **And** und **Or** noch Klammerungen sind erlaubt. Sie können **#If**-Strukturen beliebig tief verschachteln; nur die Speichergröße begrenzt Sie hierbei.

Es gibt keine einzeilige Variante wie bei **If...Then**.

Bei einem **Kommandozeilen-Aufruf** des Compilers (siehe [Seite 12-9](#)) beziehen sich die Systemparameter auf die beim Aufruf übergebenen Parameter /Sx und /Px.

Siehe auch

< = > (Vergleich), If ... Then ... {Else ...} EndIf

Beispiel

```
Rem Processdelay auf 800µs setzen
#If Processor = Ticol Then
  Rem 800µs = 40000 x 20ns
  Processdelay = 40000
#EndIf
```

Import

Import bindet Funktionen und Unterprogramme aus der angegebenen Library-Datei bei der Kompilierung mit ein.

Syntax

```
Import {path}file
```

Parameter

file	Dateiname der Library-Datei <i>ohne</i> Hochkommas. Die Dateiendung ist .TL1 für Tico1, .TL2 für Tico2.	CONST STRING
path	Vollständiger Pfad mit Laufwerk oder relativer Pfad der Library-Datei; <i>ohne</i> Hochkommas	CONST STRING

Bemerkungen

Allgemeine Informationen über Bibliotheken finden Sie im Abschnitt [Kapitel 4.5.3 auf Seite 116](#).

Fügen Sie **Import**-Anweisungen ganz am Anfang Ihres Quelltextes ein (vor der Variablendeklaration). Wenn Sie im Quelltext einer Library-Datei weitere Library-Dateien importieren, müssen Sie dies zusätzlich auch im aufrufenden Quelltext tun.

Es werden nur diejenigen Funktionen und Unterprogramme aus der Library-Datei eingebunden, die Sie in Ihrem Quelltext aufrufen.

Wenn Sie keinen Pfadnamen angeben, wird die Datei nur im Standard-Verzeichnis (siehe Menü Options [Directories](#), [Seite 68](#)) gesucht. Verwenden Sie bei der Pfadangabe den Backslash "\", um Verzeichnisnamen voneinander zu trennen.

Das Basisverzeichnis für relative Pfadangaben ist – wenn der Quelltext Teil einer Projektdatei ist – das Verzeichnis der Projektdatei, anderenfalls das Verzeichnis der Quelltextdatei.

Derzeit sind im Lieferumfang von *TiCoBasic* keine Bibliotheken enthalten.

Siehe auch

[#Include](#)

Beispiel

```
Rem Benutzerdefinierte Library für TiCol importieren
Import C:\MyFiles\ADwinLibs\dig2volt.TL1
```


In

In gibt den Inhalt einer bestimmten Speicherstelle im I/O-Adressbereich des *TiCo*-Prozessors zurück.

Syntax

```
ret_val = In(addr)
```

Parameter

<code>addr</code>	Adresse der auszulesenden Speicherstelle.	LONG
<code>ret_val</code>	Inhalt der Speicherstelle.	LONG

Bemerkungen

Sie benötigen den Befehl **In** in aller Regel nicht. Nutzen Sie die Befehle der Include-Dateien, um den *TiCo*-Prozessor zu steuern.

In ist nur für spezielle Anwendungen vorgesehen, die in Zusammenarbeit mit unserem Support erstellt werden. Die Dokumentation enthält daher keine Beschreibung von Speicherstellen des *TiCo*-Prozessors.

Wenn in einem Projekt ein extern gesteuerter Prozess enthalten ist, darf **In** nur in einem hochprioren Prozess eingesetzt werden.

Siehe auch

[Out](#)

Beispiel

Rem Das Beispiel zeigt nur die Verwendung der Befehle In
Rem und Out nur symbolisch.

Rem Führen Sie das Programm nicht aus!

Event:

```
If (In(100h) <> 0) Then 'Speicherstelle 100h auslesen
    Out (0, 255)         'Adresse 0 auf den Wert 255
                        'setzen
Else
    Out (0, 0)           'Adresse 0 auf den Wert 0
                        'setzen
EndIf
```

Inc

Inc erhöht den Wert einer lokalen oder globalen ganzzahligen Variablen um Eins.

Syntax

Inc (**var**)

Parameter

var

Name einer lokalen oder globalen Long-Variablen

VAR

~~**CONST**~~

LONG

Bemerkungen

Die Anweisung **Inc** (**var**) führt zum gleichen Ergebnis wie die Programmzeile: **var** = **var** + 1. Allerdings kann diese Anweisung eine geringere Ausführungszeit haben.

Siehe auch

[Dec, + \(Addition\)](#)

Beispiel

```
Dim index As Long
Dim Data_1[1000] As Long

Init:
    index=1

Event:
    Data_1[index] = ADC(1) 'Messwert im Feld ablegen
    Inc(index)             'index um 1 erhöhen
    Rem Nach 1000 Messungen das Programm beenden
    If (index > 1000) Then End
```

#Include

#Include bindet den vollständigen Inhalt einer Include-Datei in den Quelltext ein.

Syntax

```
#Include {path}filename
```

Parameter

<code>filename</code>	Name der einzubindenden Datei (mit Endung „.Inc“) ohne Hochkommas	CONST <u>STRING</u>
<code>path</code>	Vollständiger Pfad mit Laufwerk oder relativer Pfad.	CONST <u>STRING</u>

Bemerkungen

Allgemeine Informationen über Include-Dateien finden Sie in [Kapitel 4.5.2 auf Seite 115](#).

Fügen Sie **#Include**-Anweisungen ganz am Anfang Ihres Quelltextes ein (vor der Variablendeklaration). Im Quelltext einer Include-Datei können Sie weitere Include-Dateien importieren.

Wenn die eingebundene Include-Datei Library-Funktionen verwendet, müssen Sie mit **Import** auch die entsprechenden Library-Dateien einbinden.

Wenn Sie keinen Pfadnamen angeben, wird die Datei nur im Standard-Verzeichnis (siehe Menü Options [Directories](#), [Seite 68](#)) gesucht. Verwenden Sie bei der Pfadangabe den Backslash "\", um Verzeichnisnamen voneinander zu trennen.

Das Basisverzeichnis für relative Pfadangaben ist – wenn der Quelltext Teil einer Projekts ist – das Verzeichnis der Projektdatei, anderenfalls das Verzeichnis der Quelltextdatei.

Um eine der im Lieferumfang von *TiCoBasic* enthaltenen Include-Dateien – sie enthalten Befehle zur Ansteuerung der Hardware-I/Os – einzubinden, geben Sie die ersten Zeichen des Befehls **#Include** ein, drücken [CTRL][SPACE] und wählen die passende Include-Datei aus der Liste.

Folgende Include-Dateien gehören zum Lieferumfang von *TiCoBasic* und enthalten die Befehle zur Ansteuerung der Hardware-I/Os:

GoldIITiCo.inc	<i>ADwin-Gold II</i> : Alle Befehle; Textbaustein einfügen mit IG[TAB].
AInTiCo.inc	<i>ADwin-Pro II</i> : Befehle für mehrere Modultypen; Textbaustein einfügen mit IP[TAB].
AOutTiCo.inc	
AOut1TiCo.inc	
ArincTiCo.inc	
CAN_TiCo.inc	
Cnt_TiCo.inc	
DIO32TiCo.inc	
MIO_TiCo.inc	
MIO-D12_TiCo.inc	
Fieldbus_TiCo.inc	
RS_LIN_TiCo.inc	
SPI_TiCo.inc	
Math_TiCo.inc	Mathematik-Befehle.

Siehe auch

[#Define, Import, Function ... EndFunction, Sub ... EndSub](#)

Beispiel

```

Rem Datei im angegebenen Verzeichnis suchen
#Include C:\Test\demofunc.Inc

Rem Datei im Standard-Verzeichnis suchen
#Include demofunc.Inc

Rem Relative Pfadangabe.
Rem Der Pfad ist relativ zum Verzeichnis der Projektdatei,
Rem wenn der Quelltext zu einem Projekt gehört.
Rem Gehört der Quelltext nicht zu einem Projekt, ist
Rem der Pfad relativ zum Verzeichnis der Quelltextdatei.
#Include .\demofunc.Inc

```

Init:

Das Kennwort **Init:** bezeichnet den Anfang eines Programmabschnitts zur Initialisierung.

Syntax

```
Init:
```

Parameter- / -Bemerkungen

Zur Übersicht der Programmabschnitte siehe [Kapitel 4.1.1 auf Seite 93](#).

Der Programmabschnitt **Init:** wird einmal durchlaufen, sobald der Prozess gestartet wird. Die Zeit zwischen dem letzten Befehl im Abschnitt **Init:** und dem Beginn des Abschnitts **Event:** beträgt etwas mehr als $1 \times \text{Processdelay}$.

Der Programmabschnitt hat die für den Prozess eingestellte Priorität (Menüpunkt „Options / Process“). Bei hoher Priorität kann der Programmabschnitt nicht unterbrochen werden und sollte dann möglichst kurz sein.

Siehe auch

[Dim](#), [Event:](#), [Finish:](#), [Processdelay](#)

Beispiel

```
Dim val_1 As Long
```

```
Init:  
    val_1 = -5
```

Lib_Function ... Lib_EndFunction

`Lib_Function...Lib_EndFunction` definiert in einer Library-Datei eine Funktion mit Übergabe- und Rückgabeparametern.

Syntax

```
Lib_Function lib_name(<lib_par1> {,  
    <lib_par2>, ...} )  
As <fct_type>  
    {Dim var As <var_type>}  
    ...                               'Anweisungsblock  
    lib_name = ...  
Lib_EndFunction
```

Syntax der Übergabeparameter `<lib_par>`:
`<by_type> var_name As <var_type>`

Parameter

<code>lib_name</code>	Name der Library-Funktion und des Rückgabewerts; Datentyp <code><fct_type></code> .
<code><fct_type></code>	Datentyp: <code>Float32</code> , <code>Float64</code> , <code>Float</code> (bis T11), <code>Long</code> .
<code>var_name</code>	Name eines Übergabeparameters innerhalb der Library-Funktion; für Felder ist die Syntax mit Dimensionsklammern erforderlich: <code>array[]</code> oder <code>Data_n[]</code> .
<code><by_type></code>	Methode zur Übergabe der Parameter: <code>ByRef</code> : Zeiger auf Variable oder Feld übergeben. <code>ByVal</code> : Wert übergeben.
<code><var_type></code>	Datentyp: <code>Float32</code> , <code>Float64</code> , <code>Float</code> (bis T11), <code>Long</code> , <code>String</code> .

Bemerkungen

Allgemeine Informationen über Library-Dateien finden Sie in [Kapitel 4.5.3 auf Seite 116](#).

Erstellen Sie Library-Funktionen (und -Unterprogramme) in einer eigenen Quelltext-Datei. Nach der Kompilierung mit „Build / Make lib file“ können Sie mit dem Befehl `Import` diejenigen Module einer Library in einen Prozess einbinden, die dort tatsächlich aufgerufen werden.

Innerhalb einer Library-Funktion können Sie

- lokale Variablen und Felder deklarieren und verwenden. Deklarieren Sie Variablen immer am Anfang, keinesfalls außerhalb des Unterprogramms.
- globale Variablen und Felder verwenden, wenn sie als Parameter übergeben werden.
- dem Funktionsnamen einen Wert zuweisen, damit dieser zum Rückgabewert an die aufrufende Stelle wird.

Innerhalb einer Library-Funktion ist *nicht* erlaubt:

- Prozess-Abschnitte wie `Init:`, `Event:`, oder `Finish:` definieren.
- Library-Funktion oder -Unterprogramm aus der gleichen Library-Datei aufrufen.

Gegebenenfalls müssen Sie die aufzurufende Funktion in eine neue Library-Datei auslagern und von dort importieren.

- Anweisung `SelectCase` benutzen.
- Symbolische Namen mit `#Define` deklarieren.
- Auf Hardware zugreifen wie auf analoge oder digitale Ein- oder Ausgänge.

Die Methoden zur Übergabe von Parametern unterscheiden sich folgendermaßen:

- **ByRef**: Die Library-Funktion kann den Parameter verändern, so dass der geänderte Wert anschließend im aufrufenden Programm vorliegt (es wird die Adresse des Parameters übergeben).
- **ByVal**: Die Library-Funktion kann nur auf den Wert des Parameters zugreifen, diesen aber selbst nicht ändern. Der Parameter bleibt also für das aufrufende Programm gleich.

Wenn ein Feld als Übergabeparameter einer Library verwendet wird, ist die Syntax für Definition und Aufruf unterschiedlich:

- Definition des Funktionsparameters *mit* Klammern:
`Lib_Function name(array[]) ...`
- Funktionsaufruf mit dem Parameter *ohne* Klammern:
`ret_val = name(array)`

Definieren Sie Felder als Übergabeparameter immer **ByRef** und ohne Feldgröße. Sie können keine Ringbuffer-Felder als Übergabeparameter verwenden.

Siehe auch

`Lib_Sub ... Lib_EndSub`, `Import`, `Function ... EndFunction`, `Sub ... EndSub`

Beispiel

```
Rem ----- Mittelwertbildung -----
Rem Binärdatei speichern als MEAN.TL1
Rem (Endung je nach Prozessor)
Lib_Function average(ByRef array[] As Long, ByVal ptr As Long,
    ByVal cnt As Long) As Long
    Dim i As Long
    average = 0
    If (cnt > 0) Then
        For i = ptr To (ptr + cnt)
            average = average + array[i]
        Next i
        average = average / cnt
    EndIf
Lib_EndFunction
```

Library-Aufruf

Den Aufruf der Library-Funktion **average** sehen Sie im folgenden Beispiel, einem sogenannten „moving average filter“:

```

Rem Library 'MEAN' importieren
Import C:\MyFiles\ADwinLibs\MEAN.tl1 'siehe oben
#Define cnt 10 'Anzahl der Summanden (Samples)
#Define samples Data_1 'Anzahl Messwerte
#Define filtered Data_2 'Anzahl gefilterte Messwerte
#Define length 1000 'Feldlänge
Dim samples[length] As Long 'Quell-Feld
Dim filtered[length] As Long 'Ziel-Feld
Dim i As Long 'Zählvariable

Init:
    i = 1 'Zählvariable initialisieren
    Processdelay = 40000 'Messung mit 1 kHz

Event:
    samples[i] = ADC(1) 'Analogwerte messen und speichern
    Inc i 'Zählvariable erhöhen
    If (i > length) Then End '1000 Messungen durchgeführt?
    'Wenn ja: Finish

Finish:
    For i = 1 To (length - cnt) 'Alle Messwerte aufrufen
        Rem Library-Funktion "average" aufrufen
        filtered[i + cnt] = average(samples,i,cnt)
        Rem Beachten Sie die Syntax beim Feld 'samples'
        Rem als Übergabeparameter ohne eckige Klammern
    Next i

```

Lib_Sub ... Lib_EndSub

`Lib_Sub...Lib_EndSub` definiert in einer Library-Datei ein Unterprogramm mit Übergabeparametern.

Syntax

```
Lib_Sub lib_name(<lib_par1> {, <lib_par2>, ...})
    {Dim var As <var_type>}
    {#Define name expression}
    ...
    'Anweisungsblock
Lib_EndSub
```

Syntax der Übergabeparameter `<lib_par>`:

```
<by_type> var_name As <var_type>
```

Parameter

<code>lib_name</code>	Name des Library-Unterprogramms.
<code>var_name</code>	Name eines Übergabeparameters innerhalb des Library-Unterprogramms; für Felder ist die Syntax mit Dimensionsklammern erforderlich: <code>array[]</code> oder <code>Data_n[]</code> .
<code><by_type></code>	Methode zur Übergabe eines Parameters: <code>ByRef</code> : Zeiger auf Variable oder Feld übergeben. <code>ByVal</code> : Wert übergeben.
<code><var_type></code>	Datentyp: <code>Float</code> , <code>Long</code> , <code>String</code> .

Bemerkungen

Allgemeine Informationen über Bibliotheken (Libraries) finden Sie in [Kapitel 4.5.3 auf Seite 116](#).

Erstellen Sie Library-Unterprogramme (und -Funktionen) in einer eigenen Quelltext-Datei. Mit „Build / Make lib file“ kompilieren sie diese und erzeugen die Library-Datei. Der Befehl `Import` bindet diejenigen Module einer Library in einen Prozess ein, die dort tatsächlich aufgerufen werden.

Innerhalb eines Library-Unterprogramms können Sie

- lokale Variablen und Felder deklarieren und verwenden. Deklarieren Sie Variablen immer am Anfang, keinesfalls außerhalb des Unterprogramms.
- globale Variablen und Felder verwenden, wenn sie als Parameter übergeben werden.

Innerhalb eines Library-Unterprogramms ist *nicht* erlaubt:

- Prozess-Abschnitte wie **Init:**, **Event:**, oder **Finish:** definieren.
- Library-Funktion oder -Unterprogramm aus der gleichen Library-Datei aufrufen.
Gegebenenfalls müssen Sie die aufzurufende Funktion in eine neue Library-Datei auslagern und von dort importieren.
- die Anweisung **SelectCase** benutzen.
- Symbolische Namen mit **#Define** deklarieren.
- Auf Hardware zugreifen wie analoge oder digitale Ein- oder Ausgänge.

Die Methoden zur Übergabe von Parametern unterscheiden sich folgendermaßen:

- **ByRef**: Das Library-Unterprogramm kann den Parameter verändern, so dass der geänderte Wert anschließend im aufrufenden Programm vorliegt (es wird die Adresse des Parameters übergeben).
- **ByVal**: Das Library-Unterprogramm kann nur auf den Wert des Parameters zugreifen, diesen aber selbst nicht ändern. Der Parameter bleibt also für das aufrufende Programm gleich.

Wenn ein Feld als Übergabeparameter einer Library verwendet wird, ist die Syntax für Definition und Aufruf unterschiedlich:

- Definition des Unterprogrammparameters *mit* Klammern: **LIB_Sub subname(array[]) ...**
- Aufruf mit dem Parameter *ohne* Klammern:
subname(array)

Definieren Sie Felder als Übergabeparameter immer **ByRef** und ohne Feldgröße. Sie können keine Ringbuffer-Felder als Übergabeparameter verwenden.

Siehe auch

Lib_Function ... Lib_EndFunction, **Import, Function ... EndFunction**, **Sub ... EndSub**

Beispiel

```
Rem Binärdatei speichern als DIG2VOLT.TL1
Rem (Endung je nach Prozessor)
Rem Messwertumrechnung von Digit (0...65535)
Rem nach Volt (±10V)
Lib_Sub dig2volt (ByRef digit[] As Long, ByVal ptr As Long,
    ByVal cnt As Long, ByVal gain As Long,
    ByRef volt[] As Float)
    Dim i As Long
    For i = ptr To (ptr + cnt)
        volt[i] = ((digit[i] * 20 / 65536) - 10) / gain
    Next i
Lib_EndSub
```

Library-Aufruf

Den Aufruf der Library-Funktion `dig2volt` sehen Sie im folgenden Beispiel, einer Messwert-Umwandlung:

```

Rem Die Library 'DIG2VOLT' wird importiert
Import C:\MyFiles\ADwinLibs\DIG2VOLT.tl1
#Define cnt 1000           'Anzahl der Samples
#Define ptr 1              'Erstes Sample
#Define gain 1             'Verstärkungsfaktor PGA
#Define samples Data_1     'Speicher für Messwerte
#Define scaled Data_2      'Speicher konvertierte Messwerte
#Define length 1000        'Feldlänge
Dim samples[length] As Long 'Quell-Feld
Dim scaled[length] As Long 'Ziel-Feld
Dim i As Long              'Zählvariable

Init:
    i = 1                  'Startwert Zählvariable
    Processdelay = 40000   'Messung mit 1 kHz

Event:
samples[i] = ADC(1) 'Analogwerte messen und speichern
    Inc i            'Zählvariable erhöhen
    If (i > length) Then End '1000 Messungen durchgeführt?
                        'Wenn ja: Finish

Finish:
    Rem Die gewünschten Messwerte konvertieren durch
    Rem Aufruf des Library-Unterprogramms 'dig2volt'
    dig2volt(samples,ptr,cnt,gain,scaled)

    Rem Beachten Sie die Syntax beim Feld 'samples'
    Rem als Übergabeparameter ohne eckige Klammern []

```

Max_Long

Max_Long liefert den größeren von 2 ganzzahligen Werten.

Syntax

```
ret_val = Max_Long(val1, val2)
```

Parameter

<code>val_1</code>	Vergleichswert 1	LONG
<code>val_2</code>	Vergleichswert 2	LONG
<code>ret_val</code>	Der größere der beiden Vergleichswerte.	LONG

Bemerkungen

- / -

Siehe auch

[Absl](#), [Min_Long](#)

Beispiel

```
Event:
    Par_10 = Max_Long(Par_1, Par_2)
```

Min_Long

Min_Long liefert den kleineren von 2 ganzzahligen Werten.

Syntax

```
ret_val = Min_Long(val1, val2)
```

Parameter

val_1 Vergleichswert 1

LONG

val_2 Vergleichswert 2

LONG

ret_val Der kleinere der beiden Vergleichswerte.

LONG

Bemerkungen

- / -

Siehe auch

[Absl](#), [Max_Long](#)

Beispiel

Event:

```
Par_10 = Min_Long(Par_1, Par_2)
```


NOP

NOP (No OPeration) lässt den Prozessor für die Zeit von einem Taktzyklus warten.

Syntax

NOP

Bemerkungen

Diese Anweisung benötigt in der Regel einen Prozessor-Taktzyklus:

TiCo1	20ns
TiCo2	10ns

Sie können mit dieser Anweisung erforderliche Wartezeiten überbrücken (beispielsweise nach **Set_Mux1**), wenn es keine sinnvolle andere Verwendung der Prozessorzeit gibt.

Siehe auch

[NOPs](#), [Sleep](#)

NOPs

NOPS lässt den Prozessor für mehrere Taktzyklen warten. Die Wartezeit entspricht der Anzahl mehrerer **NOP**-Befehle.

Syntax

NOPS (*number*)

Parameter

number

Anzahl (≥ 1) der einzufügenden NOP-Befehle.
Für Konstanten ist 32767 der größte mögliche Wert.

LONG


Bemerkungen

Verwenden Sie **NOPS** nur in hochprioren Prozessen. In niederprioren Prozessen ist **Sleep** die bessere Wahl, vor allem weil bei hohen Werten für *value* unerwartet lange Wartezeiten auftreten können.

Der Befehl **NOPS** benötigt weniger Speicherplatz als die entsprechende Anzahl an **NOP**-Befehlen.

Wenn *number* eine Konstante ist, ersetzt **NOPS** die angegebene Zahl von **NOP**-Befehlen exakt. Bei Verwendung einer Variablen werden 2...4 Taktzyklen zusätzlich benötigt; ebenso werden beim Zugriff auf externen Speicher oder bei Berechnungen weitere Taktzyklen benötigt.

Unter bestimmten Voraussetzungen fügt der Compiler nach **NOPS** automatisch einen zusätzlichen **NOP**-Befehl ein. Wenn stattdessen eine exakte Wartezeit benötigt wird, verringern Sie *number* um 1 und fügen den **NOP**-Befehl (hinter **NOPS**) von Hand ein.

Wenn die Ausführung von **NOPS** unerwartet lange dauert, könnte die Variable *number* einen negativen Wert enthalten. Verwenden Sie stattdessen eine positive Konstante. 

Siehe auch

NOP, **Sleep**

Beispiel

- / -

Not

Not invertiert die Bits eines Werts.

Syntax

```
ret_val = Not(val)
```

Parameter

val zu invertierender Wert (kein logischer Ausdruck)

LONG

ret_val invertiertes Argument

LONG

Bemerkungen

Not arbeitet nur bitweise. Daher können Sie mit **Not** keine logischen Ausdrücke (True / False) negieren. Falsch wäre also: **Not** (**Par_2** > 2).

Siehe auch

And, If ... Then ... {Else ... } EndIf, Or, XOr

Beispiel

```
Dim val_1 As Long
```

```
Dim val_2 As Long
```

```
val_1 = -3
```

```
Rem -3 = 11111111111111111111111111111111101b
```

```
val_2 = Not(val_1) 'Ergebnis: val_2 = 010b = 2
```

Or

Der Operator **Or** verknüpft zwei ganzzahlige Werte bitweise oder zwei Boolesche Ausdrücke als Boolescher Operator.

Syntax

```
ret_val = val_1 Or val_2           'Bitweiser Operator
If ((expr1) Or (expr2)) Then      'Boolescher Operator
```

Parameter

val_1, val_2	Ganzzahliger Wert	LONG
expr1, expr2	Boolescher Ausdruck mit dem Wert „wahr“ oder „falsch“	LOGIC

Bemerkungen

Sie können mit **Or** nur gleichartige Ausdrücke verknüpfen (ganzzahlige oder Boolesche), ein Mischen ist nicht möglich.

