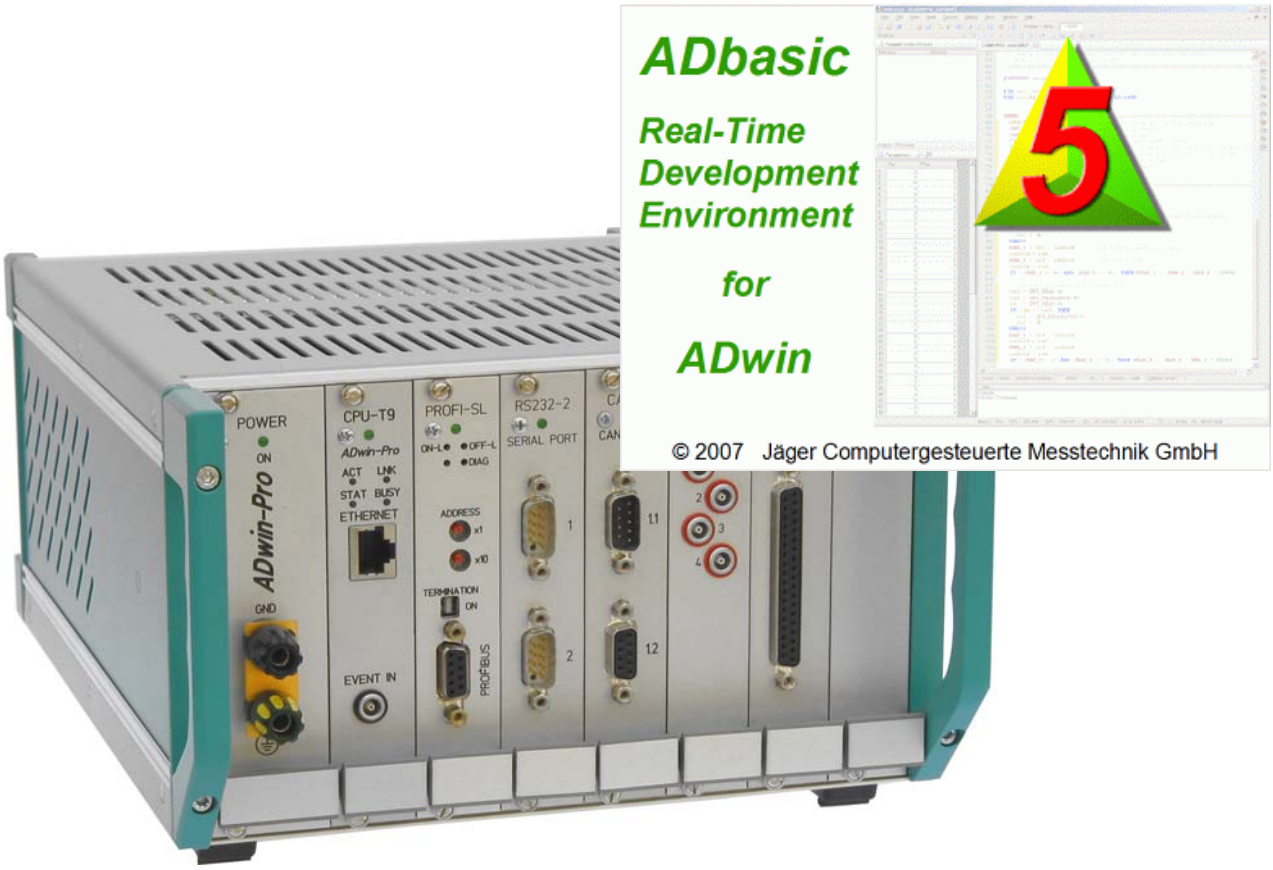
JÄGER
Computergesteuerte
Messtechnik GmbH

ADwin-Pro I

Systembeschreibung
Programmierung in *ADbasic*



ADwin-Pro I Software, Handbuch Feb. 2017

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

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

Inhaltsverzeichnis

Typografische Konventionen

Das „Achtung“-Zeichen steht bei Informationen, die auf Folgeschäden durch Fehlbedienung an der Hard- oder Software, am Messaufbau oder an Personen hinweisen.

Einen „Hinweis“ finden Sie bei

– Informationen, die für einen fehlerfreien Betrieb unbedingt beachtet werden müssen.