Sie können Boolesche Ausdrücke nur mit den Anweisungen **If ... Then ... Else** oder **Do ... Until** verwenden (Variablen können keine Booleschen Werte annehmen).

Wenn Sie in einer Zeile mehrere Boolesche Operatoren verwenden, müssen Sie jede Verknüpfung separat in Klammern setzen. Bei der bitweisen Verknüpfung ganzzahliger Werte sind keine Klammern erforderlich.

Siehe auch

And, If ... Then ... {Else ... } EndIf, Not, XOr

Beispiel

```
Dim val_1, val_2, val3 As Long
Dim x, val4 As Long

Init:
    Rem Bitweise Operator
    val_1 = 0100b
    val_2 = 0110b
    val3 = val_1 Or val_2 'Ergebnis: val3 = 0110b

    Rem Boolescher Operator
    x = 15

Event:
    If ((x < 3) Or (x > 9)) Then
        val4 = 1
    Else
        val4 = 0
    EndIf 'Ergebnis: val4 = 1
```

Out

Out schreibt einen Wert in eine bestimmte Speicherstelle im I/O-Adressbereich des *TiCo*-Prozessors.

Syntax

```
Out (addr, value)
```

Parameter

addr	Adresse der zu beschreibenden Speicherstelle.	LONG
value	Zu schreibender Wert.	LONG

Bemerkungen

Sie benötigen den Befehl **Out** in aller Regel nicht. Nutzen Sie die Befehle der Include-Dateien, um den *TiCo*-Prozessor zu steuern.

Out ist nur für spezielle Anwendungen vorgesehen, die in Zusammenarbeit mit unserem Support erstellt werden. Die Dokumentation enthält daher keine Beschreibung von Speicherstellen des *TiCo*-Prozessors.

Siehe auch

[In](#)

Beispiel

*Rem Das Beispiel zeigt die Verwendung der Befehle IN und
Rem OUT nur symbolisch. Führen Sie das Programm nicht aus!*

Init:

If (In (100h)=0) Then	<i>'Speicherstelle 100h = 0?</i>
Out (1,12)	<i>'Wert 12 in Adresse 1 schreiben</i>
EndIf	

Processdelay

Die Systemvariable **Processdelay** definiert das Processdelay (die Zykluszeit) eines Prozesses.

Processdelay ersetzt die Systemvariable **Globaldelay**, die aus Kompatibilitätsgründen weiterhin gültig ist.

Syntax

```
ret_val = Processdelay
```

oder

```
Processdelay = expr
```

Parameter

<code>ret_val</code>	Aktuell eingestellte Zykluszeit in Taktzyklen.	LONG
<code>expr</code>	Einzustellende Zykluszeit: Anzahl (≥ 1) der Taktzyklen.	LONG

Bemerkungen

Bei einem zeitgesteuerten Prozess wird der Abschnitt **Event**: vom internen Zähler zyklisch und in festen Zeitabständen aufgerufen. Der Zeitabstand zwischen zwei Aufrufen, Zykluszeit oder Processdelay genannt, wird in Zähler-Taktzyklen gezählt.

Die Zeiteinheit des Processdelay ist bei TiCo1 20ns, bei TiCo2 10ns.

Wählen Sie bei hochprioren Prozessen eine ausreichend großes Processdelay, um eine Überlastung des *TiCo*-Prozessors zu vermeiden. Als Faustregel sollte die Auslastung des Prozessors (Anzeige: „Busy x%“ in der Statusleiste) möglichst unter 90% bleiben und darf keinesfalls 100% übersteigen.

Wenn die Bearbeitungszeit des Abschnitts **Event**: größer ist als das Processdelay, kommen der nächste Zähler-Aufruf und die folgenden verspätet.

Ein konstantes Processdelay können Sie festlegen, indem Sie der Variablen **Processdelay** im Abschnitt **Init**: einen Wert zuweisen. Sie überschreiben damit ggf. die Voreinstellung, die Sie in der Entwicklungsumgebung *TiCoBasic* im Dialogfenster „Options / Process“ unter „Initial Processdelay“ eingegeben haben.

Sie können die Systemvariable innerhalb eines Abschnitts nur einmal beschreiben.

Wenn der Parameter **Processdelay** innerhalb eines Prozesszyklus, d.h. im Abschnitt **Event**: geändert wird, wird die Zykluszeit dadurch sofort verändert. Dies kann insbesondere bei einer Verkürzung der Zykluszeit kritisch sein: Achten Sie immer darauf, dass die Bearbeitungszeit des Prozesszyklus kürzer bleibt als die neu eingestellte Zykluszeit.

Siehe auch

[Read_Timer](#), Kapitel 6.2.1 „Processdelay“

Beispiel

Init:

```
Rem Zykluszeit auf 800 µs einstellen
```

```
Processdelay = 40000
```

Wenn Sie eine längere Zykluszeit benötigen als mit **Processdelay** einstellbar ist, können Sie eine Hilfsvariable verwenden:

Init:

```
Rem Max. Zykluszeit einstellen, bei TiCo1 etwa 42,9 s
```

```
Processdelay = 2147483647
```

```
Rem Hilfsvariable initialisieren
```

```
Par_1 = 0
```

Event:

```
Inc Par_1
```

```
Rem 100fache Zykluszeit verwenden: bei TiCo1 71,5 Min.
```

```
If (Par_1 = 100) Then
```

```
  Par_1 = 0
```

```
  Rem Programm ausführen
```

```
  ...
```

```
EndIf
```


Process_Error

[Process_Error](#) gibt den (in der Regel) zuletzt aufgetretenen Fehler des aktuellen Prozesses zurück.

Syntax

```
ret_val = Process_Error
```

Parameter

<code>ret_val</code>	Nummer des zuletzt aufgetretenen Fehlers im Prozess. Einige Fehlernummern: LONG
----------------------	---

0: kein Fehler
10: Zugriff auf eine zu große Elementnummer eines globalen Felds.
11: Zugriff auf eine zu kleine Elementnummer (≤ 0) eines globalen Felds.
12: Zugriff auf eine zu große Elementnummer eines lokalen Felds.
13: Zugriff auf eine zu kleine Elementnummer (≤ 0) eines lokalen Felds.

Bemerkungen

Der Rückgabewert der Systemvariablen ist nur definiert, wenn der Debug-Modus eingeschaltet ist (siehe [Option Debug mode](#), [Seite 72](#)). Die Variable kann nur gelesen werden.

Siehe auch

- / -

ProcessN_Running

Die Systemvariable `ProcessN_Running` gibt den aktuellen Status eines bestimmten Prozesses zurück.

Syntax

```
ret_val = ProcessN_Running
```

Parameter

`n` Prozessnummer `n` (0...4)

CONST

LONG

`ret_val` Prozess-Status:
 1 Prozess läuft
 0 Prozess ist gestoppt
 -1 Prozess wird gestoppt

LONG

Bemerkungen

Diese Systemvariable kann nur gelesen werden.

Siehe auch

EndBeispiel

Event:

```
Rem Status von Prozess 2 ermitteln
Par_2 = Process2_Running
```

Read_Timer

Read_Timer gibt den aktuellen Zählerstand des ADwin-Systems zurück.

Syntax

```
ret_val = Read_Timer()
```

Parameter

ret_val Aktueller Zählerstand in Prozessor-Takzyklen.

LONG

Bemerkungen

Der Zählerstand kann nur gelesen werden.

Ein Taktzyklus ist je nach Prozessortyp unterschiedlich lang:

Prozessor	Taktzyklus
TiCo1	20ns
TiCo2	10ns

Sie können eine Zeitdifferenz aus der Differenz von 2 Zählerständen ermitteln. Beachten Sie dabei bitte, dass ein gelesener Zählerstand nach einer bestimmten Zeit wieder erneut erreicht wird. Dieser Zählerüberlauf hängt von den oben angegebenen Zeiteinheiten ab:

Prozessor	Dauer
TiCo1	42,9 Sek.
TiCo2	21,5 Sek.

Siehe auch

[Processdelay](#)

Beispiel

```
Dim timervalue As Long
Event:
    timervalue = Read_Timer()
```

Siehe auch Beispiel `seconds_timer_TiCo.bas` im Ordner
C:\ADwin\TiCoBasic\samples_ADwin

Rem, '

Die Compiler-Anweisungen *Rem* oder „*'*“ ermöglichen das Einfügen von Kommentaren im Quelltext. Sie bewirken, dass der in einer Programmzeile folgende Text vom Compiler ignoriert wird.

Syntax

```
Rem comment  
  
instr : Rem comment  
  
instr 'comment
```

Parameter

<i>comment</i>	Beliebige Zeichenfolge
<i>instr</i>	TiCoBasic-Anweisung

Bemerkungen

Die Anweisung gilt nur für die Zeile, in der sie benutzt wird. Für einen mehrzeiligen Kommentar müssen Sie jede Zeile einzeln mit der Anweisung *Rem* oder dem Zeichen *'* beginnen.

Wenn Sie eine *Rem*-Anweisung hinter einer anderen Anweisung in einer Befehlszeile einfügen, dann trennen Sie beide durch einen Doppelpunkt *:*. Bei dem Kommentarzeichen *'* ist dies nicht erforderlich.

Sie können die Position eines Kommentars hinter einer Anweisung festlegen; mehr dazu finden Sie unter [Kommentare positionieren](#) (Seite 26).

Beispiel

```
Rem Dies ist ein Kommentar, der mehr als eine  
Rem Textzeile benötigt  
'Dies ist auch eine Kommentarzeile  
Dim min As Long : Rem Kommentar nach einer Anweisung  
Dim max As Long 'Kommentar nach einer Anweisung
```

RingBuffer

Mit `Dim Data_n As RingBuffer_For_x` wird ein globales `Data`-Feld als Ringspeicher zum Lesen oder zum Schreiben dimensioniert.

Syntax

```
Dim Data_n[length] As Long As RingBuffer_For_Read
    {At <Mem_Type>}

Dim Data_n[length] As Long As RingBuffer_For_Write
    {At <Mem_Type>}
```

Parameter

<code>Data_n</code>	Name des deklarierten <code>Data</code> -Felds (<code>n</code> : 1...16).	
<code>length</code>	Feldgröße (≥ 1): Anzahl der Elemente im Feld. Im externen Datenspeicher ist der Wertebereich eingeschränkt.	CONST
<code><Mem_Type></code>	Speicherbereich, in dem die Variablen abgelegt werden: <code>DRAM_Extern</code> : externer Datenspeicher. Hier ist der Wertebereich für <code>length</code> nur in Schritten einstellbar: $length = 8 \times a + 7$; $a \geq 0$ <code>SRAM_Extern</code> : externer Datenspeicher bei Pro II-Modulen (anstelle von DRAM). <code>DM_Local</code> : interner Datenspeicher (Default).	

Bemerkungen



Der Umgang mit Ringspeichern ist nicht einfach. Bei falscher Anwendung können schwer auffindbare Fehler entstehen. Die Verwendung des Datentyps `RingBuffer` ist daher erfahrenen Benutzern von *AD-basic* und *TiCoBasic* vorbehalten. Beachten Sie in jedem Fall die Hinweise in [Kapitel 4.3.3 auf Seite 105](#).

Sie können nur `Data`-Felder als Ringspeicher verwenden. Dadurch kann ein `RingBuffer`-Feld nicht mehr gleichzeitig als „normales“ Feld verwendet werden.

Nach der Dimensionierung sollten Sie den Ringspeicher im Abschnitt **Init:** mit **RingBuffer_Clear** initialisieren.

Für einen Ringspeicher im externen Datenspeicher gilt:

- Wenn Sie eine nicht erlaubte Feldgröße **length** angeben, wird der Ringspeicher automatisch mit der nächstgrößeren, erlaubten Feldgröße dimensioniert. Beispielsweise ändert der Compiler die Feldgröße **[1000]** automatisch in **[1007]**.
- In einem Lese-Ringspeicher sollten Sie die Daten nach der Dimensionierung im Abschnitt **Init:** mit dem Befehl **Refresh_RingBuffer** aktualisieren.

Wenn Sie schneller Daten in einen Ringspeicher schreiben als auslesen, werden alte gespeicherte Daten überschrieben und gehen damit verloren. Zur Abhilfe können Sie die Befehle **RingBuffer_Empty** und **RingBuffer_Full** verwenden.

Siehe auch

[Dim](#), [Data_n](#), [Refresh_RingBuffer](#), [RingBuffer_Clear](#), [RingBuffer_Empty](#), [RingBuffer_Full](#), „Globale Felder (Arrays)“ auf Seite 99, „Die Datenstruktur RingBuffer“ auf Seite 105

Beispiel

```
REM Dimension the global array Data_15 with  
REM 1007 LONG elements as read ringbuffer  
Dim Data_15[1007] As Long As RingBuffer_For_Read
```

Refresh_RingBuffer

Refresh_RingBuffer aktualisiert die Daten in einem Lese-Ringspeicher im externen Speicher.

Syntax

```
Refresh_RingBuffer (data_no)
```

Parameter

data_no Nummer (1...16) des Ringspeichers **Data_x**.

LONG

Bemerkungen

Bei einem Ringspeicher im internen Speicher sowie bei einem Schreib-Ringspeicher im externen Speicher ist die Aktualisierung mit **Refresh_RingBuffer** nicht erforderlich.

Bei einem Lese-Ringspeicher im externen Speicher müssen Sie die Daten (vor dem Auslesen) in folgenden Fällen aktualisieren:

- Im Lese-Ringspeicher sind schon Daten vorhanden und es wird zum ersten Mal ein Wert gelesen.
- Nach dem vorherigen Auslesen enthielt der Ringspeicher weniger als 8 Werte und anschließend wurden neue Daten in den Ringspeicher geschrieben.

Anders gesagt: Sie können die regelmäßige Aktualisierung mit **Refresh_RingBuffer** vermeiden, wenn der Ringspeicher nach jedem Auslesen eines Werts 8 oder mehr Werte enthält.

Die Aktualisierung richtet in keinem Fall Schaden an, d.h. es können keine Werte doppelt gelesen werden oder verloren gehen. Im Zweifelsfall ist es also besser, einmal zuviel mit **Refresh_RingBuffer** zu aktualisieren als einmal zuwenig.

Siehe auch

[RingBuffer](#) (Deklaration), [RingBuffer_Clear](#), [RingBuffer_Empty](#), [RingBuffer_Full](#), Kapitel 4.3.3 „Die Datenstruktur RingBuffer“

Beispiel

```
#Include AOut_TiCo.inc
Rem Use global array Data_12 as ringbuffer
Dim Data_12[999] As Long As Ringbuffer_For_Read At DRAM_Extern
Dim i As Long

Init:
  Rem initialize ringbuffer
  RingBuffer_Clear(12, Par_1)
  Rem wait until ringbuffer contains more than 7 values
  Do
    Until (RingBuffer_Full(12, Par_1) > 7)
    Rem refresh ringbuffer data
    Refresh_RingBuffer(12)

Event:
  Rem read 500 ringbuffer values, but always have more
  Rem than 7 values left in it
  If (RingBuffer_Full(12, Par_1) > 507)
    For i = 1 To 500
      Par_10 = Data_12
      DAC(2, Par_10) 'do something with Par_10
    Next i
  EndIf

Finish:
  For i = 1 To RingBuffer_Full(12, Par_1)
    Par_10 = Data_12
  Next i
```

RingBuffer_Clear

RingBuffer_Clear initialisiert den Schreib- und den Lese-Zeiger eines Ringspeichers.

Syntax

```
RingBuffer_Clear (data_no, Par_x)
```

Parameter

data_no	Nummer (1...16) des Ringspeichers Data_x .	LONG
Par_x	Bei Datenübertragung <i>ADwin</i> CPU / <i>TiCo</i> : Globale Variable (Par_1 ... Par_80), die eine Kopie des Schreib- oder Lesezeigers auf den Ringspeicher enthält.	LONG

Bemerkungen

Die Initialisierung eines Ringspeichers ist in 2 Fällen sinnvoll:

- Vor dem ersten Zugriff auf den Ringspeicher.
Sie sollten dies auf jeden Fall im Abschnitt **Init**: tun, weil die Ringspeicher-Zeiger bei der Dimensionierung nicht initialisiert werden.
- Während des Programmlauf, wenn Sie alle im Feld gesammelten Daten (z.B. wegen eines Messfehlers) verwerfen wollen.

Das Initialisieren des Schreib- und des Lese-Zeigers ändert die im Ringspeicher enthaltenen Daten nicht.

Die globale Variable **Par_x** wird nur für die Datenübertragung zwischen *ADwin* CPU (z.B. T11) und *TiCo*-Prozessor benötigt. Die Variable enthält dann eine Kopie des Schreib- oder Lesezeigers auf den Ringspeicher.

Wenn Sie beispielsweise in *ADbasic* mit **Set_TiCo_RingBuffer** Daten in dem Ringspeicher schreiben, aktualisiert die *ADwin* CPU bei jedem Schreibvorgang den Schreibzeiger und kopiert den Wert in die globale Variable **Par_x** auf dem *TiCo*-Prozessor. Mit dem Wert in **Par_x** kann **RingBuffer_Full** in *TiCoBasic* die Anzahl der belegten Elemente im Ringspeicher berechnen.

Bei einer Datenübertragung zwischen ADwin CPU und TiCo-Prozessor muss folgende Reihenfolge eingehalten werden, um einen Ringspeicher zu initialisieren:

1. Initialisieren Sie in *TiCoBasic* den Ringspeicher mit **Ringbuffer_Clear**.
2. Stellen Sie sicher, dass weder in *ADbasic* noch in *TiCoBasic* Daten in den Ringspeicher geschrieben oder daraus gelesen werden.
3. Initialisieren Sie in *ADbasic* die Datenübertragung mit **TDrv_Init**.
4. Jetzt können Sie mit dem Ringspeicher weiter arbeiten.

Siehe auch

[RingBuffer](#) (Deklaration), [Refresh_RingBuffer](#), [RingBuffer_Empty](#), [RingBuffer_Full](#), Kapitel 4.3.3 „Die Datenstruktur RingBuffer“

Beispiel

```
REM 1007 LONG elements as write ringbuffer
Dim Data_11[1007] As Long As Ringbuffer_For_Write

Init:
  Rem initialize read pointer of Data_11 (using Par_4)
  Ringbuffer_Clear(11, Par_4)
```

RingBuffer_Empty

RingBuffer_Empty gibt die Anzahl der freien Elemente in einem Schreib-Ringspeicher zurück.

Syntax

```
ret_val = RingBuffer_Empty (data_no, Par_x)
```

Parameter

data_no	Nummer (1...16) des Ringspeichers Data_x .	LONG
Par_x	Bei Datenübertragung <i>ADwin</i> CPU / <i>TiCo</i> : Globale Variable (Par_1...Par_80), die eine Kopie des Lesezeigers auf den Ringspeicher enthält. Bei Datenübertragung ext. Speicher / <i>TiCo</i> : 0.	LONG
ret_val	Anzahl der freien Ringspeicher-Elemente.	LONG

Bemerkungen

Initialisieren Sie den Schreibzeiger des Schreib-Ringspeichers mit **RingBuffer_Clear**, bevor Sie das erste Mal auf den Ringspeicher zugreifen.

Wenn Sie Daten in einen Ringspeicher schreiben wollen, sollten Sie vorher mit **RingBuffer_Empty** überprüfen, ob noch genügend Platz im Ringspeicher frei ist.

Bitte beachten Sie, dass Ringspeicher im externen Speicher immer in Schritten von 8 Elementen dimensioniert werden (see [Seite 206](#)).

Die globale Variable **Par_x** wird nur für die Datenübertragung zwischen *ADwin* CPU (z.B. T11) und *TiCo*-Prozessor benötigt. Die Variable enthält dann eine Kopie des Schreib- oder Lesezeigers auf den Ringspeicher.

Wenn Sie beispielsweise in *ADbasic* mit **Get_TiCo_RingBuffer** Daten aus dem Ringspeicher lesen, aktualisiert die *ADwin* CPU bei jedem Lesevorgang den Lesezeiger und kopiert den Wert in die globale Variable **Par_x** auf dem *TiCo*-Prozessor. Mit dem Wert in **Par_x** kann **RingBuffer_Empty** in *TiCoBasic* die Anzahl der freien Elemente im Ringspeicher berechnen.

Siehe auch

[RingBuffer](#) (Deklaration), [RingBuffer_Full](#), [Get_TiCo_RingBuffer](#) (*ADbasic*), [Kapitel 4.3.3 „Die Datenstruktur RingBuffer“](#)

Beispiel

```
REM 1007 LONG elements as write ringbuffer
Dim Data_11[1007] As Long As Ringbuffer_For_Write
```

Init:

```
Ringbuffer_Clear(11, Par_4)
```

Event:

```
Rem read number of unused elements in Data_11,
Rem using Par_4 as read pointer
Par_1 = RingBuffer_Empty(11, Par_4)
```

RingBuffer_Full

RingBuffer_Full gibt die Anzahl der genutzten Elemente in einem Lese-Ringspeicher zurück.

Syntax

```
ret_val = RingBuffer_Full (data_no, Par_x)
```

Parameter

data_no	Nummer (1...16) des Ringspeichers Data_x .	LONG
Par_x	Bei Datenübertragung <i>ADwin</i> CPU / <i>TiCo</i> : Globale Variable (Par_1 ... Par_80), die eine Kopie des Schreibzeigers (auf der <i>ADwin</i> CPU) auf den Ringspeicher enthält.	VAR
	Bei Datenübertragung ext. Speicher / <i>TiCo</i> : 0.	LONG
ret_val	Anzahl der belegten Ringspeicher-Elemente.	LONG

Bemerkungen

Initialisieren Sie den Schreibzeiger des Schreib-Ringspeichers mit **RingBuffer_Clear**, bevor Sie das erste Mal auf den Ringspeicher zugreifen.

Wenn Sie Daten aus einem Ringspeicher lesen, sollten Sie vorher mit **RingBuffer_Full** überprüfen, ob im Ringspeicher noch Daten enthalten sind. Falls keine Daten mehr vorhanden sind, wird aus dem Ringspeicher ein undefinierter Wert gelesen.

Bitte beachten Sie, dass Ringspeicher im externen Speicher immer in Schritten von 8 Elementen dimensioniert werden (see [Seite 206](#)).

Die globale Variable **Par_x** wird nur für die Datenübertragung zwischen *ADwin* CPU (z.B. T11) und *TiCo*-Prozessor benötigt. Die Variable enthält dann eine Kopie des Schreib- oder Lesezeigers auf den Ringspeicher.

Wenn Sie beispielsweise in *ADbasic* mit **Set_TiCo_RingBuffer** Daten in dem Ringspeicher schreiben, aktualisiert die *ADwin* CPU bei jedem Schreibvorgang den Schreibzeiger und kopiert den Wert in die globale Variable **Par_x** auf dem *TiCo*-Prozessor. Mit dem Wert in

Par_x kann **RingBuffer_Full** in *TiCoBasic* die Anzahl der belegten Elemente im Ringspeicher berechnen.

Siehe auch

(Deklaration), [RingBuffer_Empty](#), [Set_TiCo_RingBuffer](#) (*ADbasic*), [Kapitel 4.3.3 „Die Datenstruktur RingBuffer“](#)

Beispiel

```
REM 1007 LONG elements as read ringbuffer
Dim Data_12[1007] As Long As RingBuffer_For_Read

Init:
    Ringbuffer_Clear(12, Par_3)

Event:
    Rem read number of used elements in Data_12
    Rem (using Par_3)
    Par_1 = RingBuffer_Full(12, Par_3)
```

SelectCase

Die Kontrollstruktur **SelectCase** bewirkt in Abhängigkeit von einem Wert die Ausführung eines von mehreren Anweisungsblöcken.

Syntax

```
SelectCase var
Case const1a{, const1b, ...}
    ...                               'Anweisungsblock
CCase const2a{, const2b, ...}
    ...                               'Anweisungsblock
CaseElse
    ...                               'Anweisungsblock
EndSelect
```

Parameter

var	Auszuwertendes Argument (kein Berechnungs- ausdruck).	LONG
const1a, const1b, const2a, const2b	Wert von var (0...255), bei dem der nachfolgende Anweisungsblock ausgeführt wird.	CONST LONG

Bemerkungen

In einer Library-Funktion oder einem Library-Unterprogramm kann die Kontrollstruktur nicht verwendet werden.

Sie können mehrere **SelectCase**-Strukturen beliebig tief verschachteln; nur die Speichergröße begrenzt Sie hierbei.

Je nach Argument können Sie mit **SelectCase** verschachtelte **If**-Strukturen ersetzen und damit übersichtlicher gestalten; vor allem aber wird diese Struktur schneller ausgeführt als mehrere aufeinander folgende **If**-Strukturen.

Wenn das auszuwertende Argument mit keiner der **Case**-Konstanten übereinstimmt, wird – falls vorhanden – nur der **CaseElse**-Anwei-

sungsblock ausgeführt. Dies geschieht ebenfalls, wenn das auszuwertende Argument außerhalb des Wertebereichs der Konstanten liegt.

CCase steht für „Continue Case“: Wenn ein **Case**- oder **CCase**-Anweisungsblock abgearbeitet wurde, dann wird ein direkt folgender **CCase**-Anweisungsblock ebenfalls abgearbeitet.

Im Beispiel unten wird deshalb nicht nur die Anweisung **ADC (5)**, sondern auch **ADC (7)** ausgeführt. Wäre dagegen **Par_1 = 3**, dann würde nur **ADC (7)** ausgeführt.

Wenn Sie in den Anweisungsblöcken Variablen so verändern, dass sich der Wert des Arguments ändert, wird dies erst bei der nächsten **SelectCase**-Abfrage berücksichtigt.

Die Struktur verwendet intern eine Sprungtabelle im Datenspeicher (DM), deren Speicherbedarf sich nach der größten angegebenen **Case** / **CCase**-Konstanten richtet. Um den Speicherplatz-Bedarf zu beschränken, ist der Wertebereich der Konstanten auf 0...255 eingeschränkt. Es gilt:

Speicherbedarf in Bytes = [(größter Konstantenwert)+1] 4

Beispielsweise wäre der Speicherbedarf bei **Case 200**:

(200 + 1) 4 = 804 Bytes; der max. Bedarf beträgt genau 1 KiB.

Siehe auch

Do ... Until, For ... To ... {Step ... } Next, If ... Then ... {Else ... } EndIf

Beispiel**Event:**

```

Par_1 = 2
SelectCase Par_1      'Par_1 auswerten
Case 0                'Ist Par_1 = 0?
    Par_10 = ADC(1)    'ADC(1) auslesen
Case 1                'Ist Par_1 = 1?
    Par_10 = ADC(3)    'ADC(3) auslesen
Case 2                'Ist Par_1 = 2?
    Par_10 = ADC(5)    'ADC(5) und auch (durch
                        'CCase) ADC(7) auslesen
CCase 3                'Ist Par_1 = 3?
    Par_11 = ADC(7)    'ADC(7) auslesen
Case 4,5,6,7,16       'Ist Par_1 = 4, 5, 6, 7 oder 16?
    Rem digitale Eingänge einlesen
    Par_2 = Digin_Word()
CaseElse               'Par_1: sonstige Werte
    Rem Wert von Par_10 an digitale Ausgänge ausgeben
    Digout_Word(Par_10)
EndSelect              'Abschluss der Auswahl

```

Shift_Left

Shift_Left verschiebt alle Bits eines Werts um eine bestimmte Stellenzahl nach links. Rechts frei werdende Bits werden zu 0 gesetzt.

Syntax

```
ret_val = Shift_Left(val, num)
```

Parameter

<code>val</code>	Argument	LONG
<code>num</code>	Anzahl der Stellen, um die das Argument geschoben wird (0...31).	LONG
<code>ret_val</code>	Argument mit verschobenen Bits oder 0 für (<code>num < 0</code>) und für (<code>num > 31</code>)	LONG

Bemerkungen

Das Verschieben um n Stellen nach links entspricht einer Multiplikation mit 2^n . Ein eventuell vorkommender Überlauf wird nicht berücksichtigt, d. h. ein gesetztes Bit ist verloren, wenn es aus dem Argument nach links „heraus geschoben“ wird.

Die Ausführungszeit ist gleich derjenigen bei einem vergleichbaren Multiplikations-Operator.

Siehe auch

[Shift_Right](#)

Beispiel

```
Dim val_1, val_2 As Long
```

Event:

```
val_1 = 1024
```

```
val_2 = Shift_Left(val_1, 2) 'Ergebnis: val_2=4096
```

Shift_Right

Shift_Right verschiebt alle Bits eines Werts um eine bestimmte Stellenzahl nach rechts. Links frei werdende Bits werden zu 0 gesetzt.

Syntax

```
ret_val = Shift_Right(val, num)
```

Parameter

val	Argument	LONG
num	Anzahl der Stellen, um die das Argument geschoben wird (0...31).	LONG
ret_val	Argument mit verschobenen Bits oder 0 für (num < 0) und für (num > 31)	LONG

Bemerkungen

Falls das Argument eine positive Zahl ist, entspricht das Verschieben um n Stellen nach rechts einer Division durch 2^n . Ein eventuell vorkommender Divisions-Rest wird nicht berücksichtigt, d.h. ein gesetztes Bit, das aus dem Argument nach rechts „heraus geschoben“ wird, ist verloren.

Die Ausführungszeit ist geringer als bei einem vergleichbaren Divisions-Operator, d.h. `val_2 = Shift_Right(val_1, 3)` ist schneller als `val_2 = val_1 / 8`.

Siehe auch

[Shift_Left](#)

Beispiel

```
Dim val_1, val_2 As Long
```

Event:

```
val_1 = 1024
```

```
val_2 = Shift_Right(val_1, 3) 'Ergebnis: val_2=128
```

Sleep

Sleep lässt den Prozessor für eine definierte Zeit warten.

Syntax

```
Sleep(val)
```

Parameter

val Anzahl (≥ 1) der zu wartenden Zeiteinheiten in 100ns.

LONG

Bemerkungen

Da der Befehl **Sleep** als Zählschleife ausgeführt wird, kann er bei einem hochprioren Prozess nicht unterbrochen werden.

Verwenden Sie nach Möglichkeit eine Konstante als Argument. Wenn das Argument **val** eine Berechnung erfordert, benötigt dies eine bestimmte Zeitspanne zusätzlich; diese ist jeweils konstant und beträgt einige wenige Taktzyklen.

Folgende Fälle erfordern eine Berechnung:

- Das Argument ist ein Berechnungsausdruck mit Variablen oder Feldelementen.
- Die Variable im Argument ist für den Speicherbereich **DRAM_Extern** deklariert.
- Das Argument ist ein Feld.

Siehe auch

NOP, **NOPs**

Beispiel

```
Event:
Set_Mux(0)           'Multiplexer setzen
Sleep(25) '2.5 µs (=25*100ns) warten
                        '= Einschwingzeit des MUX
Start_Conv(1)        'Konvertierung starten
Rem ...
```

Für ungültige Argumente ist der Rückgabewert undefiniert.

Sub ... EndSub

`Sub...EndSub` definiert ein Unterprogramm-Makro mit Übergabeparametern.

Syntax

```
Sub macro_name({val_1, val_2, ...})
    {Dim var As <var_type>}
    ...
EndSub
```

'Anweisungsblock

Parameter

`macro_name` Name des Unterprogramms

`val_1, val_2` Namen der Übergabeparameter; FLOAT
für Felder ist die Syntax mit Dimensionsklammern LONG
erforderlich: `array[]` oder `Data_n[]`.

Bemerkungen

Allgemeine Informationen über Makros finden Sie in [Kapitel 4.5.1 auf Seite 115](#).

Diese Anweisung definiert ein Unterprogramm-Makro, d.h. der vollständige Anweisungsblock zwischen `Sub` und `EndSub` wird an der aufrufenden Stelle eingefügt.

Unterprogramm-Makros erhöhen die Übersichtlichkeit Ihres Quelltextes. Beachten Sie aber, dass auch jeder Aufruf des Unterprogramms die kompilierte Datei vergrößert.

Sie können Unterprogramme an 3 Stellen einfügen:

1. Vor dem Abschnitt `Init:`.
2. Nach dem Abschnitt `Finish:`.
3. In einer separaten Datei, die Sie mit `#Include` einbinden (aber nur an einer der Stellen, die unter 1. und 2. angegeben sind).

Beachten Sie bitte, dass Sie in Unterprogrammen:

- keine Prozess-Abschnitte wie **Init:**, **Event:**, oder **Finish:** definieren dürfen.
- am Anfang lokale Variablen definieren können, die nur innerhalb des Unterprogramms und für die Dauer der Abarbeitung verfügbar sind.

Dies gilt auch, wenn die Variablen den gleichen Namen haben wie Variablen außerhalb des Unterprogramms.

Bei Berechnungsausdrücken in einem Unterprogramm sollten die Übergabeparameter in Klammern stehen. Auf diese Weise vermeiden Sie Probleme mit der Rangfolge von Operatoren (z.B. Punkt- vor Strich-Rechnung).

Ein Unterprogramm wird mit seinem Namen und allen definierten Argumenten aufgerufen. Als Argument ist jeder Berechnungsausdruck (auch Felder) zulässig, solange er den passenden Datentyp hat. Wenn Sie keine Argumente definieren, müssen Sie dennoch beim Aufruf des Unterprogramms die Leerkammern verwenden: `name()`.

Wenn ein Feld (kein Feldelement) als Übergabeparameter eines Unterprogramms verwendet wird, ist die Syntax für Aufruf und Definition unterschiedlich:

- Unterprogramm-Aufruf *ohne* Dimensions-Klammern:
`subname(array_pass)`
- Unterprogramm-Definition *mit* Dimensions-Klammern:
`Sub subname(array_def[]) ...`

Werte werden an Feldelemente (eines Felds als Übergabeparameter) zugewiesen wie gewöhnlich:

```
array_def[2] = value
```

Wenn Sie einem Übergabeparameter `x` im Unterprogramm einen Wert zuweisen, darf beim Unterprogramm-Aufruf für `x` keine Konstante angegeben werden, sondern nur eine Variable oder ein einzelnes Feldelement. Auf diese Weise können Übergabeparameter auch einen Rückgabewert enthalten.

Siehe auch

`#Include, Function ... EndFunction`

Beispiel

- / -

XOr

Der Operator **XOr** (Exklusiv-Oder) verknüpft zwei ganzzahlige Werte bitweise.

Syntax

```
... val_1 XOr val_2 ...
```

Parameter

val_1, val_2 Ganzzahliger Wert

LONG

Siehe auch

[And](#), [Not](#), [Or](#)

Beispiel

```
Dim value As Long
```

```
Event:
```

```
value = 0100b XOr 0110b
```

```
Rem Ergebnis: value = (4 XOr 6) = 0010b = 2
```

7.3 Mathematik-Befehle

Die Include-Datei `Math_TiCo.inc` enthält zusätzliche Mathematik-Befehle, die über den Befehlssatz des *TiCoBasic*-Compilers hinaus gehen.

Mathematik-Befehle

Name	Funktion
<code>Mod</code>	Mod berechnet den ganzzahligen Rest bei einer Division von zwei ganzen Zahlen.

Mod

Mod berechnet den ganzzahligen Rest bei einer Division von zwei ganzen Zahlen.

Syntax

```
#Include Math_TiCo.inc
ret_val = Mod(x_param, y_param)
```

Parameter

x_param	Dividend.	LONG
y_param	Divisor.	LONG
ret_val	Rest aus der Division von x_param / y_param.	LONG

Bemerkungen

Die Division mit Rest arbeitet nach der symmetrischen Variante, bei der der Quotient durch Abschneiden der Nachkommastellen berechnet wird. Mit dieser Definition wird der Quotient zur Null hin gerundet und der Rest hat das gleiche Vorzeichen wie der Dividend.

Der ganzzahlige Rest bei einer Division durch Null ist gleich dem Dividend: **Mod**(x, 0) = x.

Die Ausführungszeit für die Modulo-Funktion beträgt beim Tico-1 bis zu 0,86µs (hohe Priorität).

Siehe auch

[/ \(Division\)](#), [Absl](#)

Beispiel

```
#Include Math_TiCo.inc
Par_1 = Mod(17, 3)      'Par_1 = 2
Par_2 = Mod(-9, 5)      'Par_2 = -4
Par_3 = Mod(72, Par_2)  'Par_3 = 3
```

7.4 Gold II: TiCo-Prozessor

Mit den folgenden *ADbasic*-Befehlen greift die *ADwin* CPU (Prozessor T11) auf den *TiCo*-Prozessor zu:

Datenzugriff Initialisieren	TDrv_Init
Globale Variablen lesen und schreiben	Get_Par , Get_Par_Block Set_Par , Set_Par_Block
Globale Felder lesen und schreiben	GetData_Long , SetData_Long
Ringspeicher lesen, schreiben und Status abfragen	Get_TiCo_RingBuffer , RingBuffer_Empty Set_TiCo_RingBuffer , RingBuffer_Full
Processdelay setzen und abfragen	TiCo_Get_Processdelay , TiCo_Set_Processdelay
Debug-Fehler abfragen	TiCo_Get_Process_Error
Prozesse starten und stoppen, Prozessor steuern	TiCo_Start_Process , TiCo_Stop_Process TiCo_Stop , TiCo_Start , TiCo_Restart , TiCo_Reset_Mode
Systeminformationen abfragen	Get_TiCo_Status , Process_Status , Workload
Binärdatei übertragen	TiCo_Flash , TiCo_Load

Beachten Sie bitte: Bei fast allen Befehlen muss zuerst die Datenübertragung mit **TDrv_Init** initialisiert werden.

Get_Par

Get_Par gibt den Wert einer globalen Variablen **par_x** von einem *TiCo*-Prozessor zurück.

Syntax

```
#Include ADwinGoldII.inc  
  
ret_val = Get_Par(tico_no, par_no)
```

Parameter

<code>tico_no</code>	Nummer (1) des <i>TiCo</i> -Prozessors.	LONG
<code>par_no</code>	Nummer (1...80) der globalen Variablen.	LONG
<code>ret_val</code>	Wert ($-2^{31} \dots +2^{31}-1$) der globalen Variablen.	LONG

Bemerkungen

Mehrere Werte werden mit **Get_Par_Block** schneller gelesen.

Die Nummer `tico_no` ist nicht zu verwechseln mit dem Typ des *TiCo*-Prozessors wie *TiCo1* oder *TiCo2* und muss derzeit den Wert 1 haben. Der Parameter ist für spätere Entwicklungen vorgesehen, wenn *ADwin*-Hardware mehrere *TiCo*-Prozessoren enthält.

Siehe auch

[Get_Par_Block](#), [GetData_Long](#), [Set_Par](#), [Set_Par_Block](#), [SetData_Long](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long 'settings array for data transfer
```

Init:

```
  TDrv_Init(tico_no,tset)
```

Event:

```
  REM read Par_1 from TiCo into Par_2 of ADwin CPU
  Par_2 = Get_Par(tico_no, 1)
  REM read Par_2 from TiCo
  Par_5 = 2
  Par_6 = Get_Par(tico_no, Par_5)
```

Get_Par_Block

Get_Par_Block liest eine Anzahl von globalen Variablen **Par_x** des *TiCo*-Prozessors und schreibt die Werte in ein Feld.

Syntax

```
#Include ADwinGoldII.inc

Get_Par_Block(tico_no, dest_array[],
              dest_array_idx, par_no, par_count)
```

Parameter

<code>tico_no</code>	Nummer (1) des <i>TiCo</i> -Prozessors.	LONG
<code>dest_array[]</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<code>dest_array_idx</code>	Ziel-Startindex (1...n): Feldelement, ab dem die Werte abgelegt werden.	LONG
<code>par_no</code>	Index (1...80) der ersten globalen Variablen, die gelesen wird.	LONG
<code>par_count</code>	Anzahl (1...80) der zu lesenden Variablen.	LONG

Bemerkungen

- / -

Siehe auch

[Get_Par](#), [GetData_Long](#), [Set_Par](#), [Set_Par_Block](#), [SetData_Long](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim Data_1[80] As Long
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    TDrv_Init(tico_no,tset)

Event:
    REM read Par_1...Par_80 from TiCo and write values to
    REM Data_1[1]...Data_1[80] of ADwin CPU
    Get_Par_Block(tico_no,Data_1,1,1,80)
```

Get_TiCo_RingBuffer

Get_TiCo_RingBuffer liest Werte aus einem Ringspeicher von einem *TiCo*-Prozessor und schreibt die Werte in ein Feld der *ADwin* CPU.

Syntax

```
#Include ADwinGoldII.inc

ret_val =
    Get_TiCo_RingBuffer(tdrv_datatable[],
        src_array_no, dest_array[], dest_array_idx,
        maxcount, flowrate, tico_par, struct)
```

Parameter

tdrv_ datatable []	Feld, das Einstellungen für die Datenübertragung enthält.	ARRAY LONG
src_ array_no	Nummer (1...16) des Schreib-Ringspeichers Data_n auf dem <i>TiCo</i> -Prozessor.	FLOAT
dest_ array []	Zielfeld, in das die Werte übertragen werden.	ARRAY LONG
dest_ array_idx	Index (1...n) des ersten Elements im Feld dest_array [], in das geschrieben wird.	LONG
maxcount	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
flowrate	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
tico_par	Kennzahl zum Übertragen des Lesezeigers zum <i>TiCo</i> -Prozessor. 1...80: Nummer (1...80) der globalen Variablen Par_n des <i>TiCo</i> -Prozessors, in die der aktuelle Lesezeigerwert geschrieben wird. 0: Der Lesezeiger wird nicht übertragen.	LONG

struct	Kennwert für Messwert-Sätze: 0: Beliebige Anzahl von Daten lesen. ≥0: Die Anzahl der übertragenen Daten muss ein Vielfaches von struct sein.	LONG
ret_val	Erfolgsstatus des Befehls: -1: Fehler, der Schreib-Ringspeicher des <i>TiCo</i> ist nicht korrekt deklariert. ≥0: Befehl war erfolgreich. Der Rückgabewert ist die Anzahl der übertragenen Werte.	LONG

Bemerkungen

Der Befehl liest nicht mehr als **maxcount** Daten. Wenn weniger Daten im Ringspeicher enthalten sind, werden alle vorhandenen Daten übertragen.

Beim Lesen von einem Ringspeicher speichert **Get_TiCo_Ringbuffer** die letzte Leseposition, den Lesezeiger, in dem Feld **tdrv_datatable[]**. Wenn der Befehl **Ringbuffer_Empty** des *TiCo*-Prozessors korrekt arbeiten soll, muss in **tico_par** die Nummer einer globalen Variablen angegeben werden. Dadurch wird der Wert des Lesezeigers in die globale Variable des *TiCo*-Prozessors übertragen.

Siehe auch

[Set_TiCo_RingBuffer](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

Get_TiCo_Status

Get_TiCo_Status gibt zurück, ob der *TiCo*-Prozessor aktiv ist.

Syntax

```
#Include ADwinGoldII.inc  
ret_val = Get_TiCo_Status()
```

Parameter

ret_val	Status des <i>TiCo</i> -Prozessors:	LONG
	0: Prozessor ist gestoppt.	
	1: Prozessor läuft.	
	-3: Fehler, kein <i>TiCo</i> -Prozessor vorhanden.	

Bemerkungen

- / -

Siehe auch

[Process_Status](#), [Workload](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

GetData_Long

GetData_Long liest Werte aus einem globalen Feld von einem *TiCo*-Prozessor und schreibt die Werte in ein Feld.

Syntax

```
#Include ADwinGoldII.inc

GetData_Long(tdrv_datatable[], src_array_no,
             src_array_idx, count, dest_array[],
             dest_array_idx, flowrate)
```

Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält.	LONG
<code>src_</code>		
<code>array_no</code>	Nummer (1...16) des globalen Felds <code>Data_x</code> auf dem <i>TiCo</i> -Prozessor.	LONG
<code>src_</code>		
<code>array_idx</code>	Index (1...n) des ersten Elements, das im globalen Feld <code>src_array_no</code> gelesen wird.	LONG
<code>count</code>	Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>dest_</code>		
<code>array[]</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<code>dest_</code>		
<code>array_idx</code>	Ziel-Startindex (1...n): Feldelement, ab dem die Werte im Zielfeld abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Es muss gewährleistet sein, dass

- das globale Feld `src_array_no` auf dem *TiCo*-Prozessor deklariert ist und
- das Feld `dest_array` mit mindestens `count` Elementen deklariert ist.

Siehe auch

[Get_Par](#), [Set_Par](#), [SetData_Long](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset_41[150] As Long   'settings array for data
                             'transfer
Dim Data_4[200] As Long

Init:
  REM initialize data transfer ADwin CPU <-> TiCo
  TDrv_Init(tico_no,tset_41)

Event:
  REM read 120 values from TiCo Data_2 (starting from
  REM Data_2[20]) into Data_4 of ADwin CPU (starting
  REM from index 5). flowrate is high.
  GetData_Long(tset_41,2,20,120,Data_4,5,3)
```

Process_Status

Process_Status gibt den Status eines Prozesses auf einem *TiCo*-Prozessor zurück.

Syntax

```
#Include ADwinGoldII.inc

ret_val = Process_Status(tico_no, process_no)
```

Parameter

<code>tico_no</code>	Nummer (1) des <i>TiCo</i> -Prozessors.	LONG
<code>process_no</code>	Nummer (1...4) des <i>TiCo</i> -Prozesses.	LONG
<code>ret_val</code>	Status des Prozesses: ≠1: Prozess läuft. 0: Prozess läuft nicht, d.h. er ist nicht geladen, nicht gestartet oder gestoppt.	LONG

Bemerkungen

- / -

Siehe auch

[Get_TiCo_Status](#), [TiCo_Get_Processdelay](#), [TiCo_Set_Processdelay](#),
[TiCo_Start_Process](#), [TiCo_Stop_Process](#), [Workload](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim Data_4[200] As Long
Dim tset[150] As Long      'settings array for data
                           'transfer
```

Init:

```
  TDrv_Init(tico_no,tset)
```

Event:

```
  REM read status of TiCo process 1
  Par_1 = Process_Status(tico_no,1)
  REM if process is running, read TiCo Par_5
  If (Par_1 = 1) Then
    Par_2 = Get_Par(tico_no,5)
  EndIf
```

RingBuffer_Empty

RingBuffer_Empty gibt die Anzahl der freien Elemente in einem Schreib-Ringspeicher eines *TiCo*-Prozessors zurück.

Syntax

```
#Include ADwinGoldII.inc

ret_val = RingBuffer_Empty(tdrv_datatable[],
    Data_no)
```

Parameter

tdrv_ datatable []	Feld, das Einstellungen für die Datenübertragung enthält.	ARRAY LONG
Data_no	Nummer (1...16) des Schreib-Ringspeichers Data_n auf dem <i>TiCo</i> -Prozessor.	LONG
ret_val	Anzahl der freien Elemente im Schreib-Ring-speicher.	LONG

Bemerkungen

Bei **Get_TiCo_RingBuffer** ist keine vorherige Abfrage mit **RingBuffer_Empty** erforderlich.

Siehe auch

[Get_TiCo_RingBuffer](#), [RingBuffer_Full](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

RingBuffer_Full

RingBuffer_Full gibt die Anzahl der genutzten Elemente in einem Lese-Ringspeicher eines *TiCo*-Prozessors zurück.

Syntax

```
#Include ADwinGoldIII.inc

ret_val = RingBuffer_Full(tdrv_datatable[],
    Data_no)
```

Parameter

tdrv_ datatable []	Feld, das Einstellungen für die Datenübertragung enthält.	ARRAY <u>LONG</u>
Data_no	Nummer (1...16) des Lese-Ringspeichers Data_n auf dem <i>TiCo</i> -Prozessor.	<u>LONG</u>
ret_val	Anzahl (0...n) der genutzten Elemente im Lese-Ringspeicher.	<u>LONG</u>

Bemerkungen

Bei **Set_TiCo_RingBuffer** ist keine vorherige Abfrage mit **RingBuffer_FULL** erforderlich.

Siehe auch

[Set_TiCo_RingBuffer](#), [RingBuffer_Empty](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

Set_Par

Set_Par setzt den Wert einer globalen Variablen **Par_x** auf einem *TiCo*-Prozessor.

Syntax

```
#Include ADwinGoldII.inc
Set_Par(tico_no, par_no, value)
```

Parameter

<code>tico_no</code>	Nummer (1) des <i>TiCo</i> -Prozessors.	LONG
<code>par_no</code>	Nummer (1...80) der globalen Variablen.	LONG
<code>value</code>	Wert ($-2^{31} \dots +2^{31}-1$) der globalen Variablen.	LONG

Bemerkungen

Set_Par setzt den Wert der globalen Variablen unabhängig davon, ob gerade ein *TiCo*-Prozess läuft. Da *ADwin* CPU- und *TiCo*-Prozesse nicht synchron arbeiten, können sich die Werte globaler Variablen für den *TiCo*-Prozess völlig unerwartet während der Laufzeit ändern.

Falls erforderlich, empfehlen wir eine Synchronisierung von *ADwin* CPU und *TiCo* mit Hilfe eines Software-Handshakes.