– Tipps und Ratschlägen für einen effizienten Betrieb.

Das Zeichen „Information“ verweist auf weiterführende Informationen in dieser Dokumentation oder andere Quellen wie Handbücher, Datenblätter, Literatur etc.

<C:\ADwin\…>

Dateinamen und -verzeichnisse sind in spitzen Klammern und im Schrifttyp Courier New angeben.

Programmtext

Programmanweisungen und Benutzer-Eingaben sind durch den Schrifttyp Courier New gekennzeichnet.

```
Var_1
```

Elemente eines Quelltextes wie Befehle, Variablen, Kommentar und sonstiger Text werden im Schrifttyp Courier New und farbig dargestellt.

In einem Datenwort (hier: 16 Bit) werden die Bits wie folgt nummeriert:

Bit-Nr.	15	14	13	…	1	0
Wert des Bits	2^{15}	2^{14}	2^{13}	…	2^1=2	2^0=1
Bezeichnung	MSB	-	-	-	-	LSB

1 Einführung

Mit dem Echtzeit-Entwicklungstool *ADbasic* steht Ihnen ein Werkzeug zur Verfügung, das die Programmierung des komplexen Prozessrechner-Systems *ADwin-Pro* einerseits denkbar einfach gestaltet und andererseits die „multi processing“ Fähigkeiten des Systems vollständig nutzt.

Dieses Handbuch beschreibt die *ADbasic*-Befehle zum Ansprechen der verschiedenen Module (Befehlsübersicht nach Modulen im Anhang).
Darüber hinaus beschreibt das *ADbasic*-Handbuch grundlegende Befehle z.B. für Berechnungen, den Aufbau der Programmstruktur oder das Steuern von Prozessen.

Die Befehle zum Ansprechen des *ADwin-Pro*-Systems aus *ADbasic* werden in Include-Dateien zur Verfügung gestellt. Die Include-Dateien finden Sie im Verzeichnis `<C:\ADwin\ADbasic\Inc>` (Standard-Installation).

Um den Zugriff auf die Module des *ADwin-Pro*-Systems zu ermöglichen, binden Sie mit folgender Zeile alle erforderlichen Include-Dateien in Ihr *ADbasic*-Programm ein:

```
#INCLUDE ADwinPro_All.Inc
```

Wenn Sie bereits *ADbasic*-Programme geschrieben haben, haben Sie dort für jede Modulgruppe eine eigene Include-Datei eingebunden. Löschen Sie die Include-Zeilen vollständig und fügen Sie stattdessen nur die oben stehende Zeile ein.

Bitte beachten Sie folgende Hinweise

Damit Ihr *ADwin*-System sicher arbeitet, halten Sie sich an die Informationen dieser und weiterführender Dokumentationen, auf die hier verwiesen wird.

Einschränkung der Anwendergruppe

Der Hersteller des in dieser Dokumentation beschriebenen Systems geht davon aus, dass an dem Gerät nur qualifiziertes Personal arbeitet.

> *Qualifiziertes Personal sind Personen, die aufgrund ihrer Ausbildung, Erfahrung und Unterweisung sowie ihrer Kenntnisse über einschlägige Normen, Bestimmungen, Unfallverhütungsvorschriften und Betriebsverhältnisse von dem für die Sicherheit der Anlage Verantwortlichen berechtigt worden sind, die jeweils erforderlichen Tätigkeiten auszuführen und die dabei mögliche Gefahren erkennen und vermeiden können. (Definition für Fachkräfte nach VDE 105 und IEC 60364).*

Verfügbarkeit der Unterlagen

Diese Produktdokumentation und Unterlagen, auf die verwiesen wird, müssen stets verfügbar sein und konsequent beachtet werden. Für Schäden, die durch Missachtung der Informationen in dieser bzw. der weiterführenden Dokumentation entstehen, übernimmt die Firma *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch, keine Haftung.

Rechtliche Grundlagen

Änderungen vorbehalten.

Hotline-Adresse siehe vordere Umschlagseite, innen.

2 Das Programm ADpro.exe

Das Programm <ADpro.exe> erfüllt eine Reihe von Aufgaben:

– Bestückung eines *ADwin-Pro* Systems anzeigen sowie Informationen zu den Modulen.