Siehe auch

[Get_Par](#), [Get_Par_Block](#), [GetData_Long](#), [Set_Par_Block](#), [SetData_Long](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.Inc
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                             'transfer

Init:
  TDrv_Init(tico_no,tset)
  Par_5=200

Event:
  REM write value of Par_5 (ADwin CPU) to TiCo Par_2
  Set_Par(tico_no,2,Par_5)
```

Set_Par_Block

Set_Par_Block schreibt Werte aus einem Feld in eine Anzahl von globalen Variablen `Par_x` des *TiCo*-Prozessors.

Syntax

```
#Include ADwinGoldII.inc

Set_Par_Block(tico_no, src_array[],
              src_array_idx, par_no, par_count)
```

Parameter

<code>tico_no</code>	Nummer (1) des <i>TiCo</i> -Prozessors.	LONG
<code>src_array[]</code>	Quellfeld, aus dem Daten übertragen werden.	ARRAY
<code>src_array_idx</code>	Indes des Feldelements, ab dem Werte aus dem Quellfeld gelesen werden.	LONG
<code>par_no</code>	Index (1...80) der ersten globalen <i>TiCo</i> -Variablen, die beschrieben wird.	LONG
<code>par_count</code>	Anzahl der zu übertragenden Werte.	LONG

Bemerkungen

Set_Par_Block setzt die Wert der globalen Variablen unabhängig davon, ob gerade ein *TiCo*-Prozess läuft. Da *ADwin* CPU- und *TiCo*-Prozesse nicht synchron arbeiten, können sich die Werte globaler Variablen für den *TiCo*-Prozess völlig unerwartet während der Laufzeit ändern.

Falls erforderlich, empfehlen wir eine Synchronisierung von *ADwin* CPU und *TiCo* mit Hilfe eines Software-Handshakes.

Siehe auch

[Get_Par](#), [Get_Par_Block](#), [GetData_Long](#), [Set_ParSetData_Long](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.Inc
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long 'array for data transfer settings
Dim Data_1[40] As Long
Dim i As Long
```

Init:

```
TDrv_Init(tico_no,tset)
For i = 1 To 40
    Data_1[i] = i*10
Next
Par_15=1111
Par_16=2222
Par_17=3333
Par_18=4444
Par_19=5555
```

Event:

```
REM write 40 values from Data_1 (starting from Data_1[1])
REM into Par_1...Par_40 (TiCo)
Set_Par_Block(tico_no,Data_1,1,1,40)
```

Set_TiCo_RingBuffer

Set_TiCo_RingBuffer schreibt Werte aus einem Feld der ADwin CPU in einen Ringspeicher auf einem *TiCo*-Prozessor.

Syntax

```
#Include ADwinGoldII.inc

ret_val =
    Set_TiCo_RingBuffer(tdrv_datatable[],
        dest_array_no, src_array[], src_array_idx,
        maxcount, flowrate, tico_par, struct)
```

Parameter

<code>tdrv_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung enthält.	ARRAY LONG
<code>dest_array_no</code>	Nummer (1...16) des Lese-Ringspeichers Data_n auf dem <i>TiCo</i> -Prozessor.	FLOAT
<code>src_array[]</code>	Quellfeld, aus dem die Werte übertragen werden.	ARRAY LONG
<code>src_array_idx</code>	Index (1...n) des ersten Elements im Feld <code>src_array[]</code> , das gelesen wird.	LONG
<code>maxcount</code>	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>flowrate</code>	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>tico_par</code>	Kennzahl zum Übertragen des Schreibzeigers zum <i>TiCo</i> -Prozessor. 1...80: Nummer (1...80) der globalen Variablen Par_n des <i>TiCo</i> -Prozessors, in die der aktuelle Schreibzeiger geschrieben wird. 0: Der Schreibzeiger wird nicht übertragen.	LONG

struct	Kennwert für Messwert-Sätze: 0: Beliebige Anzahl von Daten schreiben. >0: Die Anzahl der übertragenen Daten muss ein Vielfaches von struct sein.	LONG
ret_val	Erfolgsstatus des Befehls: -1: Fehler, der Lese-Ringspeicher des <i>TiCo</i> ist nicht korrekt deklariert. ≥0: Befehl war erfolgreich. Der Rückgabewert ist die Anzahl der übertragenen Werte.	LONG

Bemerkungen

Der Befehl schreibt nicht mehr als **maxcount** Daten. Wenn weniger freie Elemente im Ringspeicher vorhanden sind, werden nur die freien Elemente im Ringspeicher beschrieben.

Beim Schreiben in einen Ringspeicher speichert **Set_TiCo_Ringbuffer** die letzte Schreibposition, den Schreibzeiger, in dem Feld **tdrv_datatable[]**. Wenn der Befehl **Ringbuffer_Full** des *TiCo*-Prozessors korrekt arbeiten soll, muss in **tico_par** die Nummer einer globalen Variablen angegeben werden. Dadurch wird der Wert des Lesezeigers in die globale Variable des *TiCo*-Prozessors übertragen.

Siehe auch

[Get_TiCo_RingBuffer](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

SetData_Long

SetData_Long liest Werte aus einem Feld und schreibt sie in ein globales Feld eines *TiCo*-Prozessors.

Syntax

```
#Include ADwinGoldII.inc

SetData_Long(tdrv_datatable[], dest_array_no,
             dest_array_idx, count, src_array[],
             src_array_idx, flowrate)
```

Parameter

<code>tdrv_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung enthält.	ARRAY LONG
<code>dest_array_no</code>	Nummer (1...16) des globalen Felds <code>Data_x</code> auf dem <i>TiCo</i> -Prozessor.	LONG
<code>dest_array_idx</code>	Index (1...n) des ersten Elements, das im globalen Feld <code>dest_array_no</code> beschrieben wird.	LONG
<code>count</code>	Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>src_array[]</code>	Quellfeld, aus dem die Werte gelesen werden.	ARRAY LONG
<code>src_array_idx</code>	Quell-Startindex (1...n): Feldelement, ab dem die Werte gelesen werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Es muss gewährleistet sein, dass

- das globale Feld `dest_array_no` auf dem *TiCo*-Prozessor deklariert ist und
- das Feld mit mindestens `count` Elementen deklariert ist.

Siehe auch

[Get_Par](#), [GetData_Long](#), [Set_Par](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

-/-

Beispiel

```

#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                                'transfer

Dim Data_1[150] As Long
Dim tset_41[150] As Long
Dim i As Long

Init:
    REM initialize data transfer ADwin CPU <-> TiCo
    TDrv_Init(tico_no,tset_41)
    For i = 1 To 80
        Data_1[i] = i
    Next

Event:
    REM read 120 values from TiCo Data_2 (starting from
    REM Data_2[1]) into Data_1 of ADwin CPU (starting
    REM from Data_1[5]). flowrate is high.
    SetData_Long(tset_41,2,1,120,Data_1,5,3)

```

TDrv_Init

TDrv_Init initialisiert die Datenübertragung zwischen *ADwin* CPU und einem *TiCo*-Prozessor.

Syntax

```
#Include ADwinGoldIII.inc  
  
REM define settings array for TiCo x  
Dim tdrv_datatable[150] As Long  
  
TDrv_Init(tico_no, tdrv_datatable[])
```

Parameter

<code>tico_no</code>	Nummer (1) des <i>TiCo</i> -Prozessors.	LONG
<code>tdrv_datatable</code>	Feld, das Einstellungen für die Datenübertragung aufnimmt.	ARRAY
<code>[]</code>		LONG

Bemerkungen

Der Befehl muss vor der Datenübertragung zwischen *ADwin* CPU und *TiCo* ausgeführt werden. Der Befehl sollte im Abschnitt **Init:** stehen.

Fast alle Befehle, die auf den *TiCo* zugreifen, benötigen eine Initialisierung der Datenübertragung.

Die Initialisierung muss für jeden *TiCo*-Prozessor separat durchgeführt werden. Für jeden *TiCo*-Prozessor muss ein Feld `tdrv_datatable[]` mit 150 Elementen angelegt werden.

Das Feld `tdrv_datatable[]` wird von den Befehlen **GetData_Long**, **SetData_Long** und **Workload** verwendet.

Siehe auch

[Get_Par](#), [Get_Par_Block](#), [TiCo_Get_Processdelay](#), [GetData_Long](#), [Set_Par](#), [Set_Par_Block](#), [TiCo_Set_Processdelay](#), [SetData_Long](#)

Gültig für

Gold II

Pinbelegungen

-/-

Beispiel

```
#Include ADwinGoldII.inc
#Define tico_no 1          'TiCo no.
Dim Data_4[200] As Long
Dim Data_7[1000] As Long
Dim tset[150] As Long
```

Init:

```
REM initialize data transfer ADwin CPU <-> TiCo
TDrv_Init(tico_no,tset)
```

Event:

```
REM read 120 values from TiCo Data_2 (starting from
REM Data_2[20]) into ADwin CPU Data_4 (starting from
REM index 5).
REM flowrate is 3 (high)
GetData_Long(tset,2,20,120,Data_4,5,3)
REM read 800 values from TiCo Data_5 (starting from
REM Data_7[9]) into Data_5 of ADwin CPU (starting from
REM index 1).
REM Flowrate is 3 (high)
GetData_Long(tset,5,9,800,Data_7,1,3)
```

TiCo_Flash

TiCo_Flash überträgt eine *TiCoBasic*-Binärdatei aus einem Feld in den Flash-Speicher eines *TiCo*-Prozessors.

Syntax

```
#Include ADwinGoldIII.inc

ret_val = TiCo_Flash(tico_no, array[])
```

Parameter

<code>tico_no</code>	Nummer (1) des <i>TiCo</i> -Prozessors.	LONG
<code>array[]</code>	Feld, in dem die <i>TiCoBasic</i> -Binärdatei enthalten ist.	ARRAY
<code>ret_val</code>	Status der Datenübertragung: 0: Datenübertragung erfolgreich. -1: Fehler: Befehl darf nur bei niedriger Priorität verwendet werden. 2: Fehler: Programmspeicher ist zu klein. 3: Fehler: Datenspeicher ist zu klein. 4: Fehler: Programm- und Datenspeicher sind zu klein. 5: Fehler: Falsches Passwort. 6: Fehler: Externer Speicher ist zu klein. 7: Fehler: Flash-Speicher (EEPROM) ist zu klein. 12: Fehler: Die Binärdatei ist nicht für den <i>TiCo</i> -Prozessor geeignet.	LONG

Bemerkungen

Verwenden Sie **TiCo_Flash** nur in niederpriorien Prozessen.

Der Befehl **TiCo_Flash** dient dazu, eine *TiCoBasic*-Binärdatei – ein kompiliertes *TiCo*-Programm – in den Flash-Speicher eines *TiCo*-Prozessors zu übertragen. Dazu sind folgende Schritte erforderlich:

- Erzeugen Sie in der Oberfläche *TiCoBasic* eine Binärdatei mit Build ► Make Bin File.
- Übertragen Sie die Binärdatei in ein globales Feld der *ADwin* CPU. Geeignet ist die Funktion `File2Data` (bzw. `LoadFrom-`

File), die in mehreren Programmiersprachen zur Verfügung steht.

- Übertragen Sie die Daten im globalen Feld `array[]` mit dem Befehl `TiCo_Flash` in den Flash-Speicher.
- Wenn der Bootloader aktiviert ist, startet der Prozess der *TiCo-Basic*-Binärdatei automatisch nach dem nächsten Einschalten der *ADwin*-Hardware.

Wenn das Feld `array[]` keine *TiCoBasic*-Binärdatei enthält, ist der Rückgabewert des Befehls ungültig.

Siehe auch

[TiCo_Load](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

TiCo_Get_Processdelay

TiCo_Get_Processdelay gibt das Processdelay (die Zykluszeit) eines Prozesses auf einem *TiCo*-Prozessor zurück.

Syntax

```
#Include ADwinGoldIII.inc  
  
ret_val =  
    TiCo_Get_Processdelay(tdrv_datatable[],  
        process_no)
```

Parameter

<code>tdrv_datatable</code>	Feld, das Einstellungen für die Datenübertragung enthält.	ARRAY LONG []
<code>process_no</code>	Nummer (1...4) des <i>TiCo</i> -Prozesses.	LONG
<code>ret_val</code>	Aktuell eingestellte Zykluszeit ($-2^{31} \dots +2^{31}-1$) des <i>TiCo</i> -Prozessors in Taktzyklen. Ein Taktzyklus dauert 20ns.	LONG

Bemerkungen

Bei einem zeitgesteuerten Prozess wird der Abschnitt **Event:** vom internen Zähler zyklisch und in festen Zeitabständen aufgerufen. Der Zeitabstand zwischen zwei Aufrufen, Zykluszeit oder Processdelay genannt, wird in Zähler-Taktzyklen gezählt.

Siehe auch

[Process_Status](#), [TiCo_Set_Processdelay](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    TDrv_Init(tico_no, tset)

Event:
    REM read processdelay of process 1 into Par_1
    Par_1 = TiCo_Get_Processdelay(tset, 1)
```

TiCo_Get_Process_Error

TiCo_Get_Process_Error gibt den zuletzt aufgetretenen Fehler des angegebenen Prozesses auf einem *TiCo*-Prozessor zurück.

Syntax

```
#Include ADwinGoldIII.inc

ret_val = TiCo_Get_Process_Error(tdrv_datatable[],
                                process_no)
```

Parameter

<code>tdrv_datatable</code>	Feld, das Einstellungen für die Datenübertragung enthält.	ARRAY _LONG
<code>process_no</code>	Nummer (1...4) des <i>TiCo</i> -Prozesses.	_LONG
<code>ret_val</code>	Nummer des zuletzt aufgetretenen Fehlers im Prozess: 0: kein Fehler 10: Zugriff auf eine zu große Elementnummer eines globalen Felds. 11: Zugriff auf eine zu kleine Elementnummer (≤ 0) eines globalen Felds. 12: Zugriff auf eine zu große Elementnummer eines lokalen Felds. 13: Zugriff auf eine zu kleine Elementnummer (≤ 0) eines lokalen Felds.	_LONG

Bemerkungen

Der Rückgabewert ist nur definiert, wenn der Debug-Modus auf dem *TiCo*-Prozessor eingeschaltet ist (siehe [Option Debug mode](#), Seite 72).

Siehe auch

[Process_Status](#), [TiCo_Get_Processdelay](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    TDrv_Init(tico_no, tset)

Event:
    REM read process error of process 1 into Par_1
    Par_1 = TiCo_Get_Process_Error(tset, 1)
```

TiCo_Load

TiCo_Load überträgt eine *TiCoBasic*-Binärdatei aus einem Feld in den Speicher eines *TiCo*-Prozessors und startet den Prozess.

Syntax

```
#Include ADwinGoldIII.inc
ret_val = TiCo_Load(tico_no, array[])
```

Parameter

<code>tico_no</code>	Nummer (1) des <i>TiCo</i> -Prozessors.	LONG
<code>array[]</code>	Feld, in dem die <i>TiCoBasic</i> -Binärdatei enthalten ist.	ARRAY LONG
<code>ret_val</code>	Status der Datenübertragung: 0: Datenübertragung erfolgreich. -1: Fehler: Befehl darf nur bei niedriger Priorität verwendet werden. 1: Fehler: Ungültige Prozessornummer. 2: Fehler: Programmspeicher ist zu klein. 3: Fehler: Datenspeicher ist zu klein. 4: Fehler: Programm- und Datenspeicher sind zu klein. 5: Fehler: Falsches Passwort. 6: Fehler: Zugriff auf externen Speicher ist nicht möglich. 10: Fehler: Der benötigte externe SRAM-Speicher ist zu klein. 11: Fehler: Das Feld enthält keine gültige <i>TiCoBasic</i> -Binärdatei. 12: Fehler: Die Binärdatei ist nicht für den <i>TiCo</i> -Prozessor geeignet.	LONG

Bemerkungen

Verwenden Sie **TiCo_Load** nur in niederprioren Prozessen.

Der Befehl **TiCo_Load** dient dazu, eine *TiCoBasic*-Binärdatei – ein kompiliertes *TiCo*-Programm – in den *TiCo*-Prozessor zu übertragen. Dazu sind folgende Schritte erforderlich:

- Erzeugen Sie in der Oberfläche *TiCoBasic* eine Binärdatei mit Build ► Make Bin File.
- Übertragen Sie die Binärdatei in ein globales Feld des *ADwin*-Prozessors. Geeignet ist die Funktion `File2Data` (bzw. `LoadFromFile`), die in mehreren Programmiersprachen zur Verfügung steht.
- Übertragen Sie die Daten im globalen Feld `array[]` mit dem Befehl **TiCo_Load** auf den *TiCo*-Prozessor.
- Der Prozess der *TiCoBasic*-Binärdatei wird automatisch gestartet.

Wenn das Feld `array[]` keine *TiCoBasic*-Binärdatei enthält, ist der Rückgabewert des Befehls ungültig.

Siehe auch

[TiCo_Flash](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

TiCo_Restart

TiCo_Restart stoppt den *TiCo*-Prozessor und startet ihn anschließend wieder.

Syntax

```
#Include ADwinGoldII.inc  
TiCo_Restart()
```

Parameter

- / -

Bemerkungen

Das Stoppen beendet auf dem *TiCo*-Prozessor sofort jeden Prozess sowie den Zähler. Die Daten werden durch das Stoppen nicht verändert.

Siehe auch

[TiCo_Reset_Mode](#), [TiCo_Stop](#), [TiCo_Start](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

TiCo_Reset_Mode

TiCo_Reset_Mode stellt ein, ob beim *TiCo*-Prozessor ein Reset durchgeführt wird, wenn die *ADwin* CPU (T11) gebootet wird.

Syntax

```
#Include ADwinGoldII.inc
TiCo_Reset_Mode(mode)
```

Parameter

mode	Gewünschter Reset-Modus beim Booten des T11. LONG
	0: Betriebszustand des <i>TiCo</i> -Prozessors bleibt unverändert. Default.
	1: Der <i>TiCo</i> -Prozessor wird gestoppt und neu gestartet.

Bemerkungen

Der Betriebsmodus **mode**=1 hat nur eine Bedeutung, wenn der *TiCo*-Prozessor bereits läuft.

Siehe auch

[TiCo_Restart](#), [TiCo_Stop](#), [TiCo_Start](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.inc
INIT:
    TiCo_Reset_Mode(1)           'reset TiCo processor with T11
                                'boot
```


TiCo_Set_Processdelay

TiCo_Set_Processdelay setzt das Processdelay (die Zykluszeit) eines Prozesses auf einem *TiCo*-Prozessor.

Syntax

```
#Include ADwinGoldIII.inc

TiCo_Set_Processdelay(tdrv_
    datatable[], process_no,
    value)
```

Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält.	LONG
<code>process_no</code>	Nummer (1...4) des <i>TiCo</i> -Prozesses.	LONG
<code>value</code>	Einzustellender Wert für das Processdelay.	LONG

Bemerkungen

- / -

Siehe auch

[TiCo_Get_Processdelay](#), [Process_Status](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  TDrv_Init(tico_no, tset)
  Processdelay = 6000       'set cycle time

Event:
  REM set TiCo processdelay to run with same cycle time as
  REM the process of the ADwin processor (T11)
  TiCo_Set_Processdelay(tset, 1, Processdelay/6)
```

TiCo_Start

TiCo_Start startet den *TiCo*-Prozessor.

Syntax

```
#Include ADwinGoldII.inc  
TiCo_Start()
```

Parameter

- / -

Bemerkungen

- / -

Siehe auch

[TiCo_Stop](#), [TiCo_Restart](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

TiCo_Start_Process

TiCo_Start_Process startet einen Prozess auf einem *TiCo*-Prozessor.

Syntax

```
#Include ADwinGoldII.inc

TiCo_Start_Process(tdrv_datatable[] ,
    process_no)
```

Parameter

tdrv_ datatable []	Feld, das Einstellungen für die Datenübertragung enthält.	ARRAY <u>LONG</u>
process_ no	Nummer (1...4) des <i>TiCo</i> -Prozesses.	<u>LONG</u>

Bemerkungen

Der Prozess muss bereits auf dem *TiCo*-Prozessor vorhanden sein.

Siehe auch

[TiCo_Get_Processdelay](#), [Process_Status](#), [TiCo_Set_Processdelay](#),
[TiCo_Start_Process](#), [TiCo_Stop_Process](#), [Workload](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
Dim tset[150] As Long      'settings array for data
                           'transfer

#Define tico_no 1          'TiCo no.

Init:
  Par_1 = 0
  Processdelay = 100000
  REM start TiCo process in parallel to ADwin CPU
  REM process
  TDrv_Init(tico_no,tset)
  TiCo_Start_Process(tset,1)

Event:
  REM ... program

Finish:
  REM stop TiCo process in parallel to ADwin CPU
  REM process
  TiCo_Stop_Process(tset,1)
```

TiCo_Stop

TiCo_Stop stoppt den *TiCo*-Prozessor.

Syntax

```
#Include ADwinGoldII.inc  
TiCo_Stop()
```

Parameter

- / -

Bemerkungen

Das Stoppen beendet auf dem *TiCo*-Prozessor sofort jeden Prozess sowie den Zähler. Die Daten werden durch das Stoppen nicht verändert.

Siehe auch

[TiCo_Start](#), [TiCo_Restart](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

- / -

TiCo_Stop_Process

TiCo_Stop_Process stoppt einen Prozess auf einem *TiCo*-Prozessor.

Syntax

```
#Include ADwinGoldIII.inc  
  
TiCo_Stop_Process (tdrv_datatable[] ,  
                  process_no)
```

Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält.	LONG
<code>process_</code>	Nummer (1...4) des <i>TiCo</i> -Prozesses.	LONG
<code>no</code>		

Bemerkungen

Der Prozess muss auf dem *TiCo*-Prozessor vorhanden sein.

Siehe auch

[TiCo_Get_Processdelay](#), [Process_Status](#), [TiCo_Set_Processdelay](#),
[TiCo_Start_Process](#), [Workload](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldIII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  Par_1 = 0
  Processdelay = 100000
  REM start TiCo process in parallel to ADwin CPU process
  TDrv_Init(tico_no,tset)
  TiCo_Start_Process(tset,1)

Event:
  REM ... program

Finish:
  REM stop TiCo process
  TiCo_Stop_Process(tset,1)
```


Workload

Workload gibt die Auslastung eines *TiCo*-Prozessors zurück.

Syntax

```
#Include ADwinGoldIII.inc  
ret_val = Workload(tdrv_datatable[])
```

Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält.	LONG
<code>[]</code>		
<code>ret_val</code>	Prozessor-Auslastung in Prozent (0.0 ... 100.0) oder Fehlerwert: <0: <i>TiCo</i> -Prozessor ist gestoppt oder kein <i>TiCo</i> -Prozessor vorhanden.	FLOAT

Bemerkungen

Der Rückgabewert ist die mittlere Prozessorauslastung in der Zeit zwischen dem vorherigen und dem aktuellen Aufruf von **Workload**. Beim ersten Aufruf in einem Programm ist der Rückgabewert daher ungültig.

Der kürzeste Zeitabstand zwischen zwei Befehlsaufrufen sollte mindestens das 100fache des **Processdelay** betragen. Anderenfalls kann der Rückgabewert einen Fehler von mehr als 1% haben.

Wenn der vorherige Aufruf von **Workload** länger als 85 Sekunden zurück liegt, ist der Rückgabewert ungültig. Rufen Sie den Befehl dann ein zweites Mal auf, um einen gültigen Rückgabewert zu erhalten.

Siehe auch

[TiCo_Get_Processdelay](#), [Process_Status](#), [TiCo_Set_Processdelay](#),
[TiCo_Stop_Process](#), [TDrv_Init](#)

Gültig für

Gold II

Pinbelegungen

- / -

Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    TDrv_Init(tico_no,tset)

Event:
    REM read TiCo workload
    FPar_1 = Workload(tset)
```

7.5 Pro II: TiCo-Prozessor

Mit den folgenden *ADbasic*-Befehlen greift die *ADwin* CPU auf den *TiCo*-Prozessor zu:

Datenzugriff Initialisieren	P2_TDrv_Init, P2_Get_TiCo_Status
Globale Variablen lesen und schreiben	P2_Get_Par, P2_Get_Par_Block P2_Set_Par, P2_Set_Par_Block
Globale Felder lesen und schreiben	P2_GetData_Long, P2_SetData_Long
Ringspeicher lesen, schreiben und Status abfragen	P2_Get_TiCo_RingBuffer, P2_RingBuffer_Empty P2_Set_TiCo_RingBuffer, P2_RingBuffer_Full
Processdelay setzen und abfragen	P2_TiCo_Get_Processdelay, P2_TiCo_Set_Processdelay
Debug-Fehler abfragen	P2_TiCo_Get_Process_Error
Prozesse starten und stoppen, Prozessor steuern	P2_TiCo_Start_Process, P2_TiCo_Stop_Process P2_TiCo_Stop, P2_TiCo_Start, P2_TiCo_Restart
Systeminformationen abfragen	P2_Process_Status, P2_Workload
Binärdatei übertragen	P2_TiCo_Flash, P2_TiCo_Load

Beachten Sie bitte: Bei fast allen Befehlen muss zuerst die Datenübertragung mit **P2_TDrv_Init** initialisiert werden.

P2_Get_Par

P2_Get_Par gibt den Wert einer globalen Variablen **Par_x** von einem TiCo-Prozessor des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Get_Par(module, tico_no, par_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1) des TiCo-Prozessors auf dem Modul (nicht Typ).	LONG
par_no	Nummer (1...80) der globalen Variablen.	LONG
ret_val	Wert ($-2^{31} \dots +2^{31}-1$) der globalen Variablen.	LONG

Bemerkungen

Mehrere Werte werden mit **P2_Get_Par_Block** schneller gelesen.

Siehe auch

[P2_Get_Par_Block](#), [P2_GetData_Long](#), [P2_Set_Par](#), [P2_Set_Par_Block](#), [P2_SetData_Long](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    P2_TDrv_Init(module,tico_no,tset)

Event:
    REM read Par_1 from TiCo and write value to Par_2 of ADwin
    REM CPU
    Par_2 = P2_Get_Par(module,tico_no,1)
```

P2_Get_Par_Block

P2_Get_Par_Block liest eine Anzahl von globalen Variablen **Par_x** des TiCo-Prozessors vom angegebenen Modul und schreibt die Werte in ein Feld.