– Moduladresse für Pro II-Module einstellen (siehe Hardware-Handbuch).

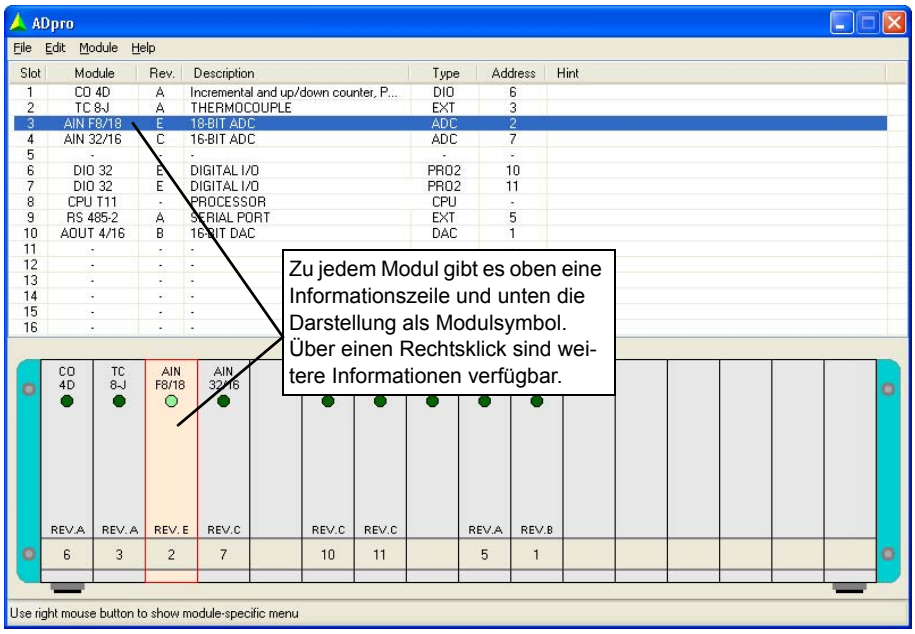
  Bei Pro I-Modulen wird die Moduladresse manuell eingestellt; im Programm wird die Adresse nur angezeigt.

– Funktion von Pro I- und Pro II-Modulen prüfen: analoge Ein-/Ausgangsmodule, Digital- und Zählermodule, einige Busmodule.

– Pro I- und Pro II-Module kalibrieren (analoge Ein-/Ausgangsmodule).

  Die Kalibrierung erfüllt nur einfache Ansprüche.

Die (interaktive) Anwendung des Programms ADpro ist selbsterklärend; manche Funktionen sind über das Kontextmenü (rechte Maustaste) verfügbar. Achten Sie bei Unklarheiten auf die Begleittexte und folgen Sie den Hinweisen.



Hinweise

ADpro.exe initialisiert das *ADwin*-System, d.h. es beendet und löscht noch laufende Prozesse.

Wenn beim Programmstart eine Fehlermeldung auftritt, prüfen Sie bitte, ob auf Ihrem Rechner das Programmpaket <Microsoft .NET Framework 2.0> installiert ist.

Das Erkennen von Modultypen, eine der Funktionen von ADpro.exe, wird manchmal auch als Funktion in ADbasic gewünscht. Es hat sich aber gezeigt, dass der organisatorische Aufwand für den Anwender den Nutzen bei weitem übersteigt: die notwendigen Modulinformationen müssten regelmäßig aktualisiert, ausgewertet und manuell in ADbasic-Programme eingepflegt werden. Daher ist die Funktion „Erkennen von Modultypen“ nur in ADpro.exe, nicht aber in ADbasic verfügbar.

3 *ADbasic*-Anweisungen für *ADwin-Pro I*-Module

Dieser Abschnitt enthält die Anweisungen zum Ansprechen der klassischen *ADwin-Pro I*-Module. Die Anweisungen sind zuerst nach Modulgruppen und dann alphabetisch sortiert.

Im Anhang finden Sie außerdem sortierte Befehls-Übersichten:

– Alphabetische Befehlsübersicht
– Befehlsübersicht nach Modulen

  Nutzen Sie diese Übersicht, um die Funktionen eines Moduls anhand der gültigen Befehle kennen zu lernen.
– Thematische Befehlsübersicht

Um eine Anweisung verwenden zu können, müssen Sie folgende Zeile am Anfang Ihres *ADbasic*-Programms einbinden:

```
#INCLUDE ADwinPRO_ALL.Inc
```

Zu jeder Befehlsbeschreibung gehören

– Syntax und Übergabeparameter.
– Bemerkungen über Besonderheiten.
– Liste verwandter Befehle.
– Liste der Module, auf welche der Befehl anwendbar ist.
– meistens ein Anwendungsbeispiel.