Syntax

```
#Include ADwinPro_All.inc

P2_Get_Par_Block(module, tico_no, dest_array[],
                 dest_array_idx, par_no, par_count)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1) des TiCo-Prozessors auf dem Modul (nicht Typ).	LONG
dest_array[]	Zielfeld, in das die Werte übertragen werden.	ARRAY
dest_array_idx	Ziel-Startindex (1...n): Feldelement, ab dem die Werte abgelegt werden.	LONG
par_no	Index (1...80) der ersten globalen Variablen, die gelesen wird.	LONG
par_count	Anzahl (1...80) der zu lesenden Variablen.	LONG

Bemerkungen

- / -

Siehe auch

[P2_Get_Par](#), [P2_GetData_Long](#), [P2_Set_Par](#), [P2_Set_Par_Block](#), [P2_SetData_Long](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-

TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim Data_1[80] As Long
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    P2_TDrv_Init(module,tico_no,tset)

Event:
    REM read Par_1...Par_80 from TiCo and write values to
    REM Data_1[1]...Data_1[80] of ADwin CPU
    P2_Get_Par_Block(module,tico_no,Data_1,1,1,80)
    REM read Par_20...Par_29 from TiCo and write values to
    REM Par_5...Par_14 of ADwin CPU
    P2_Get_Par_Block(module,tico_no,PAR,5,20,10)
```

P2_Get_TiCo_Bootloader_Status

P2_Get_TiCo_Bootloader_Status gibt den Status des *TiCo*-Bootloaders auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Get_TiCo_Bootloader_Status(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Status des Bootloaders: 0: Bootloader ist ausgeschaltet. 1: Bootloader ist eingeschaltet. -1: Fehler; Befehl bei hoher Priorität genutzt. -2: Fehler; Modul nicht ansprechbar (Timeout). -3: Fehler; Kein <i>TiCo</i> -Prozessor auf dem Modul.	LONG

Bemerkungen

Der Befehl kann nur bei niedriger Prozesspriorität ausgeführt werden.

Siehe auch

[P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_Get_TiCo_RingBuffer

P2_Get_TiCo_RingBuffer liest Werte aus einem Ringspeicher von einem TiCo-Prozessor und schreibt die Werte in ein Feld der ADwin CPU.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Get_TiCo_RingBuffer(tdrv_datatable[],
    src_array_no, dest_array[], dest_array_idx,
    maxcount, flowrate, tico_par, struct)
```

Parameter

<code>tdrv_</code> <code>datatable</code> <code>[]</code>	Feld, das Einstellungen für die Datenübertragung enthält, u.a. Moduladresse und TiCo-Nummer.	ARRAY LONG
<code>src_</code> <code>array_no</code>	Nummer (1...16) des Schreib-Ringspeichers <code>Data_n</code> auf dem TiCo-Prozessor.	FLOAT
<code>dest_</code> <code>array[]</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY LONG
<code>dest_</code> <code>array_idx</code>	Index (1...n) des ersten Elements im Feld <code>dest_array[]</code> , in das geschrieben wird.	LONG
<code>maxcount</code>	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>tico_par</code>	Kennzahl zum Übertragen des Lesezeigers zum TiCo-Prozessor. 1...80: Nummer (1...80) der globalen Variablen <code>Par_n</code> des TiCo-Prozessors, in die der aktuelle Lesezeigerwert geschrieben wird. 0: Der Lesezeiger wird nicht übertragen.	LONG

struct	Kennwert für Messwert-Sätze: 0: Beliebige Anzahl von Daten lesen. >0: Die Anzahl der übertragenen Daten muss ein Vielfaches von struct sein.	LONG
ret_val	Erfolgsstatus des Befehls: -1: Fehler, der Schreib-Ringspeicher des TiCo ist nicht korrekt deklariert. ≥0: Befehl war erfolgreich. Der Rückgabewert ist die Anzahl der übertragenen Werte.	LONG

Bemerkungen

Der Befehl liest nicht mehr als **maxcount** Daten. Wenn weniger Daten im Ringspeicher enthalten sind, werden alle vorhandenen Daten übertragen.

Beim Lesen von einem Ringspeicher speichert **P2_Get_TiCo_RingBuffer** die letzte Leseposition, den Lesezeiger, in dem Feld **tdrv_datatable[]**. Wenn der Befehl **RingBuffer_Empty** des TiCo-Prozessors korrekt arbeiten soll, muss in **tico_par** die Nummer einer globalen Variablen angegeben werden. Dadurch wird der Wert des Lesezeigers in die globale Variable des TiCo-Prozessors übertragen.

Siehe auch

[P2_Set_TiCo_RingBuffer](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_Get_TiCo_Status

P2_Get_TiCo_Status gibt zurück, ob der *TiCo*-Prozessor auf dem angegebenen Modul aktiv ist.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Get_TiCo_Status(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Status des <i>TiCo</i> -Prozessors. -3: Fehler, kein <i>TiCo</i> -Prozessor vorhanden. -2: Fehler, kein <i>Pro II</i> -Modul. 0: Prozessor ist gestoppt. 1: Prozessor läuft.	LONG

Bemerkungen

- / -

Siehe auch

[P2_Process_Status](#), [P2_Workload](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_GetData_Long

P2_GetData_Long liest Werte aus einem globalen Feld von einem TiCo-Prozessor und schreibt die Werte in ein Feld.

Syntax

```
#Include ADwinPro_All.inc

P2_GetData_Long(tdrv_datatable[], src_array_no,
                src_array_idx, count, dest_array[],
                dest_array_idx, flowrate)
```

Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code> []	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>src_</code> <code>array_no</code>	Nummer (1...16) des globalen Felds <code>Data_x</code> auf dem TiCo-Prozessor.	LONG
<code>src_</code> <code>array_idx</code>	Index (1...n) des ersten Elements, das im globalen Feld <code>src_array_no</code> gelesen wird.	LONG
<code>count</code>	Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>dest_</code> <code>array[]</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<code>dest_</code> <code>array_idx</code>	Ziel-Startindex (1...n): Feldelement, ab dem die Werte im Zielfeld abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Es muss gewährleistet sein, dass

- das globale Feld `src_array_no` auf dem *TiCo*-Prozessor deklariert ist und
- das Feld `dest_array` mit mindestens `count` Elementen deklariert ist.

Siehe auch

[P2_Get_Par](#), [P2_Set_Par](#), [P2_SetData_Long](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset_41[150] As Long   'settings array for data
                             'transfer

Dim Data_4[200] As Long
```

Init:

```
REM initialize data transfer ADwin CPU <-> TiCo
P2_TDrv_Init(module,tico_no,tset_41)
```

Event:

```
REM read 120 values from TiCo Data_2 (starting from Data_
REM 2[20])
REM into Data_4 of ADwin CPU (starting from index 5).
REM flowrate is high
P2_GetData_Long(tset_41,2,20,120,Data_4,5,3)
```

P2_Process_Status

P2_Process_Status gibt den Status eines Prozesses auf einem TiCo-Prozessor des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Process_Status(module, tico_no,
                             process_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1) des TiCo-Prozessors auf dem Modul (nicht Typ).	LONG
process_no	Nummer (1...4) des TiCo-Prozesses.	LONG
ret_val	Status des Prozesses: ≠1: Prozess läuft. 0: Prozess läuft nicht, d.h. er ist nicht geladen, nicht gestartet oder gestoppt.	LONG

Bemerkungen

- / -

Siehe auch

[P2_TiCo_Get_Processdelay](#), [P2_TiCo_Set_Processdelay](#), [P2_TiCo_Start_Process](#), [P2_TiCo_Stop_Process](#), [P2_Workload](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim Data_4[200] As Long
Dim tset[150] As Long      'settings array for data
                           'transfer
```

Init:

```
    P2_TDrv_Init(module,tico_no,tset)
```

Event:

```
    REM read status of TiCo process 1
    Par_1 = P2_Process_Status(module,tico_no,1)
    REM if process is running, read TiCo Par_5
    If (Par_1 = 1) Then
        Par_2 = P2_Get_Par(module,tico_no,5)
    EndIf
```

P2_RingBuffer_Empty

P2_RingBuffer_Empty gibt die Anzahl der freien Elemente in einem Schreib-Ringspeicher eines TiCo-Prozessors zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Ringbuffer_Empty(tdrv_datatable[],
                              data_no)
```

Parameter

tdrv_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
data_no	Nummer (1...16) des Schreib-Ringspeichers Data_n auf dem TiCo-Prozessor.	LONG
ret_val	Anzahl der freien Elemente im Schreib-Ring-speicher.	LONG

Bemerkungen

Bei **P2_Get_TiCo_RingBuffer** ist keine vorherige Abfrage mit **P2_RingBuffer_Empty** erforderlich.

Siehe auch

[P2_Get_TiCo_RingBuffer](#), [P2_RingBuffer_Full](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_RingBuffer_Full

P2_RingBuffer_Full gibt die Anzahl der genutzten Elemente in einem Lese-Ringspeicher eines TiCo-Prozessors zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Ringbuffer_Full (tdrv_datatable[],
                             data_no)
```

Parameter

tdrv_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
data_no	Nummer (1...16) des Lese-Ringspeichers Data_n auf dem TiCo-Prozessor.	LONG
ret_val	Anzahl (0...n) der genutzten Elemente im Lese-Ringspeicher.	LONG

Bemerkungen

Bei **P2_Set_TiCo_RingBuffer** ist keine vorherige Abfrage mit **P2_RingBuffer_Full** erforderlich.

Siehe auch

[P2_Set_TiCo_RingBuffer](#), [P2_RingBuffer_Empty](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_Set_Par

P2_Set_Par setzt den Wert einer globalen Variablen **Par_x** auf einem TiCo-Prozessor des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.inc
P2_Set_Par(module, tico_no, par_no, value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1) des TiCo-Prozessors auf dem Modul (nicht Typ).	LONG
par_no	Nummer (1...80) der globalen Variablen.	LONG
value	Wert ($-2^{31} \dots +2^{31}-1$) der globalen Variablen.	LONG

Bemerkungen

P2_Set_Par setzt den Wert der globalen Variablen unabhängig davon, ob gerade ein TiCo-Prozess läuft. Da *ADwin* CPU- und TiCo-Prozesse nicht synchron arbeiten, können sich die Werte globaler Variablen für den TiCo-Prozess völlig unerwartet während der Laufzeit ändern.

Falls erforderlich, empfehlen wir eine Synchronisierung von *ADwin* CPU und TiCo mit Hilfe eines Software-Handshakes.

Siehe auch

[P2_Get_Par](#), [P2_Get_Par_Block](#), [P2_GetData_Long](#), [P2_Set_Par_Block](#), [P2_SetData_Long](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  P2_TDrv_Init(module,tico_no,tset)
  Par_5=200

Event:
  REM write value of Par_5 (ADwin CPU) to TiCo Par_2
  P2_Set_Par(module,tico_no,2,Par_5)
```

P2_Set_Par_Block

P2_Set_Par_Block schreibt Werte aus einem Feld in eine Anzahl von globalen Variablen **Par_x** des TiCo-Prozessors vom angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

P2_Set_Par_Block(module, tico_no, src_array[],
                 src_array_idx, par_no, par_count)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1) des TiCo-Prozessors auf dem Modul (nicht Typ).	LONG
src_array[]	Quellfeld, aus dem Daten übertragen werden.	ARRAY
src_array_idx	Indes des Feldelements, ab dem Werte aus dem Quellfeld gelesen werden.	LONG
par_no	Index (1...80) der ersten globalen TiCo-Variablen, die beschrieben wird.	LONG
par_count	Anzahl der zu übertragenden Werte.	LONG

Bemerkungen

P2_Set_Par_Block setzt die Wert der globalen Variablen unabhängig davon, ob gerade ein TiCo-Prozess läuft. Da *ADwin* CPU- und TiCo-Prozesse nicht synchron arbeiten, können sich die Werte globaler Variablen für den TiCo-Prozess völlig unerwartet während der Laufzeit ändern.

Falls erforderlich, empfehlen wir eine Synchronisierung von *ADwin* CPU und *TiCo* mit Hilfe eines Software-Handshakes.

Siehe auch

[P2_Get_Par](#), [P2_Get_Par_Block](#), [P2_GetData_Long](#), [P2_Set_ParP2_SetData_Long](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                             'transfer

Dim Data_1[40] As Long
Dim i As Long
```

Init:

```
P2_TDrv_Init(module,tico_no,tset)
For i = 1 To 40
    Data_1[i] = i*10
Next
Par_15=1111
Par_16=2222
Par_17=3333
Par_18=4444
Par_19=5555
```

Event:

```
REM write 40 values from Data_1 (starting from Data_1[1])
REM into Par_1...Par_40 (TiCo)
P2_Set_Par_Block(module,tico_no,Data_1,1,1,40)
REM write Par_15...Par_19 (ADwin CPU) into Par_50...Par_54
REM (TiCo)
P2_Set_Par_Block(module,tico_no,PAR,15,50,5)
```

P2_Set_TiCo_RingBuffer

P2_Set_TiCo_RingBuffer schreibt Werte aus einem Feld der ADwin CPU in einen Ringspeicher auf einem TiCo-Prozessor.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Set_TiCo_RingBuffer(tdrv_datatable[],
    dest_array_no, src_array[], src_array_idx,
    maxcount, flowrate, tico_par, struct)
```

Parameter

tdrv_ datatable []	Feld, das Einstellungen für die Datenübertragung enthält, u.a. Moduladresse und TiCo-Nummer.	ARRAY LONG
dest_ array_no	Nummer (1...16) des Lese-Ringspeichers Data_ n auf dem TiCo-Prozessor.	FLOAT
src_ array []	Quellfeld, aus dem die Werte übertragen werden.	ARRAY LONG
src_ array_idx	Index (1...n) des ersten Elements im Feld src_ array [], das gelesen wird.	LONG
maxcount	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
flowrate	Auswertung nur für niederpriorie Prozesse: Kenn- wert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
tico_par	Kennzahl zum Übertragen des Schreibzeigers zum TiCo-Prozessor. 1...80: Nummer (1...80) der globalen Variablen Par_n des TiCo-Prozessors, in die der aktuelle Schreibzei- ger geschrieben wird. 0: Der Schreibzeiger wird nicht übertragen.	LONG

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>[]</code>		
<code>dest_</code>	Nummer (1...16) des Lese-Ringspeichers <code>Data_</code>	FLOAT
<code>array_no</code>	<code>n</code> auf dem TiCo-Prozessor.	
<code>src_</code>	Quellfeld, aus dem die Werte übertragen werden.	ARRAY
<code>array[]</code>		LONG
<code>src_</code>	Index (1...n) des ersten Elements im Feld <code>src_</code>	LONG
<code>array_idx</code>	<code>array[]</code> , das gelesen wird.	
<code>maxcount</code>	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>tico_par</code>	Kennzahl zum Übertragen des Schreibzeigers zum TiCo-Prozessor. 1...80: Nummer (1...80) der globalen Variablen <code>Par_n</code> des TiCo-Prozessors, in die der aktuelle Schreibzeiger geschrieben wird. 0: Der Schreibzeiger wird nicht übertragen.	LONG
<code>struct</code>	Kennwert für Messwert-Sätze: 0: Beliebige Anzahl von Daten schreiben. >0: Die Anzahl der übertragenen Daten muss ein Vielfaches von <code>struct</code> sein.	LONG
<code>ret_val</code>	Erfolgsstatus des Befehls: -1: Fehler, der Lese-Ringspeicher des TiCo ist nicht korrekt deklariert. ≥0: Befehl war erfolgreich. Der Rückgabewert ist die Anzahl der übertragenen Werte.	LONG

Bemerkungen

Der Befehl schreibt nicht mehr als `maxcount` Daten. Wenn weniger

freie Elemente im Ringspeicher vorhanden sind, werden nur die freien Elemente im Ringspeicher beschrieben.

Beim Schreiben in einen Ringspeicher speichert **P2_Set_TiCo_RingBuffer** die letzte Schreibposition, den Schreibzeiger, in dem Feld `tdrv_datatable[]`. Wenn der Befehl **RingBuffer_Full** des *TiCo*-Prozessors korrekt arbeiten soll, muss in `tico_par` die Nummer einer globalen Variablen angegeben werden. Dadurch wird der Wert des Lesezeigers in die globale Variable des *TiCo*-Prozessors übertragen.

Siehe auch

[P2_Get_TiCo_RingBuffer](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_SetData_Long

P2_SetData_Long liest Werte aus einem Feld und schreibt sie in ein globales Feld eines TiCo-Prozessors auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

P2_SetData_Long(tdrv_datatable[], dest_array_no,
    dest_array_idx, count, src_array[],
    src_array_idx, flowrate)
```

Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>dest_</code>		
<code>array_no</code>	Nummer (1...16) des globalen Felds <code>Data_x</code> auf dem TiCo-Prozessor.	LONG
<code>dest_</code>		
<code>array_idx</code>	Index (1...n) des ersten Elements, das im globalen Feld <code>dest_array_no</code> beschrieben wird.	LONG
<code>count</code>	Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>src_</code>		
<code>array[]</code>	Quellfeld, aus dem die Werte gelesen werden.	ARRAY
<code>src_</code>		
<code>array_idx</code>	Quell-Startindex (1...n): Feldelement, ab dem die Werte gelesen werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Es muss gewährleistet sein, dass

- das globale Feld `dest_array_no` auf dem TiCo-Prozessor deklariert ist und
- das Feld mit mindestens `count` Elementen deklariert ist.

Siehe auch

[P2_Get_Par](#), [P2_GetData_Long](#), [P2_Set_Par](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                             'transfer

Dim Data_1[150] As Long
Dim tset_41[150] As Long
Dim i As Long
```

Init:

```
REM initialize data transfer ADwin CPU <-> TiCo
P2_TDrv_Init(module,tico_no,tset_41)
For i = 1 To 80
    Data_1[i] = i
Next
```

Event:

```
REM write 120 values
REM from Data_1 of ADwin CPU, starting from Data_1[5]
REM into TiCo Data_2, starting from Data_2[1]
REM flowrate is high
P2_SetData_Long(tset_41,2,1,120,Data_1,5,3)
```

P2_TDrv_Init

P2_TDrv_Init initialisiert die Datenübertragung zwischen *ADwin* CPU und einem bestimmten *TiCo*-Prozessor.

Syntax

```
#Include ADwinPro_All.inc

REM define settings array for TiCo y on module x
Dim tdrv_datatable[150] As Long

P2_TDrv_Init(module, tico_no, tdrv_datatable[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1) des <i>TiCo</i> -Prozessors auf dem Modul (nicht Typ).	LONG
tdrv_datatable	Feld, das Einstellungen für die Datenübertragung aufnimmt.	ARRAY
	[]	LONG

Bemerkungen

Der Befehl muss vor der Datenübertragung zwischen *ADwin* CPU und *TiCo* ausgeführt werden. Der Befehl sollte im Abschnitt **Init:** stehen.

Fast alle Befehle, die auf den *TiCo* zugreifen, benötigen eine Initialisierung der Datenübertragung.

Die Initialisierung muss für jeden *TiCo*-Prozessor separat durchgeführt werden. Dabei muss auch jeweils ein Feld **tdrv_datatable[]** mit 150 Elementen angelegt werden.

Das Feld **tdrv_datatable[]** wird von den Befehlen **P2_GetData_Long**, **P2_SetData_Long** und **P2_Workload** verwendet.

Siehe auch

[P2_Get_Par](#), [P2_Get_Par_Block](#), [P2_TiCo_Get_Processdelay](#), [P2_GetData_Long](#), [P2_Set_Par](#), [P2_Set_Par_Block](#), [P2_TiCo_Set_Processdelay](#), [P2_SetData_Long](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define module_a 4          'address of module a
#Define TiCo_a 1            'TiCo no.
#Define module_b 7          'address of module b
#Define TiCo_b 1            'TiCo no.
Dim Data_4[200] As Long
Dim Data_5[1000] As Long
Dim tset_41[150] As Long
Dim tset_71[150] As Long
```

Init:

```
REM initialize data transfer ADwin CPU <-> TiCo
P2_TDrv_Init(module_a,TiCo_a,tset_41)
P2_TDrv_Init(module_b,TiCo_b,tset_71)
```

Event:

```
REM read 120 values from module a, TiCo Data_2 (starting
REM from
REM Data_2[20]) into Data_4 of ADwin CPU (starting from
REM index 5)
REM flowrate is high
P2_GetData_Long(tset_41,2,20,120,Data_4,5,3)
REM read 800 values from module b, TiCo Data_7 (starting
REM from
REM Data_7[9]) into Data_5 of ADwin CPU (starting from
REM index 1).
REM flowrate is high
P2_GetData_Long(tset_71,7,9,800,Data_5,1,3)
```

P2_TiCo_Get_Processdelay

P2_TiCo_Get_Processdelay gibt das Processdelay (die Zykluszeit) eines Prozesses auf einem *TiCo*-Prozessor des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_TiCo_Get_Processdelay (
    tdrv_datatable[], process_no)
```

Parameter

tdrv_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
process_	Nummer (1...4) des TiCo-Prozesses.	LONG
no		
ret_val	Aktuell eingestellte Zykluszeit ($-2^{31} \dots +2^{31}-1$) des TiCo-Prozessors in Taktzyklen. Ein Taktzyklus dauert 20ns.	LONG

Bemerkungen

Bei einem zeitgesteuerten Prozess wird der Abschnitt **Event**: vom internen Zähler zyklisch und in festen Zeitabständen aufgerufen. Der Zeitabstand zwischen zwei Aufrufen, Zykluszeit oder Processdelay genannt, wird in Zähler-Taktzyklen gezählt.

Siehe auch

[P2_Process_Status](#), [P2_TiCo_Set_Processdelay](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12

Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4
Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer
```

Init:

```
P2_TDrv_Init(module,tico_no,tset)
```

Event:

```
REM read processdelay of process 1 into Par_1
Par_1 = P2_TiCo_Get_Processdelay(tset,1)
```

P2_TiCo_Get_Process_Error

P2_TiCo_Get_Process_Error gibt den (in der Regel) zuletzt aufgetretenen Fehler des Prozesses auf einem *TiCo*-Prozessor zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_TiCo_Get_Process_Error (
    tdrv_datatable[], process_no)
```

Parameter

tdrv_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
process_	Nummer (1...4) des TiCo-Prozesses.	LONG
no		
ret_val	Nummer des zuletzt aufgetretenen Fehlers im Prozess:	LONG
	0: kein Fehler	
	10: Zugriff auf eine zu große Elementnummer eines globalen Felds.	
	11: Zugriff auf eine zu kleine Elementnummer (≤ 0) eines globalen Felds.	
	12: Zugriff auf eine zu große Elementnummer eines lokalen Felds.	
	13: Zugriff auf eine zu kleine Elementnummer (≤ 0) eines lokalen Felds.	

Bemerkungen

Der Rückgabewert der Systemvariablen ist nur definiert, wenn der Debug-Modus eingeschaltet ist (siehe [Option Debug mode](#), Seite 72). Die Variable kann nur gelesen werden.

Siehe auch

[P2_Process_Status](#), [P2_TiCo_Set_Processdelay](#), [P2_TDrv_Init](#)

Gültig für

- / -

Beispiel

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer
```

Init:

```
P2_TDrv_Init(module,tico_no,tset)
```

Event:

```
REM read recent process error of process 1 into Par_1
Par_1 = P2_TiCo_Get_Process_Error(tset,1)
```

P2_TiCo_Flash

P2_TiCo_Flash überträgt eine *TiCoBasic*-Binärdatei aus einem Feld in den Flash-Speicher eines *TiCo*-Prozessors.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_TiCo_Flash(module,tico_no,array[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1) des TiCo-Prozessors auf dem Modul (nicht Typ).	LONG
array[]	Feld, in dem die zu übertragenden Daten enthalten sind.	ARRAY
ret_val	Status der Datenübertragung: 0: Datenübertragung erfolgreich. -2: kein Modul an dieser Adresse oder das Modul hat keinen <i>TiCo</i> -Prozessor. -1: Fehler: Befehl darf nur bei niedriger Priorität verwendet werden. 1: Fehler: Ungültige Prozessornummer. 2: Fehler: Programmspeicher ist zu klein. 3: Fehler: Datenspeicher ist zu klein. 4: Fehler: Programm- und Datenspeicher sind zu klein. 5: Fehler: Falsches Passwort. 6: Fehler: Externer Speicher ist zu klein. 7: Fehler: Flash-Speicher (EEPROM) ist zu klein. 12: Fehler: Die Binärdatei ist nicht für den <i>TiCo</i> -Prozessor geeignet.	LONG

Bemerkungen

Verwenden Sie **P2_TiCo_Flash** nur in niederprioren Prozessen.

Der Befehl **P2_TiCo_Flash** dient dazu, eine *TiCoBasic*-Binärdatei – ein kompiliertes *TiCo*-Programm – in den Flash-Speicher eines *TiCo*-Prozessors zu übertragen. Dazu sind folgende Schritte erforderlich:

- Erzeugen Sie in der Oberfläche *TiCoBasic* eine Binärdatei mit Build ► Make Bin File.
- Übertragen Sie die Binärdatei in ein globales Feld der *ADwin* CPU. Geeignet ist die Funktion `File2Data` (bzw. `LoadFromFile`), die in mehreren Programmiersprachen zur Verfügung steht.
- Übertragen Sie die Daten im globalen Feld `array[]` mit dem Befehl **P2_TiCo_Flash** in den Flash-Speicher.
- Wenn der Bootloader aktiviert ist, startet der Prozess der *TiCoBasic*-Binärdatei automatisch nach dem nächsten Einschalten der *ADwin*-Hardware.

Wenn das Feld `array[]` keine *TiCoBasic*-Binärdatei enthält, ist der Rückgabewert des Befehls ungültig.

Siehe auch

[P2_TiCo_Load](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_TiCo_Load

P2_TiCo_Load überträgt eine *TiCoBasic*-Binärdatei aus einem Feld in den Speicher eines *TiCo*-Prozessors und startet den Prozess.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_TiCo_Load(module,tico_no,array[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1) des <i>TiCo</i> -Prozessors auf dem Modul (nicht Typ).	LONG
array[]	Feld, in dem die <i>TiCoBasic</i> -Binärdatei enthalten sind.	ARRAY
ret_val	Status der Datenübertragung: 0: Datenübertragung erfolgreich. -2: kein Modul an dieser Adresse oder das Modul hat keinen <i>TiCo</i> -Prozessor. -1: Fehler: Befehl darf nur bei niedriger Priorität verwendet werden. 1: Fehler: Ungültige Prozessornummer. 2: Fehler: Programmspeicher ist zu klein. 3: Fehler: Datenspeicher ist zu klein. 4: Fehler: Programm- und Datenspeicher sind zu klein. 5: Fehler: Falsches Passwort. 6: Fehler: Externer Speicher ist zu klein. 10: Fehler: Der benötigte externe SRAM-Speicher ist zu klein. 11: Fehler: Das Feld enthält keine gültige <i>TiCoBasic</i> -Binärdatei. 12: Fehler: Die Binärdatei ist nicht für den <i>TiCo</i> -Prozessor geeignet.	LONG

Bemerkungen

Verwenden Sie **P2_TiCo_Load** nur in niederprioren Prozessen.

Der Befehl **P2_TiCo_Load** dient dazu, eine *TiCoBasic*-Binärdatei – ein kompiliertes *TiCo*-Programm – in den *TiCo*-Prozessor zu übertragen. Dazu sind folgende Schritte erforderlich:

- Erzeugen Sie in der Oberfläche *TiCoBasic* eine Binärdatei mit Build ► Make Bin File.
- Übertragen Sie die Binärdatei in ein globales Feld des *ADwin*-Prozessors. Geeignet ist die Funktion `File2Data` (bzw. `LoadFromFile`), die in mehreren Programmiersprachen zur Verfügung steht.
- Übertragen Sie die Daten im globalen Feld `array[]` mit dem Befehl **P2_TiCo_Load** auf den *TiCo*-Prozessor.
- Der Prozess der *TiCoBasic*-Binärdatei wird automatisch gestartet.

Siehe auch

[P2_TiCo_Flash](#)

Gültig für

AIIn-32/18-D-TiCo Rev. E, AIIn-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

siehe auch C:\ADwin\ADbasic\samples_ADwin_Proll

P2_TiCo_Restart

P2_TiCo_Restart stoppt die TiCo-Prozessoren auf den angegebenen Modulen und startet sie anschließend wieder.

Syntax

```
#Include ADwinPro_All.inc
P2_TiCo_Restart(module_pattern)
```

Parameter

module_ Bitmuster zum Ansprechen der Module, deren **LONG** |
pattern TiCos gestoppt werden sollen:
 Bit = 0: Modul ignorieren.
 Bit = 1: TiCos auf dem Modul stoppen.

Bits in module_pattern	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen

Das Stoppen beendet auf den angesprochenen TiCo-Prozessoren sofort jeden Prozess sowie den Zähler. Die Daten werden durch das Stoppen nicht verändert.

Die TiCo-Prozessoren auf den gewählten Modulen werden gleichzeitig gestartet. Der Befehl eignet sich daher, um alle angesprochenen TiCo-Prozessoren zu synchronisieren.

Siehe auch

[P2_TiCo_Stop](#), [P2_TiCo_Start](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_TiCo_Set_Processdelay

P2_TiCo_Set_Processdelay setzt das Processdelay (die Zykluszeit) eines Prozesses auf einem TiCo-Prozessor des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.inc

P2_TiCo_Set_Processdelay(tdrv_datatable[],
    process_no, value)
```

Parameter

tdrv_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
process_no	Nummer (1...4) des TiCo-Prozesses.	LONG
value	Einzustellender Wert für das Processdelay.	LONG

Bemerkungen

- / -

Siehe auch

[P2_TiCo_Get_Processdelay](#), [P2_Process_Status](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  P2_TDrv_Init(module,tico_no,tset)
  Processdelay = 6000 'set cycle time

Event:
  REM set TiCo processdelay to run with same cycle time as
  REM the process on the ADwin processor (T11)
  P2_TiCo_Set_Processdelay(tset,1,Processdelay/6)
```

P2_TiCo_Start

P2_TiCo_Start startet die TiCo-Prozessoren auf den angegebenen Modulen.

Syntax

```
#Include ADwinPro_All.inc
P2_TiCo_Start(module_pattern)
```

Parameter

module_ Bitmuster zum Ansprechen der Module, deren **LONG** |
pattern TiCos gestartet werden sollen:
 Bit = 0: Modul ignorieren.
 Bit = 1: TiCos auf dem Modul starten.

Bits in module_pattern	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen

Die TiCo-Prozessoren auf den gewählten Modulen werden gleichzeitig gestartet. Der Befehl eignet sich daher, um alle angesprochenen TiCo-Prozessoren zu synchronisieren.

Siehe auch

[P2_TiCo_Stop](#), [P2_TiCo_Restart](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_TiCo_Start_Process startet einen Prozess auf einem TiCo-Prozessor.

Syntax

```
#Include ADwinPro_All.inc

P2_Tico_Start_Process(tdrv_datatable[], process_no)
```

Parameter

tdrv_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
process_	Nummer (1...4) des TiCo-Prozesses.	LONG
no		

Bemerkungen

Der Prozess muss bereits auf dem TiCo-Prozessor vorhanden sein.

Siehe auch

[P2_TiCo_Get_Processdelay](#), [P2_Process_Status](#), [P2_TiCo_Set_Processdelay](#), [P2_TiCo_Start_Process](#), [P2_TiCo_Stop_Process](#), [P2_Workload](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  Par_1 = 0
  Processdelay = 100000
  P2_TDrv_Init(module,tico_no,tset)
  REM start TiCo process in parallel to ADwin CPU process
  P2_Tico_Start_Process(tset,1)

Event:
  REM ... program

Finish:
  REM stop TiCo process in parallel to ADwin CPU process
  P2_Tico_Stop_Process(tset,1)
```

P2_TiCo_Stop

P2_TiCo_Stop stoppt die TiCo-Prozessoren auf den angegebenen Modulen.

Syntax

```
#Include ADwinPro_All.inc
P2_TiCo_Stop(module_pattern)
```

Parameter

module_ Bitmuster zum Ansprechen der Module, deren **LONG** |
pattern TiCos gestoppt werden sollen:
 Bit = 0: Modul ignorieren.
 Bit = 1: TiCos auf dem Modul stoppen.

Bits in module_pattern	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen

Das Stoppen beendet auf den angesprochenen TiCo-Prozessoren sofort jeden Prozess sowie den Zähler. Die Daten werden durch das Stoppen nicht verändert.

Siehe auch

[P2_TiCo_Start](#), [P2_TiCo_Restart](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

- / -

P2_TiCo_Stop_Process

P2_TiCo_Stop_Process stoppt einen Prozess auf einem TiCo-Prozessor.

Syntax

```
#Include ADwinPro_All.inc

P2_Tico_Stop_Process(tdrv_datatable[], process_no)
```

Parameter

tdrv_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
process_	Nummer (1...4) des TiCo-Prozesses.	LONG
no		

Bemerkungen

Der Prozess muss auf dem TiCo-Prozessor vorhanden sein.

Siehe auch

[P2_TiCo_Get_Processdelay](#), [P2_Process_Status](#), [P2_TiCo_Set_Processdelay](#), [P2_TiCo_Start_Process](#), [P2_Workload](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  Par_1 = 0
  Processdelay = 100000
  P2_TDrv_Init(module,tico_no,tset)
  REM start TiCo process in parallel to ADwin CPU process
  P2_Tico_Start_Process(tset,1)

Event:
  REM ... program

Finish:
  REM stop TiCo process in parallel to ADwin CPU process
  P2_Tico_Stop_Process(tset,1)
```

P2_Workload

P2_Workload gibt die Auslastung eines TiCo-Prozessors auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_Workload(tdrv_datatable[])
```

Parameter

tdrv_ datatable []	Feld, das Einstellungen für die Datenübertragung enthält, u.a. Moduladresse und TiCo-Nummer.	ARRAY LONG
ret_val	Prozessor-Auslastung in Prozent (0.0 ... 100.0) oder Fehlerwert: <0: TiCo-Prozessor ist gestoppt oder das Modul besitzt keinen TiCo-Prozessor oder kein Modul an der übergebenen Adresse.	FLOAT

Bemerkungen

Der Rückgabewert ist die mittlere Prozessorauslastung in der Zeit zwischen dem vorherigen und dem aktuellen Aufruf von **P2_Workload**. Beim ersten Aufruf in einem Programm ist der Rückgabewert daher ungültig.

Der kürzeste Zeitabstand zwischen zwei Befehlsaufrufen sollte mindestens das 100fache des **Processdelay** betragen. Anderenfalls kann der Rückgabewert mit einem Fehler über 1% behaftet sein.

Wenn der vorherige Aufruf von **P2_Workload** länger als 85 Sekunden zurück liegt, ist der Rückgabewert ungültig. Rufen Sie den Befehl einfach ein zweites Mal auf, um einen gültigen Rückgabewert zu erhalten.

Siehe auch

[P2_TiCo_Get_Processdelay](#), [P2_Process_Status](#), [P2_TiCo_Set_Processdelay](#), [P2_TiCo_Stop_Process](#), [P2_TDrv_Init](#)

Gültig für

Aln-32/18-D-TiCo Rev. E, Aln-8/18-TiCo Rev. E, AOut-1/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer
```

Init:

```
P2_TDrv_Init(module,tico_no,tset)
```

Event:

```
REM read TiCo workload
FPar_1 = P2_Workload(tset)
```


8 Was tun bei Problemen?

Wenn Sie bei der Installation Probleme haben, ziehen Sie bitte die Dokumentation zu Ihrem ADwin-System zu Rate. Überprüfen Sie, ob alle Einstellungen richtig und vollständig durchgeführt wurden. Prüfen Sie auch, ob die Einstellungen im Menü „Options\Compiler“ richtig sind.


Sollten Ihre Probleme dann immer noch bestehen, rufen Sie uns bitte an. Auch wenn Sie weitergehende Hilfe wünschen, stehen wir Ihnen gern zur Verfügung; Sie finden Adresse und Telefonnummer Ihres Ansprechpartners in der vorderen Umschlagseite des Handbuchs.

Anhang

A.1 Tastaturkürzel in TiCoBasic

Um die Tastaturkürzel für Textbausteine zu sehen, öffnen Sie die Datei <TiCoBasicCS.xml> in C:\ADwin\TiCoBasic\Common\ mit einem Browser.

Tastenkürzel	Funktion	Menüaufruf
F1	Hilfethema für den markierten Befehl aufrufen.	
CTRL-F1	Inhaltsverzeichnis der Hilfe aufrufen.	Help ► Content
F2	Deklaration des markierten Befehls anzeigen.	
CTRL-F2	Zur Deklaration des markierten Befehls springen.	
F3	Text nochmals vorwärts suchen.	Edit ► Find Next
SHIFT-F3	Text nochmals rückwärts suchen.	
CTRL-F3	Text an Cursor-Position vorwärts suchen.	
CTRL-SHIFT-F3	Text an Cursor-Position rückwärts suchen.	
CTRL-F5	ADwin-CPU für Kommunikation mit dem TiCo-Prozessor initialisieren.	
CTRL-SHIFT-F5	TiCo-Prozessor sofort stoppen und zurücksetzen	
F6	Bibliothek erzeugen.	Build ► Make Lib File
F7	Binärdatei erzeugen.	Build ► Make Bin File
F8	Quelltext kompilieren.	Build ► Compile
CTRL-F8	Prozess starten.	
F9	Prozess stoppen.	
CTRL-SPACE	Deklaration einfügen oder vervollständigen.	
CTRL-SHIFT-SPACE	Parameter einer Sub / Function anzeigen.	

Tastenkürzel	Funktion	Menüaufruf
CTRL-A	Alles markieren.	Edit ► Select All
CTRL-B	Markierte Zeilen in Kommentar umwandeln.	Source context menu: Comment Block
CTRL-SHIFT-B	Kommentarzeichen aus markierten Zeilen löschen.	Source context menu: Uncomment Block
CTRL-C	Kopieren.	Edit ► Copy
CTRL-F	Text suchen.	Edit ► Find
CTRL-G	Zu einer Zeile springen.	
CTRL-H	Text ersetzen.	Edit ► Replace
CTRL-I	Zeilen nach rechts einrücken	Source context menu: Indent
CTRL-SHIFT-I	Zeilen nach links rücken	Source context menu: Outdent
CTRL-N	Neue Quelltextdatei.	File ► New
CTRL-O	Quelltextdatei öffnen.	File ► Open
CTRL-P	Quelltextdatei drucken.	File ► Print
CTRL-R	Verwendete Parameter markieren.	Parameter window: Icon 
CTRL-S	Quelltextdatei speichern.	File ► Save
CTRL-U	Kleinschreibung.	
CTRL-SHIFT-U	Großschreibung.	
CTRL-V	Einfügen.	Edit ► Paste
CTRL-X	Ausschneiden.	Edit ► Cut
CTRL-Z	Änderung zurücknehmen.	Edit ► Undo
CTRL-SHIFT-Z	Änderung wieder herstellen.	Edit ► Redo
CTRL-K + K	Textmarke ein- oder ausschalten.	
CTRL-K + N	Zur nächsten Textmarke springen.	
CTRL-K + P	Zur vorigen Textmarke springen.	
CTRL-K + X	Einen Textbaustein aus einer Liste einfügen.	

Legende:

A-B: Tasten A und B gleichzeitig drücken.

A+B: Tasten A und B nacheinander drücken, erst A, dann B.

A.2 ASCII-Zeichensatz

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
00h 0	01h 1	02h 2	03h 3	04h 4	05h 5	06h 6	07h 7
BS¹	TAB²	LF³	VT	FF	CR⁴	SO	SI
08h 8	09h 9	0Ah 10	0Bh 11	0Ch 12	0Dh 13	0Eh 14	0Fh 15
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
10h 16	11h 17	12h 18	13h 19	14h 20	15h 21	16h 22	17h 23
CAN	EM	SUB	ESC	FS	GS	RS	US
18h 24	19h 25	1Ah 26	1Bh 27	1Ch 28	1Dh 29	1Eh 30	1Fh 31
SPC⁵	!	"	#	\$	%	&	'
20h 32	21h 33	22h 34	23h 35	24h 36	25h 37	26h 38	27h 39
()	*	+	,	-	.	/
28h 40	29h 41	2Ah 42	2Bh 43	2Ch 44	2Dh 45	2Eh 46	2Fh 47
0	1	2	3	4	5	6	7
30h 48	31h 49	32h 50	33h 51	34h 52	35h 53	36h 54	37h 55
8	9	:	;	<	=	>	?
38h 56	39h 57	3Ah 58	3Bh 59	3Ch 60	3Dh 61	3Eh 62	3Fh 63
@	A	B	C	D	E	F	G
40h 64	41h 65	42h 66	43h 67	44h 68	45h 69	46h 70	47h 71
H	I	J	K	L	M	N	O
48h 72	49h 73	4Ah 74	4Bh 75	4Ch 76	4Dh 77	4Eh 78	4Fh 79
P	Q	R	S	T	U	V	W
50h 80	51h 81	52h 82	53h 83	54h 84	55h 85	56h 86	57h 87
X	Y	Z	[\]	^	_
58h 88	59h 89	5Ah 90	5Bh 91	5Ch 92	5Dh 93	5Eh 94	5Fh 95
`	a	b	c	d	e	f	g
60h 96	61h 97	62h 98	63h 99	64h 100	65h 101	66h 102	67h 103
h	i	j	k	l	m	n	o
68h 104	69h 105	6Ah 106	6Bh 107	6Ch 108	6Dh 109	6Eh 110	6Fh 111
p	q	r	s	t	u	v	w
70h 112	71h 113	72h 114	73h 115	74h 116	75h 117	76h 118	77h 119
x	y	z	{	 	}	~	␣
78h 120	79h 121	7Ah 122	7Bh 123	7Ch 124	7Dh 125	7Eh 126	7Fh 127

¹ Backspace, ² Tabulator, ³ Linefeed,
⁴ Carriage Return, ⁵ Space

A.3 Lizenzvertrag

Zwischen dem Käufer von *TiCoBasic* – nachfolgend „Lizenznehmer“ genannt –

und Jäger Computergesteuerte Messtechnik GmbH, Rheinstraße 2 - 4, 64653 Lorsch – nachfolgend „Jäger Messtechnik GmbH“ genannt – besteht der folgende Lizenzvertrag:

1. GEGENSTAND DES LIZENZVERTRAGES

1.1 Gegenstand des Lizenzvertrages ist die Software des Compilers und Entwicklungssystems *TiCoBasic* (im folgenden als „*TiCoBasic*-Software“ bezeichnet) und die gedruckte Benutzerdokumentation mit der Bezeichnung „*TiCoBasic*: Das Echtzeit-Entwicklungstool für *ADwin*-Systeme“ (im folgenden als „zugehöriges Schriftmaterial“ bezeichnet).

1.2 Die Jäger Messtechnik GmbH macht darauf aufmerksam, dass es nach dem Stand der Technik nicht möglich ist, Computersoftware so zu erstellen, dass sie in allen Anwendungen und Kombinationen fehlerfrei arbeitet. Gegenstand des Lizenzvertrages ist nur eine Computersoftware, die im Sinne der Benutzerdokumentation grundsätzlich brauchbar ist.

2. UMFANG DER BENUTZUNG

2.1 Die Jäger Messtechnik GmbH gewährt dem Lizenznehmer das einfache, nicht ausschließliche und persönliche Nutzungsrecht. Dies bedeutet, dass die beiliegende Kopie der *TiCoBasic*-Software nur auf einem einzelnen Computer und nur an einem Ort benutzt werden darf. Der Lizenznehmer darf die *TiCoBasic*-Software in körperlicher Form (d. h. auf einem Datenträger abgespeichert) von einem Computer auf einen anderen Computer übertragen, vorausgesetzt, dass sie zu jedem Zeitpunkt immer nur auf einem einzelnen Computer genutzt wird. Eine weitergehende Nutzung ist nicht zulässig.

2.2 Programme, die durch den Lizenznehmer mit der *TiCoBasic*-Software erstellt werden, dürfen uneingeschränkt verteilt und weiterverwendet werden.

3. BESONDERE BESCHRÄNKUNGEN

Dem Lizenznehmer ist es untersagt,

- a) ohne vorherige schriftliche Einwilligung der Jäger Messtechnik GmbH die *TiCoBasic*-Software oder das zugehörige schriftliche Material an

einen Dritten zu übergeben oder einem Dritten sonstwie zugänglich zu machen,

- b) die *TiCoBasic*-Software von einem Computer über ein Netz oder einen Datenübertragungskanal auf einen anderen Computer zu übertragen,
- c) ohne vorherige schriftliche Einwilligung der Jäger Messtechnik GmbH die *TiCoBasic*-Software abzuändern, zu übersetzen, zurückzuentwickeln, zu entkompilieren oder zu disassemblieren,

4. INHABERSCHAFT AN RECHTEN

4.1 Der Lizenznehmer erhält mit dem Erwerb des Produktes nur Eigentum an dem körperlichen Datenträger, auf dem die *TiCoBasic*-Software aufgezeichnet ist. Ein Erwerb von Rechten an der *TiCoBasic*-Software selbst ist damit nicht verbunden.

4.2 Die Jäger Messtechnik GmbH behält sich insbesondere alle Veröffentlichungs-, Vervielfältigungs-, Bearbeitungs- und Verwertungsrechte an der *TiCoBasic*-Software vor.

5. VERVIELFÄLTIGUNG

5.1 Die *TiCoBasic*-Software und das zugehörige Schriftmaterial sind urheberrechtlich geschützt.

Zu Sicherungszwecken ist dem Lizenznehmer das Anfertigen einer einzigen Reservekopie der *TiCoBasic*-Software erlaubt. Er ist verpflichtet, auf der Reservekopie den Urheberrechtsvermerk der Jäger Messtechnik GmbH anzubringen. Der in der *TiCoBasic*-Software vorhandene Urheberrechtsvermerk darf nicht entfernt werden.

5.2 Es wird ausdrücklich untersagt, die *TiCoBasic*-Software, wie auch das zugehörige Schriftmaterial, ganz oder teilweise in ursprünglicher oder abgeänderter Form oder in mit anderer Software zusammengemischer oder in anderer Software eingeschlossener Form zu kopieren oder anders zu vervielfältigen.

6. ÜBERTRAGUNG DES BENUTZUNGSRECHTES

6.1 Das Recht zur Benutzung der *TiCoBasic*-Software kann nur mit vorheriger schriftlicher Einwilligung der Jäger Messtechnik GmbH an einen Dritten übertragen werden. Der Lizenznehmer hat dann die bei ihm installierte Software vollständig zu löschen und dem Dritten die Software vollständig (Original-Datenträger mit Dokumentation, einschließlich der Sicherungskopie) zu übergeben. Das Nutzungsrecht darf ferner nur auf einen Dritten übertragen werden, wenn sich der Dritte zu

Gunsten der Jäger Messtechnik GmbH mit den Bestimmungen dieses Lizenzvertrages und den allgemeinen Geschäftsbedingungen der Jäger Messtechnik GmbH einverstanden erklärt.

6.2 Vermietung und Verleihung der *TiCoBasic*-Software sind ausdrücklich untersagt.

7. DAUER DES VERTRAGES

7.1 Der Lizenzvertrag läuft auf unbestimmte Zeit.

7.2 Das Recht des Lizenznehmers zur Benutzung der *TiCoBasic*-Software erlischt automatisch ohne Kündigung, wenn er eine Bedingung dieses Lizenzvertrages verletzt. Bei Beendigung des Nutzungsrechtes ist er verpflichtet, den Original-Datenträger und alle Kopien der *TiCoBasic*-Software einschließlich etwaiger abgeänderter Exemplare sowie das zugehörige Schriftmaterial zu vernichten.

8. SCHADENSERSATZ UND VERTRAGSSTRAFE BEI VERTRAGS- VERLETZUNG

8.1 Verletzt der Lizenznehmer Bestimmungen dieses Lizenzvertrages, so ist er zum Schadensersatz verpflichtet.

8.2 Unbeschadet dessen wird bei Verletzung des Urheberrechts der Jäger Messtechnik GmbH, unbefugter Benutzung der Software und unbefugter Weitergabe der Software an Dritte, eine Vertragsstrafe von 20.000,- EURO für jeden Fall der Zuwiderverhandlung vereinbart.

8.3 Unterlassungsansprüche werden von den Schadensersatzansprüchen und Vertragsstrafen nicht berührt.

9. ÄNDERUNGEN UND AKTUALISIERUNGEN

Die Jäger Messtechnik GmbH ist berechtigt, Aktualisierungen der *TiCoBasic*-Software nach eigenem Ermessen zu erstellen. Die Jäger Messtechnik GmbH ist nicht verpflichtet, Aktualisierungen der *TiCoBasic*-Software dem Lizenznehmer zur Verfügung zu stellen.