  Die Anwendungsbeispiele gehen in der Regel davon aus, dass auf dem Modul die Adresse 1 eingestellt ist.

3.1 Pro I: Alle Module

Dieser Abschnitt beschreibt folgende Befehle:

Manche Befehle für Pro I-Module benötigen u.a. die Angabe des Parameters `mod_class`, der die Modulklasse eines Moduls angibt. Sie können für diesen Parameter die folgenden Konstanten verwenden:

`dio`	= `000h`	für alle Digital- oder Zähler-Module
`ad`	= `040h`	für alle Analog/Digital-Wandler-Module
`da`	= `080h`	für alle Digital/Analog-Wandler-Module
`cpu`	= `0A0h`	für das Prozessor-Modul
`ext`	= `0C0h`	für alle übrigen Module

CheckLED

CheckLED gibt den Status der LED (oben auf der Frontplatte) auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = CheckLED(module,mod_class)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
mod_class	Modulklasse: Eine der Konstanten ad, da, dio, cpu oder ext.	LONG
ret_val	LED-Status: 0: LED ist aus (Default). 1: LED ist an.	LONG

Bemerkungen

Einige Module besitzen zusätzliche LED. Der Status der LED kann mit dieser Anweisung nicht zurückgegeben werden.

Siehe auch

SetLED

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B, AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, AOut-8/16 Rev. C, CAN-1, CAN-2, CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T, Comp-16 Rev. A, CPU-T10, CPU-T9, DIO-32 Rev. A, DIO-32 Rev. B, Inter-SL, LS-2 Rev. A, OPT-16 Rev. A, OPT-16 Rev. B, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO, PT100-4, PT100-8, PWM-4(-I), REL-16 Rev. A, REL-16 Rev. B, RS232-2, RS232-4, RS485-2, RS485-4, Storage Rev. A, TC-16, TC-4, TC-8, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  If (CheckLED(1,dio)=0) Then 'Falls LED aus ist …
    SetLED(1,dio,1)           '… dann LED einschalten
  EndIf
```

CPU_Digin

Nur Prozessoren T9, T10. **CPU_Digin** gibt zurück, ob seit dem letzten Aufruf der Anweisung eine fallende Flanke am Eingang Digin 0 des Prozessormoduls aufgetreten ist.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = CPU_Digin()
```

Parameter

ret_val	Statusmeldung, ob eine fallende Flanke am Eingang Digin 0 aufgetreten ist: 0: Fallende Flanke ist nicht aufgetreten. 1: Fallende Flanke ist ein- oder mehrfach aufgetreten.	LONG

Bemerkungen

Durch die Anweisung **CPU_Digin** wird die modulinterne Statusmeldung für fallende Flanken ausgelesen; dabei wird die Statusmeldung automatisch auf den Wert 0 zurückgesetzt.

Am Eingang Digin 0 werden TTL-Signale erwartet.

Siehe auch

CPU_Digin (T11)

Gültig für

CPU-T10, CPU-T9

Beispiel

```
#Include ADwinPro_All.Inc
Dim dummy As Long

Init:
  Rem Statusmeldung auslesen und dadurch zurücksetzen
  dummy = CPU_Digin()

Event:
  Rem …
  If(CPU_Digin() = 1) Then'Falls fallende Flanke aufgetreten …
    End                   '… dann das Programm beenden
  EndIf
  Rem …
```

EventEnable

EventEnable sperrt oder aktiviert den externen Event-Eingang auf dem angegebenen Modul.

Mit einem Signal an diesem Eingang kann der Zyklus eines *ADbasic*-Prozesses gesteuert werden.