Für umfangreiche Aktualisierungen behält sich die Jäger Messtechnik GmbH vor, einen Kostenbeitrag zu erheben.

10. GEWÄHRLEISTUNG UND HAFTUNG DER JÄGER MESSTECHNIK GMBH

a) Die Jäger Messtechnik GmbH gewährleistet gegenüber dem Lizenznehmer, dass zum Zeitpunkt der Übergabe der Datenträger, auf dem die *TiCoBasic*-Software aufgezeichnet ist, unter normalen Betriebsbe-

dingungen und bei normaler Instandhaltung in seiner Materialausführung fehlerfrei ist.

- b) Sollte der Datenträger fehlerhaft sein, so kann der Lizenznehmer Ersatzlieferung während der Gewährleistungszeit von 6 Monaten ab Lieferung verlangen. Er muss dazu den Datenträger einschließlich einer Kopie der Rechnung/Quittung an die Jäger Messtechnik GmbH oder an den Vertriebspartner, von dem das Produkt bezogen wurde, zurückgeben.
- c) Wird ein Fehler im Sinne von Ziffer 10 b) nicht innerhalb angemessener Frist durch eine Ersatzlieferung behoben, so kann der Lizenznehmer nach seiner Wahl die Herabsetzung des Erwerbspreises oder das Rückgängigmachen des Lizenzvertrages verlangen. Weitere Ansprüche gegen die Jäger Messtechnik GmbH entstehen nicht.
- d) Aus den vorstehend unter Punkt 1.2 genannten Gründen übernimmt die Jäger Messtechnik GmbH keine Haftung für die Fehlerfreiheit der *TiCoBasic*-Software. Insbesondere übernimmt die Jäger Messtechnik GmbH keine Gewähr dafür, dass die *TiCoBasic*-Software den Anforderungen und Zwecken des Lizenznehmers genügt oder mit anderen von ihm ausgewählten Programmen zusammenarbeitet. Die Verantwortung für die richtige Auswahl und die Folgen der Benutzung der *TiCoBasic*-Software, sowie der damit beabsichtigten oder erzielten Ergebnisse trägt der Lizenznehmer. Das gleiche gilt für das die *TiCoBasic*-Software begleitende zugehörige Schriftmaterial.
- e) Jäger Messtechnik GmbH haftet nicht für Schäden, es sei denn, dass ein Schaden durch Vorsatz oder grobe Fahrlässigkeit seitens der Jäger Messtechnik GmbH verursacht worden ist. Eine Haftung wegen eventuell von der Jäger Messtechnik GmbH zugesicherten Eigenschaften bleibt unberührt. Eine Haftung für Mangelfolgeschäden, die nicht von der Zusicherung umfasst sind, ist ausgeschlossen.
- f) Die Jäger Messtechnik GmbH übernimmt keine Haftung für Schäden durch Viren, die mit dem Datenträger übertragen werden. Der Lizenznehmer ist angehalten, die Datenträger auf Viren zu überprüfen, bevor er die *TiCoBasic*-Software auf seinem Computer installiert.

11. SCHLUSSBESTIMMUNGEN

Die Unwirksamkeit einzelner Bestimmungen berührt die Wirksamkeit des Lizenzvertrages im übrigen nicht.

Ergänzend zu den Bestimmungen dieses Lizenzvertrages gelten die allgemeinen Geschäftsbedingungen der Jäger Messtechnik GmbH.

A.4 Kommandozeilen-Aufruf

Der *TiCoBasic*-Compiler kann nicht nur in der Bedienoberfläche aktiviert werden, sondern auch direkt aus Windows oder DOS aufgerufen werden (sogenannter „Kommandozeilen“-Aufruf). Der Compiler arbeitet in beiden Fällen auf gleiche Weise, kann also eine Quelltext-Datei kompilieren und daraus eine Binär- oder Library-Datei erzeugen.

Der Compiler-Aufruf wird nur ausgeführt, wenn Sie in *TiCoBasic* Ihren License key bereits eingegeben haben.



Beachten Sie die allgemeinen Hinweise zur [Kommandozeile unter Windows](#) auf Seite [A-9](#).

A.4.1 Syntax

Es gibt Kommandozeilen-Aufrufe zum Erzeugen einer Binärdatei (Hauptoption `/M`) und zum Erzeugen einer Bibliothek (Hauptoption `/L`).

Über Schalter werden die Optionen für den Compiler eingestellt. Wenn ein Schalter nicht angegeben ist, wird die jeweilige Voreinstellung (Default) verwendet; wir empfehlen aber dennoch, alle Schalter anzugeben, um Unklarheiten zu vermeiden¹.

Beim Erzeugen einer Binärdatei können gleichzeitig mehrere Quelldateien kompiliert werden. Daher wird unterschieden zwischen Optionen, die für alle Dateien gelten, und Optionen, die für jede Datei separat anzugeben sind (siehe Schalter `/PROCESS`).

Alternativ können Sie die Optionen für einen Aufruf in eine Datei, das `make-file`, schreiben und den Compiler mit der Hauptoption `/MAKE` aufrufen.

Schließlich gibt es die Hauptoptionen `/H` zum Aufruf eines kurzen Hilfetexts und `/VER` zur Anzeige der Compiler-Version.

Der Aufruf wird in einer einzelnen Zeile geschrieben. Achten Sie auf Großschreibung der Schalter.

1. Beispielsweise bleibt ein Aufruf mit allen Schaltern korrekt, auch wenn sich eine Default-Einstellung ändern sollte.

Syntax

```
TiCoBasicCompiler /M [/A"dest"] [/IP"path"]
[/LP"path"] [/Lx] [/Sx] [/P1]/PROCESS src.bas [/ET |
/EA | /EN | /EE /EEAx /EEMx /EEVx /EEOx] [/PNx] [/PH
| /PL] [/PDx] [/Ox] [/Vx]

TiCoBasicCompiler /L src.bas [/A"dest"] [/IP"path"]
[/LP"path"] [/Lx] [/Sx] [/P1] [/Ox]

TiCoBasicCompiler /MAKE"makefile"

TiCoBasicCompiler /H

TiCoBasicCompiler /VER
```

Achtung: Für den Prozessortyp T12 (Option /P12) muss anstelle von TiCoBasicCompiler der Aufruf ADbasic_C geschrieben werden.

Optionale Angaben stehen in eckigen Klammern []. Das Zeichen | trennt Schalter, die sich gegenseitig ausschließen.

Dateinamen können ohne, mit relativem oder mit absolutem Pfadnamen angegeben werden. Das Basisverzeichnis für die beiden ersten Angaben ist das Arbeitsverzeichnis, aus dem heraus die Kommandozeile aufgerufen wird.

Hauptschalter

/M	Binärdatei mit der Endung .TIn erzeugen. n Prozessnummer; siehe Schalter /PNx.
/L	Library-Datei (Bibliothek) mit der Endung .TLx erzeugen. x Prozessortyp; siehe Schalter /Px.
/MAKE	Hauptschalter, Dateiname und weitere Schalter für einen Aufruf aus dem makefile entnehmen. Der Text im makefile kann über mehrere Zeilen verteilt geschrieben werden. Schalter außerhalb des makefile sind nicht erlaubt.
/H	Kurzen Hilfetext anzeigen.
/VER	Versionsnummer des Compilers anzeigen.

Schalter

<code>src.bas</code>	Dateiname des zu kompilierenden Quelltextes; mit Dateieindung <code>.bas</code> angeben. Bei Hauptschalter <code>/M</code> nach <code>/PROCESS</code> angeben. Compiler-Warnungen werden in die Datei <code>src.wrn</code> geschrieben, Fehlermeldungen in die Datei <code>src.err</code> .
<code>/A"dest"</code>	[Pfad und] Name der zu erzeugenden Datei <code><dest></code> <i>ohne</i> Dateieindung. Voreinstellung ist der Dateiname <code>src</code> . Die Dateieindung <code>.TIn</code> (Binärdatei) oder <code>.TLx</code> (Library-Datei) wird automatisch angehängt.
<code>/IP"path"</code>	Verzeichnis, in dem nach Include-Dateien gesucht wird. Die Einstellung überschreibt den Standard-Pfad von <i>TiCoBasic</i> und sollte mit Bedacht eingesetzt werden.
<code>/LP"path"</code>	Verzeichnis, in dem nach Library-Dateien gesucht wird. Die Einstellung überschreibt den Standard-Pfad von <i>TiCoBasic</i> und sollte mit Bedacht eingesetzt werden.
<code>/Lx</code>	Sprache, in der Fehlermeldungen und Warnungen ausgegeben werden. <code>/LE</code> Englisch. Default. <code>/LG</code> Deutsch
<code>/Sx</code>	Hardware einstellen, für die die Datei kompiliert wird: <code>/SGII</code> Gold II <code>//SPII</code> Pro II; Default
<code>/Px</code>	Prozessortyp, für den die Datei kompiliert wird: <code>/P1</code> <i>TiCo</i> -Prozessor 1
<code>/PROCESS</code>	Schlüsselwort für die Optionen des nächsten Quelltexts. Muss für jeden Quelltext wiederholt werden. Nur gemeinsam mit dem Hauptschalter <code>/M</code> .
<code>/ET</code>	Zeitgesteuerten Prozess erzeugen, siehe Kapitel 6.1.1 auf Seite 128 . Default. Schließt <code>/EE</code> und <code>/EN</code> aus.
<code>/EE</code>	Extern gesteuerten Prozess erzeugen, siehe Kapitel 6.1.2 auf Seite 129 ; erfordert die Optionen <code>/EEA</code> , <code>/EEM</code> , <code>/EEV</code> , <code>/EEO</code> . Schließt <code>/ET</code> und <code>/EN</code> aus.

/EEAn	Hardware-Adresse <i>n</i> (dezimal), die für das Event-Signal ausgewertet wird.
/EEMn	Maskenwert <i>n</i> (dezimal), mit dem die Adresse UND-verknüpft wird.
/EEVn	Vergleichswert <i>n</i> (dezimal).
/EEOx	Vergleichsoperator <i>x</i> : 1: < kleiner 2: = gleich 3: <= kleiner gleich 4: > größer 6: >= größer gleich 8: <> ungleich
/EN	Ungesteuerten Prozess erzeugen, siehe Kapitel 6.1.3 auf Seite 130 . Schließt /EE und /ET aus.
/PNx	Nummer <i>x</i> (1...4) des Prozesses. Default: 1.
/PH	Prozess mit hoher Priorität erzeugen. Default-Einstellung. Siehe auch Kapitel 6.1 auf Seite 128 .
/PL	Prozess mit niedriger Priorität erzeugen (nur für zeitgesteuerten Prozess). Siehe auch Kapitel 6.1 auf Seite 128 .
/PDx	Zykluszeit (Processdelay) des Prozesses auf <i>x</i> Bytes einstellen. Default: 3000. Siehe auch Kapitel 6.2.1 auf Seite 131 .
/Ox	Optimierungsstufe <i>x</i> (0, 1, 2) für die Kompilierung einstellen, siehe auch Dialogfenster „Process Options“ (Seite 60) . /O0 Optimierungsstufe 0 (=keine Optimierung) /O1 Optimierungsstufe 1 (Default) /O2 Optimierungsstufe 2 /O3 Optimierungsstufe 3 (nur mit Option /P12) /Os Geringe Speichergröße (nur mit Option /P12) /Of Hohe Geschwindigkeit (nur mit Option /P12)
/Vx	Version <i>x</i> des Prozesses einstellen, siehe Dialogfenster „Process Options“ (Seite 60) . Default: 1.

A.4.2 Bemerkungen

Die Reihenfolge der Schalter ist beliebig. Bei den Schaltern wird Groß-/Kleinschreibung unterschieden, bei Dateinamen nicht.

Wenn der Schalter `/A` nicht verwendet wird, wird die erzeugte Binärdatei oder Library-Datei in dem Verzeichnis gespeichert, in dem sich der Quelltext befindet.

Treten während der Kompilierung Warnungen oder Fehler auf, werden sie in die Dateien `src.wrn` und `src.err` geschrieben. Die Fehlermeldungen sind identisch mit denjenigen, die *TiCoBasic* im Infofenster (siehe [Kapitel 3.11.1](#)) ausgeben würde.

Die Dateien werden in dem Verzeichnis gespeichert, wo sich der Quelltext `src.bas` befindet. Wenn Sie den Schalter `/A` verwenden, werden die Dateien in dem Verzeichnis gespeichert, wo die Binär- oder Library-Datei erzeugt wird.

Wir empfehlen, vor dem Kompilieren Dateien mit Warn- und Fehlermeldungen zu löschen, damit Sie nach dem Kompilierungsvorgang einfach prüfen können, ob dieser fehlerlos abgelaufen ist.

A.4.3 Beispiele

```
C:\ADwin\TiCoBasic\TiCoBasiccompiler.exe /L
Z:\Myfiles\test.bas
```



Diese Kommandozeile kompiliert den Quelltext `<test.bas>` und erzeugt die Library-Datei `<test.TL1>` im Verzeichnis `<Z:\Myfiles\>`.

Da nichts anderes angegeben ist, werden alle Standardeinstellungen verwendet:

- erzeugte Datei im Verzeichnis der Quelldatei speichern
- englische Warnungen und Fehlermeldungen.
- Hardware: *ADwin-Pro II*.
- *TiCo*-Prozessor 1.
- Optimierungsstufe 1.

Wenn Sie sich beim Aufruf im Verzeichnis `<C:\ADwin\TiCoBasic>` befinden, können Sie obige Zeile kürzer schreiben:

```
TiCoBasiccompiler.exe /L Z:\Myfiles\test.bas
```

Die kürzeste Aufruf-Variante ergibt sich, wenn auch der Quelltext im Verzeichnis `<C:\ADwin\TiCoBasic>` liegt:

```
TiCoBasiccompiler /L test.bas
```

Wir empfehlen in jedem Fall – speziell für eine Automatisierung des Aufrufs – die vollständige Variante:

```
TiCoBasicCompiler /L test.bas /A"test" /LE /SPII /P1 /O1
```



```
TiCoBasicCompiler /M /LE /SGII /P1 /PROCESS  
TiCo_inc_Par_1.bas /ET /PN3 /PH /O1
```

Kompiliert die Demo-Datei <TiCo_inc_Par_1> in eine Binär-Datei für ein Gold II-System mit TiCo-Prozessor; der Prozess ist zeitgesteuert, hat die Nummer 3 und hohe Priorität.



```
TiCoBasicCompiler /M /A"Y:\somewhere\your_file" /LE /SGII  
/P1  
/PROCESS C:\user\my_file.bas /ET /PN3 /PH /O1
```

Die Binärdatei heißt nun <your_file.TL1> und befindet sich im Verzeichnis <Y:\somewhere>; der Prozess ist zeitgesteuert, hat die Nummer 3 und hohe Priorität.

A.4.4 Kommandozeile unter Windows

Der Begriff und die Funktionalität „Kommandozeilen-Aufruf“ stammen noch aus der DOS-Zeit, in der man Befehle an das Betriebssystem DOS in einer Kommandozeile eingeben musste. Solche Eingaben sind auch unter Windows nach wie vor möglich und eignen sich z.B. für die Automatisierung von Abläufen.

Sie haben mehrere Möglichkeiten, Kommandozeilen unter Windows einzugeben:

- Öffnen Sie unter Windows ein MS-DOS-Fenster (Windows Start-Menü, Verzeichnis Programs / Accessories). Jede Eingabe ist hier eine Kommandozeile.



Der Compiler-Aufruf erfordert in jedem Fall die Windows-Umgebung. Der Aufruf funktioniert also nur aus diesem Windows-Fenster, nicht aus der originären DOS-Ebene.

- Wählen Sie im Start-Menü den Menüeintrag „Run“ und geben Sie im Eingabefenster eine Kommandozeile ein.
- Legen Sie für öfter benötigte Kommandozeilen-Aufrufe jeweils ein Icon auf dem Desktop an. Bei der Erstellung eines Icon geben Sie eine Kommandozeile direkt ein.

Ein oder mehrere Kommandozeilen-Aufrufe können in einer Batch-Datei `<*.bat>` zusammengefasst werden, z.B. um mehrere Quelltexte eines Projekts mit nur einem Aufruf zu kompilieren.

Bei jedem Aufruf müssen Sie die jeweils passenden Schalter und Parameter übergeben.

A.5 Befehle für ADwin-Gold II

Im folgenden sind die in *TiCoBasic* verfügbaren Befehle für *TiCo*-Prozessoren aufgeführt. Befehle zur Steuerung der Ein- und Ausgänge finden Sie in der Hardware-Dokumentation.

+ (Addition)
 - (Subtraktion)
 * (Multiplikation)
 / (Division)
 ^ (Potenz)
 = (Zuweisung)
 : Doppelpunkt
 #Begin_Debug_Mode_Disable
 #Define
 #End_Debug_Mode_Disable
 #If ... Then ... {#Else ...} #EndIf
 #Include
 #..., Compiler-Anweisung

A

Absl
 And

D

Data_n
 Dec
 Dim
 Do ... Until

E

End
 Event:

F

Finish:
 For ... To ... {Step ...} Next
 Function ... EndFunction

G-L

If ... Then ... {Else ...} EndIf
 Import
 In
 Inc
 Init:
 Lib_Function ... Lib_EndFunction
 Lib_Sub ... Lib_EndSub

M-Q

Max_Long
 Min_Long
 Mod
 NOP
 NOPS
 Not
 Or
 Out
 Processdelay
 ProcessN_Running
 Process_Error

R

Read_Timer
 Refresh_RingBuffer
 Rem
 Ringbuffer
 RingBuffer_Clear
 RingBuffer_Empty
 RingBuffer_Full

S

SelectCase
 Shift_Left

Shift_Right
Sleep
Sub ... EndSub

XOr

Symbole

< = > (Vergleich)

T-Z

A.6 Befehle für ADwin-Pro

Im folgenden sind die in *TiCoBasic* verfügbaren Befehle für *TiCo*--Prozessoren aufgeführt. Befehle zur Steuerung der Ein- und Ausgänge finden Sie in der Hardware-Dokumentation.

+ (Addition)
 - (Subtraktion)
 * (Multiplikation)
 / (Division)
 ^ (Potenz)
 = (Zuweisung)
 : Doppelpunkt
 #Begin_Debug_Mode_Disable
 #Define
 #End_Debug_Mode_Disable
 #If ... Then ... {#Else ...} #EndIf
 #Include
 #..., Compiler-Anweisung

A

Absl
 And

D

Data_n
 Dec
 Dim
 Do ... Until

E

End
 Event:

F

Finish:
 For ... To ... {Step ...} Next
 Function ... EndFunction

G-L

If ... Then ... {Else ...} EndIf
 Import
 In
 Inc
 Init:
 Lib_Function ... Lib_EndFunction
 Lib_Sub ... Lib_EndSub

M-Q

Max_Long
 Min_Long
 Mod
 NOP
 NOPS
 Not
 Or
 Out
 Processdelay
 ProcessN_Running
 Process_Error

R

Read_Timer
 Refresh_RingBuffer
 Rem
 Ringbuffer
 RingBuffer_Clear
 RingBuffer_Empty
 RingBuffer_Full

S

SelectCase
Shift_Left
Shift_Right
Sleep
Sub ... EndSub

T-Z

XOr

Symbole

< = > (Vergleich)

A.7 Index

Symbole

- · 142
- # · 146
- #Begin_Debug_Mode_Disable · 153
- #Define · 156
- #Else · 173
- #End_Debug_Mode_Disable · 163
- #EndIf · 173
- #If · 173
- #Include · 179
- * · 143
- + · 141
- .gotolink TiCoBasic_Gold2-Thema_ger.fm Gold2T · 18
- / · 144
- : · 147
- < = > · 149
- = · 148
- ^ · 145
- ' (Rem) · 205

Zahlen

- 150h, siehe Device No

A

- Abbruch, siehe Prozess beenden
- Absl · 150
- Absolutwert
 - Ganze Zahlen · 150
- ActiveX
 - Kommunikation zum ADwin-System · 136
- ADbasic
 - Demo-Modus · 12
- Add Open Files to Project · 76

- Add to Project
 - Kontextmenü · 21
 - Projektfenster · 76
- Addition · 141
- ADtools
 - Leiste einstellen · 69
- ADtools · 89
- ADWIN_GOLDII · 173
- ADWIN_PROII · 173
- ADWIN_SYSTEM · 173
- Align comments · 64
- Analyse
 - Laufzeitfehler · 124
- And · 151
- Anhalten
 - TiCo-Prozessor · 14
- Anhalten, siehe Prozess beenden
- Anmerkungen, siehe Kommentar
- Anzeige
 - aktuelle Informationen · 18
 - Auslastung: CPU, PM, DM, DX · 82
 - Befehlsparameter · 46
 - Prozessoptionen · 17
 - ToDo-Liste · 85
- Arithmetische Funktionen
 - · 142
 - * · 143
 - + · 141
 - / · 144
 - ^ · 145
 - Dec · 155
 - Inc · 178
- Array-Index (lokal) zu groß / < 1, siehe Laufzeitfehler, erkennen
- Arrays, siehe Felder
- (Dim) As · 158
- ASCII-Zeichensatz · 4
- (Dim ...) At · 158

- Auslastung
 - Anzeige · 82
 - Definition · 132
- auswerten
 - Operatoren · 113
- Autoindent · 64
- Automatisch
 - Einrücken · 26
 - Formatieren · 26
 - vervollständigen, Befehle und Variablen · 44
- AutoSave · 57
- Autostart · 57
- B**
 - Bearbeitungszeit messen · 119
 - Bedienoberfläche · 15
 - Bedingter Sprung
 - If ... Then · 171
 - SelectCase · 216
 - Befehl
 - Automatisch vervollständigen · 44
 - Deklaration anzeigen · 47
 - selbst dokumentieren · 27
 - Übergabeparameter anzeigen · 46
 - zur Deklaration springen · 42
 - Befehle
 - TiCo-Prozessor · 229
 - Befehlsreferenz · 139
 - Befehls-Separator (:) · 147
 - Befehlszeile
 - Groß-/Kleinschreibung · 91
 - max. Zeilenlänge · 91
 - Begin_Debug_Mode_Disable · 153
 - benutzerdefinierte Befehle / Variablen
 - dokumentieren · 27
 - Übersicht · 94
 - Berechnungsausdrücke · 113
 - auswerten · 113
 - symbolische Namen · 94
 - Betriebssystem
 - laden, siehe Initialisieren
 - Verzeichnis einstellen · 68
 - Bibliothek
 - Allgemeines · 116
 - Ein-/ausfalten · 31
 - Einbinden · 175
 - erlaubte Zeichen im Namen · 94
 - Erzeugen
 - aus TiCoBasic · 58
 - erzeugen · 57
 - aus Kommandozeile · 9
 - Funktion · 182
 - kontra Makro · 117
 - Position im Programm · 95
 - Rekursion verboten · 117
 - Unterprogramm · 187
 - Verzeichnis einstellen · 68
 - Binärdatei
 - erzeugen · 57
 - aus Kommandozeile · 9
 - aus TiCoBasic · 58
 - siehe auch Bibliothek
 - übertragen zum TiCo-Prozessor · 49
 - Binäre Schreibweise · 98
 - Bits verschieben
 - nach links · 219
 - nach rechts · 220
 - Bookmark · 42
 - Bootloader · 128
 - Menüeintrag · 73
 - programmieren · 50
 - BTL-Datei: Verzeichnis einstellen · 68
 - Busy-Anzeige · 82

C

- Clear Parameter Scan · 48
- Code size · 84
- code snippets · 46
- Column mark · 64
- Comment Block · 27
- Compiler
 - Aufruf · 57
 - AutoSave · 57
 - Compilermeldung, Fehler / Status · 84
 - Fehlermeldung · 57
 - Kommandozeilen-Aufruf · 9
 - Optionen einstellen · 58
 - Prebuild, Postbuild · 70
- Compiler-Anweisungen
 - #Begin_Debug_Mode_Disable · 153
 - #Define · 156
 - #End_Debug_Mode_Disable · 163
 - #If ... Then · 173
 - #Include · 179
 - Überblick · 146
- Control block
 - im Kontextmenü · 21
 - markieren · 41
- Cursor-Position · 82

D

- Data_n
 - dimensionieren · 158
 - Globale Felder · 99
 - Übersicht · 154
- Data-Index (global) zu groß / <1,
siehe Laufzeitfehler, erkennen
- Dateiname
 - Bibliothek · 58
 - Binärdatei · 58

- Datenaustausch
 - zwischen ADwin CPU und TiCo-Prozessor · 136
 - zwischen PC und TiCo-Prozessor · 135
 - zwischen Prozessen · 134
- Datenspeicher
 - siehe auch Speicher
 - intern (DM) · 104
 - Übersicht, intern, extern · 103
 - Zusatzbedarf durch
 - Debug-Modus · 124
- Datenstrukturen
 - Globale Felder · 99
 - Globale Variablen · 98
 - Lokale Variablen und Felder · 102
 - Ringbuffer · 105
 - Übersicht · 96
- Datentypen
 - Übersicht · 97
- Datenverlust
 - beim Initialisieren · 14
 - beim Rücksetzen · 14
 - Ringbuffer · 106
- Datenwort: Nummerierung von Bits · 2
- Debug
 - #Begin_Debug_Mode_Disable · 153
 - #End_Debug_Mode_Disable · 163
 - Allgemein · 124
 - Debug-Modus anwenden · 124
 - Menü · 71
- Debug errors · 72
- Debug mode · 72
- Dec · 155
- Declarations · 87
- DEFINE, siehe #Define

Deklaration

- alle anzeigen · 48, 87
- anzeigen · 47
- siehe Dimensionierung
- zur ~ springen · 42

Dekrementieren · 155**Demo-Modus** · 12**Deutsch** · 68**Device No**

- Definition · 137
- einstellen · 59

Dezimale Schreibweise · 98**Dim** · 158**Dimensionierung**

- Befehl Dim · 158
- Position im Programm · 95
- Speicherbereich · 103

Directory siehe Verzeichnis**Division**

- durch 2 · 220
- einfache · 144
- ganzzahliger Rest · 228

DM, siehe Datenspeicher**Do ... Until** · 161**dokumentieren, Befehle / Variablen** · 27**DRAM_Extern**

- Dim · 158
 - siehe Externer Speicher · 104
- Druckeinstellungen**
- 67
-
- DX, siehe Externer Speicher**

E**Editor**

- General · 64
 - Print Settings · 67
 - Syntax Colors · 65
- Editor-Leiste**
- 23
-
- eigene Befehle / Variablen**
- dokumentieren · 27
- Ein-/ausfalten, Textbereich**
- 31

Einbinden

- Include-Datei · 179
- Library · 175

Einrückten

- Kommentar · 26
- Programmabschnitte · 64
- Textzeilen · 26

Else (If ...) · 171**End** · 162**End_Debug_Mode_Disable, siehe #End_Debug_Mode_Disable****EndFunction** · 167**EndIf (If ...)** · 171**ENDREGION** · 29**EndSelect (SelectCase ...)** · 216**EndSub** · 223**English** · 68**Entwicklungsumgebung**

- Leisten und Fenster · 15
- starten · 11
- Tastaturkürzel · 1

erlaubte Zeichen im Namen

- Makros, Libraries · 94

Ersetzen

- Beispiele · 38
- Reguläre Ausdrücke · 38
- Text · 33

Event

- extern gesteuert · 129
 - externes Signal: Aufruf · 127
 - nicht gesteuert · 130
 - Signalquelle einstellen · 62
 - verlorenes Signal
 - ein Prozess
 - zeitgesteuert · 133
 - extern gesteuerter Prozess · 134
 - Zeitdifferenz messen · 119
 - zeitgesteuert · 128
- Event:**
- 164
-
- Exklusiv-ODER-Verknüpfung**
- 226

Exponential-Schreibweise · 98

Externer Speicher

DX · 104

SDRAM · 103

SX · 104

externes Event-Signal · 127

Extremwert

Maximum Ganze Zahlen · 191

Minimum Ganze Zahlen · 192

F

F1: Hilfe aufrufen · 19

Faltbaren Textbereich

definieren · 29

Falten, Textbereiche · 31

Farbe einstellen · 65

Farbige Darstellung · 25

Fehler

durch Cut&Paste erzeugt · 56

geringere Optimierung

einstellen · 63

Fehlermeldung

des Compilers · 84

Laufzeitfehler · 72

Felder

Data_n · 154

erlaubte Zeichen im Namen · 102

globale · 99

erstes Element · 100

verwendete anzeigen · 48

initialisieren · 95

lokale · 102

erstes Element · 102

Ringbuffer · 206

Speicherbereich festlegen · 103

Übersicht · 96

Fenster

Compiler Options · 58

Debug errors · 72

Declarations · 87

Global Variables · 86

Info-Bereich · 83

Info-Fenster · 84

Parameter · 78

Process Options · 60

Project Settings · 69

Beschreibung · 70

General · 70

Prebuild, Postbuild · 70

Projekt · 76

Quelltext · 82

Quelltext-Statusleiste · 17

Statusleiste · 82

ToDo-Liste · 85

Toolbox · 76

Übersicht · 15

Quelltext-Informationen · 17

FIFO · 105

Finden · 33

Beispiele · 37

Deklaration von

Befehl/Variable · 42

Reguläre Ausdrücke · 38

Finish: · 165

Font

Farbe einstellen · 65

Größe einstellen · 64

Font size · 64

For ... Next · 166

formatieren, Text im Editor · 26

Function · 167

Funktion

- Allgemeines zu
 - Bibliotheken · 116
- Allgemeines zu Makros · 115
- Bibliothek (Lib_Function) · 182
- dokumentieren · 27
- erlaubte Zeichen im Namen · 94
- Makro · 167
- Position im Programm · 95

G

Ganze Zahlen

- Wertebereich · 97
- ganzzahliger Rest bei
 - Division · 228
- Gerätenummer, Definition · 137
- Get_Par · 230
- Get_Par_Block · 232
- Get_TiCo_RingBuffer · 234
- Get_TiCo_Status · 236
- GetData_Long · 237
- gleich = · 149
- Global Variables, Fenster · 86
- Globaldelay · 199
- globale Felder, siehe Felder, globale
- globale Variablen, siehe Variablen, globale
- Goto Line · 42
- Groß-/Kleinschreibung · 19
- größer als >, >= · 149
- Gutter · 64

H

Halt, siehe Prozess beenden

Hardware-Zugriff

- ADwin-Hardware · 93
 - Lesen · 177
 - Schreiben · 198
- Header · 67

Hexadezimale Schreibweise · 98

Hilfe

- stay on top · 69
- Hilfe aufrufen · 19
- Hilfsprogramme (ADtools) · 89

I

- If · 171
 - siehe auch #If · 173
- Import · 175
- In · 177
- Inc · 178
- Include · 179
- Include-Datei
 - Allgemeines · 115
 - einbinden · 179
 - Verzeichnis einstellen · 68
- Indent
 - manuell / automatisch · 26
 - Option TiCoBasic sections · 64
- Info-Bereich · 83
- Info-Fenster · 84
- Init: · 181
- Initialisieren · 14
- Initialisierung · 93
- Inkrementieren · 178
- Interner Speicher
 - Datenspeicher (DM) · 104
 - Gesamt (SRAM) · 103
 - Programm (PM) · 104

J

- Jump forward/backward · 43
- Jump to Declaration · 42

K

- kleiner als <, <= · 149
- Kommandozeilen-Aufruf · 9

Kommentar

- [positionieren](#) · 26
- [Syntax](#) · 205
- [Zeilen in ~ ändern](#) · 27

Kommunikation

- [zwischen ADwin CPU und TiCo-Prozessor](#) · 136
- [zwischen PC und TiCo-Prozessor](#) · 135
- [zwischen Prozessen](#) · 134
- [kompilieren](#) siehe [Compiler](#)
- [Konstante](#) · 94
- [Kontextmenü](#) · 21
- [Projektfenster](#) · 76
- [Kontrollstrukturen](#) · 114

L

- [Language](#) · 68
- [Laufzeitfehler](#)
 - [Anzeige](#) · 72
 - [erkennen](#) · 124
 - siehe auch [Debug-Modus](#)
- [Lib_EndFunction](#) · 182
- [Lib_EndSub](#) · 187
- [Lib_Function](#) · 182
- [Lib_Sub](#) · 187
- [Library](#)
 - [dokumentieren](#) · 27
 - [Funktion](#) · 182
 - [Import](#) · 175
 - siehe auch [Bibliothek](#)
 - [Unterprogramm](#) · 187
- [Lib-Verzeichnis einstellen](#) · 68
- [License key eingeben](#) · 12
- [Line numbers](#) · 64
- [Lizenzschlüssel eingeben](#) · 12
- [Lizenzvertrag](#) · 5
- [Load Bin File](#) · 73

Logische Funktionen

- [And](#) · 151
- [Not](#) · 195
- [Or](#) · 196
- [Shift_Left](#) · 219
- [Shift_Right](#) · 220
- [XOr](#) · 226

Long

- siehe [Ganze Zahlen](#)
- [lost events \(T12\)](#) · 132

M

- [Make Bin File, Make Lib File](#) · 57
- [Makro](#)
 - [Allgemeines](#) · 115
 - [dokumentieren](#) · 27
 - [Ein-/ausfalten](#) · 31
 - [erlaubte Zeichen im Namen](#) · 94
 - [Funktion](#) · 167
 - [kontra Bibliothek](#) · 117
 - [Position im Programm](#) · 95
- [Mark Controlblock](#) · 41
- [Matlab](#)
 - [Lizenzschlüssel](#) · 12
- [Max_Long](#) · 191
- [Maximum](#)
 - [Ganze Zahlen](#) · 191
- [Menü](#)
 - [auswählen](#) · 16
 - [Build](#) · 57
 - [Debug](#) · 71
 - [Edit](#) · 56
 - [File](#) · 55
 - [Help](#) · 74
 - [Leiste](#) · 53
 - [Options](#) · 58
 - [Tools](#) · 73
 - [View](#) · 56
 - [Window](#) · 74
- [Menüleiste](#) · 53
- [Messwertverlauf darstellen](#) · 89

Metazeichen · 38
Min_Long · 192
Minimum
 Ganze Zahlen · 192
Mod · 228
Mouse over · 64
Multiplikation
 einfache · 143
 mit 2 · 219

N

Namen
 Felder und lokale Variablen · 102
 Makros, Libraries · 94
negatives Vorzeichen · 114
Neues in TiCoBasic · 7
Next (For ...) · 166
NICHT · 195
None: ohne Event · 130
NOP · 193
Not · 195
Nummerieren, Zeilen · 64

O

ODER-Verknüpfung · 196
ohne Event · 130
Hypertext · 18
Operatoren
 And · 151
 auswerten · 113
 negatives Vorzeichen · 114
 Or · 196
 Priorität · 114
 XOr · 226

Optimierung
 Allgemein · 119
 Bearbeitungszeit messen · 119
 Konstanten statt Variablen · 121
 Polynom schneller
 berechnen · 145
 Registerzugriff · 120
 schneller messen · 121
 Speicherzugriff · 124
 Wartezeit einstellen · 122
 Wartezeit nutzen · 122

Optionen einstellen

ADtools · 69
Allgemein (Settings) · 64
Compiler · 58
Drucken · 67
Editor · 64
Hilfe · 69
Projekt · 69
 allgemein · 70
 Beschreibung · 70
 Prebuild, Postbuild · 70
Prozess · 60
Sprache · 68
strukturierte
 Befehlsdarstellung · 65
 Verzeichnisse · 68
Or · 196
Ordner siehe Verzeichnis
Out · 198
Outdent · 26

P

P2_Get_Par · 276
P2_Get_Par_Block · 278
P2_Get_TiCo_Bootloader_Status · 280
P2_Get_TiCo_RingBuffer · 281
P2_Get_TiCo_Status · 284
P2_GetData_Long · 285
P2_Process_Status · 288

- P2_RingBuffer_Empty · 290
- P2_RingBuffer_Full · 291
- P2_Set_Par · 292
- P2_Set_Par_Block · 294
- P2_Set_TiCo_RingBuffer · 296
- P2_SetData_Long · 299
- P2_TDrv_Init · 302
- P2_TiCo_Flash · 308
- P2_TiCo_Get_Process_Error · 306
- P2_TiCo_Get_Processdelay · 304
- P2_TiCo_Load · 310
- P2_TiCo_Restart · 312
- P2_TiCo_Set_Processdelay · 314
- P2_TiCo_Start · 316
- P2_TiCo_Start_Process · 317
- P2_TiCo_Stop · 319
- P2_TiCo_Stop_Process · 320
- P2_Workload · 322
- Par_n, globale Variablen · 98
- Parameter Scan · 48
- Parameter, siehe Variablen, globale
- Parameterfenster · 78
- Parse and format · 64
- Parse declarations · 64
- PM, siehe Programmspeicher
- Polynom, schneller
 - berechnen · 145
- Positionieren, Kommentar · 26
- Postbuild · 70
- Potenz · 145
 - in Polynom ersetzen · 145
- Präprozessor-Anweisungen
 - #Begin_Debug_Mode_Disable · 153
 - #Define · 156
 - #End_Debug_Mode_Disable · 163
 - #If ... Then · 173
 - #Include · 179
 - Überblick · 146
- Prebuild · 70
- Print layout · 67
- Priorität
 - Operatoren · 114
- Probleme, langsamer Editor · 64
- Process_Error · 201
- Process_Running · 202
- Process_Status · 240
- Processdelay
 - Syntax · 199
 - Zeitverhalten · 131
- Processor · 173
- Programm verbessern, siehe Optimierung
- Programmabschnitte
 - Event: · 93
 - Finish: · 93
 - Init: · 93
 - Übersicht · 93
- Programmspeicher
 - PM · 104
 - Zusatzbedarf durch
 - Debug-Modus · 124
- Programmstruktur
 - Übersicht · 114
- Bibliothek
 - Lib_Function · 182
 - Lib_Sub · 187
 - Übersicht · 116
- Ein-/ausfalten · 31
- Include-Datei · 115
- Kommentar Rem · 205
- Makros
 - Funktion Function · 167
 - Übersicht · 115
 - Unterprogramm Sub · 223
- Schleife
 - Do ... Until · 161
 - For ... Next · 166
- Sprung
 - If ... Then · 171
 - SelectCase · 216

- Projekt
 - Optionen
 - einstellen · 69
 - allgemein · 70
 - Beschreibung · 70
 - Prebuild,
Postbuild · 70
 - Projektfenster · 76
 - Projektverwaltung
 - Allgemeines · 51
 - Fenster · 76
 - verwendete Variablen anzeigen · 48
 - Prozess
 - Bearbeitungszeit · 132
 - beenden
 - sich selbst (in Event:) · 162
 - Betriebszustände beim Zeitverhalten · 133
 - extern gesteuert · 129
 - Fehler auslesen · 201
 - Kommunikation zwischen ~en · 134
 - nicht gesteuert · 130
 - optimieren, siehe Optimierung
 - Optionen
 - einstellen · 60
 - Optionen, Anzeige · 17
 - Status ermitteln · 202
 - zeitgesteuert
 - Priorität hoch · 128
 - Priorität niedrig · 138
 - Zeitverhalten · 131
 - Zyklus, siehe Prozesszyklus
 - Prozessor, siehe Processor · 173
 - Prozess-Steuerung
 - End · 162
 - Process_Error · 201
 - ProcessN_Running · 202
 - Prozesszyklus
 - Aufruf
 - mit Event-Signal · 127
 - Zeiteinheit · 131
 - Punkt vor Strich, siehe Operatoren
- Q**
- Quelltext
 - Editor-Fenster · 82
 - erstellen · 19
 - farbig darstellen · 25
 - formatieren · 25
 - im Projekt verwenden · 76
 - Reiter verschieben · 19
 - ToDo-Liste · 85
 - übersetzen · 23
 - Quelltext-Statusleiste · 17
 - Quick info · 64
- R**
- Read_Timer · 203
 - Rechenzeit sparen
 - Konstanten statt Variablen · 121
 - Registerzugriff · 120
 - schneller messen · 121
 - Wartezeit einstellen · 122
 - Wartezeit nutzen · 122
 - redo · 23
 - Refresh_RingBuffer · 208
 - REGION · 29
 - Registerzugriff · 120
 - Reguläre Ausdrücke · 38
 - Reiter, Reihenfolge verändern · 19
 - Rekursion bei Bibliothek · 117
 - Rem · 205
 - Reset
 - TiCo-Prozessor · 14
 - Rest, ganzzahliger bei Division · 228

Ringbuffer

- Aufbau der Datenstruktur · 105
- Datenverlust · 106
- dimensionieren · 158
- Elementanzahl prüfen · 107
- Übersicht · 206
- RingBuffer_Clear · 210
- RingBuffer_Empty · 212, 242
- Ringbuffer_For_Read · 206
- Ringbuffer_For_Write · 206
- RingBuffer_Full · 214, 243
- Ringspeicher · 105

S

- Save All Files of Project · 76
- Schleife, Do ... Until · 161
- Schreibweise von Zahlen · 98
- Schriftart einstellen · 65
- SDRAM, siehe Externer Speicher, SDRAM
- SelectCase · 216
- Separator : · 147
- Set_Par · 244
- Set_Par_Block · 246
- Set_TiCo_RingBuffer · 248
- SetData_Long · 250
- Shift_Left · 219
- Shift_Right · 220
- Short-Cuts · 1
- Show Declarations · 48
- Show line numbers · 64
- Sleep · 221
- Smart format · 26
- snippets · 46

Speicher

- Auslastung · 82
- Bedarf bestimmen · 84
- Bereich festlegen · 103
- Bereiche (PM, DM, DX) · 103
- siehe auch Datenspeicher
- Zusatzbedarf durch
 - Bibliotheken · 116
 - Debug-Modus · 124
 - Makros · 115

Springen

- zur Deklaration von Befehl/Variable · 42

springen

- anderes Sprungziel · 43
- zu Zeile · 42

Sprung, bedingter

- If ... Then · 171
- SelectCase · 216

Sprungziel, wechseln zwischen · 43

SRAM, siehe Interner Speicher, Gesamt (SRAM)

Stack size

- bis T11 · 84

Starten, TiCoBasic · 11

Statusleiste · 82

Statusmeldung Compiler · 84

stay on top, Hilfefenster · 69

Step (For ...) · 166

Stopp, siehe Prozess beenden

Stoppen

- TiCo-Prozessor · 14

Strukturieren

- Ein-/ausfalten · 31
- Farbige Darstellung · 25
- Programmabschnitte · 114
- Zeilen einrücken · 26

Sub · 223

Subtraktion · 142

Suchen

- Beispiele · 37
- Deklaration von
 - Befehl/Variable · 42
- Reguläre Ausdrücke · 38
- schnell · 32
- Text · 33

SX, siehe Externer Speicher
symbolische Namen · 94

Syntax

- Colors · 65
- Farbige Darstellung · 25

Systemvariablen

- Process_Error · 201
- Processdelay · 199
- ProcessN_Running · 202
- Übersicht · 101

T

Tabsize · 64

Tabulator

- Schrittweite einstellen · 64

Tastatur

- Anzeige von Einstellungen · 82
- Kürzel · 1

TDrv_Init · 253

Terminierung, siehe Prozess beenden

Text

- formatieren · 26
- schnell suchen · 32
- suchen und ersetzen · 33
- Textbausteine · 46
- Textmarke · 42

Textbausteine einfügen · 46

Textbereich ein-/ausfalten · 31

Textbereich, als faltbar
definieren · 29

Textmarke · 42

Then (If ...) · 171

TiCo_Flash · 255

TiCo_Get_Process_Error · 259

TiCo_Get_Processdelay · 257

TiCo_Load · 261

TiCo_Reset_Mode · 264

TiCo_Restart · 263

TiCo_Set_Processdelay · 265

TiCo_Start · 267

TiCo_Start_Process · 268

TiCo_Stop · 270

TiCo_Stop_Process · 271

TICO1 · 173

TICO2 · 173

TiCoBasic

- Lizenzvertrag · 5
- starten · 11
- Verzeichnisse einstellen · 68

TiCoBasic, Lizenzschlüssel · 12

TiCoBasic: Unterschiede zu
ADbasic · 7

TiCoBasicCompiler · 9

TiCo-Bootloader

- Menüeintrag · 73
- programmieren · 50

TiCo-Prozessor

- Reset / Stopp · 14

Timer, siehe Zähler

Timer-Event · 127

To (For ...) · 166

ToDo-Liste · 85

Toolbox · 76

Tools

- Binärdatei laden · 73
- Bootloader · 73
- TGraphTiCo aufrufen · 73

U

Übergabeparameter anzeigen · 46

Überlastung des Prozessors · 132

umsteigen auf TiCoBasic · 7

Uncomment Block · 27

undo · 23

UND-Verknüpfung · 151
ungleich <> · 149
Unmark Controlblock · 41
Unterprogramm
 Allgemeines zu
 Bibliotheken · 116
 Allgemeines zu Makros · 115
 Bibliothek (Lib_Sub) · 187
 dokumentieren · 27
 erlaubte Zeichen im Namen · 94
 Makro (Sub) · 223
 Position im Programm · 95
Until (Do ...) · 161

V

Variablen
 Automatisch
 vervollständigen · 44
 Deklaration anzeigen · 47
 Deklaration finden · 42
 dokumentieren · 27
 globale · 98
 Anzeige · 78
 verwendete anzeigen · 48
 Werte hexadezimal
 anzeigen · 79
 initialisieren · 95
 Initialisierung · 14
 lokale · 102
 erlaubte Zeichen im
 Namen · 102
 Länge des Namens · 102
 Speicherbereich
 festlegen · 103
 symbolische Namen · 94
 Übersicht · 96
 vordefinierte Namen · 96
 siehe auch Systemvariablen
Vergleich
 < = > · 149

vergleiche Makros mit
 Bibliotheken · 117
Verlauf der Sprungziele · 43
verlorene Zyklen (T12) · 132
Versionsverwaltung
 Versionsnummer einstellen · 63
Verzeichnisse einstellen · 68

W

Warten
 Prozessor: NOP · 193
 Sleep · 221
 Wartezeit genau einstellen · 122
Werkzeuge (*ADtools*) · 89
Werkzeugleiste · 16
Wertebereich · 97
Workload · 273
Workspace size · 84

X

XOr · 226

Z

Zahlenwerte, Schreibweise · 98
Zähler
 auslesen · 203
 interner, Taktzyklus · 131
Zeile
 einrücken · 26
 formatieren · 26
 max. Länge · 91
 nummerieren · 64
 zu ~ springen · 42
Zeilenkommentar einrücken · 26
Zeit
 Zeitdifferenz messen · 119
 Zykluszeit · 131
Zeitdifferenz messen · 119
Zeitoptimierung, siehe Optimierung

Zeitverhalten

Änderung durch

[Debug-Modus](#) · 124

Betriebszustände

[allgemein](#) · 133[ein Prozess](#)[zeitgesteuert](#) · 133[extern gesteuerter](#)[Prozess](#) · 134[zuweisen, Zahlen](#) · 98[Zuweisung \(=\)](#) · 148

Symbole		Exp	220	P	
< = > (Vergleich)	186	FFT	337	P1_Sleep	278
+ (Addition)	175	FFT_Calc	347	P2_Sleep	280
+ (String-Addition)	176	FFT_Calc_DM	349	Peek	282
- (Subtraktion)	178	FFT_Calc_DX	351	Poke	283
* (Multiplikation)	179	FFT_Init	346	Processdelay	284
/ (Division)	180	FFT_Mag	341	ProcessN_Running	288
^ (Potenz)	181	FFT_Mag_Scale	345	Process_Error	287
= (Zuweisung)	185	FFT_Phase	343	R	
: Doppelpunkt	184	FFT_Scale	339	Read_Timer	289
" " (String)	312	FIFO	221	Read_Timer_Sync	291
#Begin_Debug_Mode_		FIFO_Clear	223	Rem	292
Disable	196	FIFO_Empty	225	Reset_Event	293
#Define	208	FIFO_Full	226	Restart_Process	294
#End_Debug_Mode_		Finish:	228	Round	295
Disable	216	Flo40ToStr	232	S	
#If ... Then ... {#Else ...}		FloToStr	230	SelectCase	296
#EndIf	242	For ... To ... {Step ...}		Shift_Left	299
#Include	247	Next	234	Shift_Right	300
#..., Compiler-Anweisung		Function ... EndFunction		Sin	302
183		236		Sleep	303
A-C		G-J		Sqrt	305
Abs	187	If ... Then ... {Else ...} En-		Start_Process	306
AbsF	188	dIf	240	Start_Process_Delayed	
AbsI	189	Import	244	308	
And	190	Inc	246	Stop_Process	310
ArcCos	192	Init:	249	" " (String)	312
ArcSin	193	IO_Sleep	251	StrComp	314
ArcTan	194	K-L		StrLeft	316
Asc	195	Lib_Function ... Lib_End-		StrLen	318
Cast_Float32ToLong	199	Function	253	StrMid	319
Cast_FloatToLong	197	Lib_Sub ... Lib_EndSub		StrRight	321
Cast_LongToFloat	198	258		Sub ... EndSub	323
Cast_LongToFloat32	200	LN	262	T-Z	
Chr	201	LngToStr	263	Tan	326
Cos	202	Log	265	User_Version	327
CPU_Sleep	203	LowInit:	266	ValF	328
D		M-O		Vall	330
Data_n	205	Max_Float	268	XOr	332
Dec	207	Max_Long	270		
Dim	210	Min_Float	269		
Do ... Until	214	Min_Long	271		
E-F		Mod	334		
End	215	NOP	274		
Event:	217	Not	275		
Exit	219	Or	276		