Syntax

```
#Include ADwinPro_All.Inc
EventEnable(module,mod_class,enable)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
mod_class	Modulklasse: Eine der Konstanten ad, da, dio oder ext.	LONG
enable	Signalstatus: 0: externes Event-Signal sperren (Default). 1: externes Event-Signal zulassen.	LONG

Bemerkungen

Sie können einen hochprioren *ADbasic*-Prozess (d.h. dessen zyklischen Abschnitt **Event:**) durch ein externes Event-Signal aufrufen lassen und damit z.B. mit einem externen Prozess synchronisieren. Der *ADbasic*-Prozess muss dazu in *ADbasic* als extern gesteuert eingestellt sein (siehe Dialogfenster Process Options *ADbasic*-Handbuch, Process Options).

Die meisten Module verfügen über einen Event-Eingang. Sobald Sie mit **EventEnable** einen Eingang aktiviert haben, wird das anliegende Signal an das Prozessormodul weitergeleitet. Das Prozessormodul erkennt eine steigende Flanke als Event-Signal und der extern gesteuerte Prozess reagiert (falls vorhanden).

Der Event-Eingang eines Prozessor-Moduls ist immer aktiv und kann mit diesem Befehl nicht gesperrt werden. Der Event-Eingang an allen anderen Modulen ist nach dem Einschalten des Systems gesperrt.

Es spielt keine Rolle, welchen der Event-Eingänge Sie verwenden, denn alle Event-Signale gelangen beim Prozessormodul auf die gleiche Signalleitung. Aus diesem Grund dürfen Sie in einem System – neben dem immer aktiven Eingang am Prozessor-Modul – nur einen einzigen Event-Eingang aktivieren und müssen alle anderen deaktivieren.

Gültig für

CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T, DIO-32 Rev. A, DIO-32 Rev. B, OPT-16 Rev. A, OPT-16 Rev. B, PWM-4(-I), REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  Rem Externes Event am Digital-Modul 1 zulassen
  EventEnable(1,dio,1)

Event:
  Rem …
```

ResetWatchdogTimer

ResetWatchdogTimer setzt den Watchdog-Zähler des CPU-Moduls zurück auf den Startwert. Der Zähler bleibt aktiv.

Syntax

```
#Include ADwinPro_All.Inc
ResetWatchdogTimer()
```

Parameter

Keine

Bemerkungen

Solange der Watchdog-Zähler aktiv ist, dekrementiert er den Zählerstand kontinuierlich. Wenn der Zählerstand 0 (Null) erreicht, nimmt das System eine Fehlfunktion an und versetzt sämtliche Module in den Ursprungszustand (Power-On Status). Dadurch werden beispielsweise alle Programme gestoppt, Ein- und Ausgangsleitungen sowie Sollwerte zurückgesetzt.

Prozessortyp	Dauer
T9, T10, T11	1600 ms

Dauer des Zählvorgangs vom Startwert bis auf 0

Setzen Sie den aktiven Watchdog-Zähler während des Zählvorgangs mindestens einmal zurück auf den Startwert, um die Funktion Ihres Pro-Systems zu gewährleisten. Bei gesperrtem Zähler hat diese Anweisung keine Funktion.

Die Watchdog-Funktion dient zur Überwachung des *ADwin-Pro*-Systems.

Siehe auch

StartWatchdog, StopWatchdog

Gültig für

CPU-T10, CPU-T11, CPU-T9

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  StartWatchdog()           'Watchdog aktivieren
Event:
  ResetWatchdogTimer()      'Watchdog regelmäßig zurücksetzen
  Rem …
Finish:
  StopWatchdog()            'Watchdog deaktivieren
```

SetLED

`SetLED` schaltet die LED (oben auf der Frontplatte) auf dem angegebenen Modul ein oder aus.

Syntax

```
#Include ADwinPro_All.Inc
SetLED(module,mod_class,value)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`mod_class`	Modulklasse: Eine der Konstanten `ad`, `da`, `dio`, `cpu` oder `ext`.	LONG
`value`	Gewünschter Schaltzustand der LED. 0: ausschalten. 1: einschalten.	LONG

Bemerkungen

Auf einigen Modulen gibt es 2farbige LED (rot und grün). Für den Schaltzustand dieser LED müssen jeweils 2 Bits gesetzt werden:

Bitmuster	Schaltzustand
`00b`	aus
`01b`	leuchtet grün
`10b`	leuchtet rot
`11b`	aus

Das Modul Pro-Storage besitzt 3 2farbige LED, von denen 2 mit der Anweisung SetLED programmierbar sind. Die Zuordnung der Bits zu den LED ist wie folgt:

Bits in pattern	31:04	03:02	01:00
LED-Position	–	unten rechts	oben

Siehe auch

CheckLED

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B, AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, AOut-8/16 Rev. C, CAN-1, CAN-2, CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T, Comp-16 Rev. A, CPU-T10, CPU-T9, DIO-32 Rev. A, DIO-32 Rev. B, Inter-SL, LS-2 Rev. A, OPT-16 Rev. A, OPT-16 Rev. B, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO, PT100-4, PT100-8, PWM-4(-I), REL-16 Rev. A, REL-16 Rev. B, RS232-2, RS232-4, RS485-2, RS485-4, Storage Rev. A, TC-16, TC-4, TC-8, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  SetLED(1,cpu,1)           'LED am Prozessor-Modul 1 einschalten
  SetLED(1,ad,1)            'LED am A/D-Modul 1 einschalten
  SetLED(1,da,1)            'LED am D/A-Modul 1 einschalten
  SetLED(1,dio,1)           'LED am DIO-Modul 1 einschalten
  SetLED(2,dio,0110b)       'Am Modul Pro-Storage (DIO Nr. 2)
                            'obere
                            'LED rot und untere LED grün schalten
Event:
  Rem …

Finish:
  SetLED(1,cpu,0)           'LED am Prozessor-Modul 1 ausschalten
  SetLED(1,ad,0)            'LED am A/D-Modul 1 ausschalten
  SetLED(1,da,0)            'LED am D/A-Modul 1 ausschalten
  SetLED(1,dio,0)           'LED am DIO-Modul 1 ausschalten
  SetLED(2,dio,0)           'Alle LED am Storage-Modul
                            'ausschalten
```

StartWatchdog

`StartWatchdog` aktiviert den Watchdog-Zähler des CPU-Moduls und setzt ihn auf den Startwert.

Syntax

```
#Include ADwinPro_All.Inc
StartWatchdog()
```

Bemerkungen

Solange der Watchdog-Zähler aktiv ist, dekrementiert er seinen Zählerstand kontinuierlich. Wenn der Zählerstand 0 (Null) erreicht, nimmt das System eine Fehlfunktion an und versetzt sämtliche Module in den Ursprungszustand (Power-On Status). Dadurch werden beispielsweise alle Programme gestoppt, Ein- und Ausgangsleitungen sowie Sollwerte zurückgesetzt.

Prozessortyp	Dauer
T9, T10, T11	1600 ms

Dauer des Zählvorgangs vom Startwert bis auf 0

Setzen Sie den Watchdog-Zähler während des Zählvorgangs mindestens einmal zurück auf den Startwert, um die Funktion Ihres Pro-Systems zu gewährleisten (Befehl `ResetWatchdogTimer`).

Die Watchdog-Funktion dient zur Überwachung des *ADwin-Pro*-Systems.

Siehe auch

ResetWatchdogTimer, StopWatchdog

Gültig für

CPU-T10, CPU-T11, CPU-T9

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  StartWatchdog()           'Watchdog aktivieren
```

StopWatchdog

StopWatchdog deaktiviert den Watchdog-Zähler des CPU-Moduls.

Syntax

```
#Include ADwinPro_All.Inc
StopWatchdog()
```

Bemerkungen

Die Watchdog-Funktion dient zur Überwachung des *ADwin-Pro*-Systems. Sie kann mit StartWatchdog wieder aktiviert werden.

Siehe auch

ResetWatchdogTimer, StartWatchdog

Gültig für

CPU-T10, CPU-T11, CPU-T9

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  StartWatchdog()           'Watchdog aktivieren
  Rem …

Event:
  ResetWatchdogTimer()      'Watchdog zurücksetzen
  Rem …

Finish:
  StopWatchdog()            'Watchdog deaktivieren
```

SyncAll

`SyncAll` startet eine bestimmte Aktion synchron auf allen Modulen, die mit SyncEnable aktiviert wurden.

Syntax

```
#Include ADwinPro_All.Inc
SyncAll()
```

Bemerkungen

Je nach Modultyp wird auf allen mit `SyncEnable` aktivierten Modulen gleichzeitig (synchron) eine der folgenden Aktionen gestartet, die einem Standardbefehl entspricht. Es gelten die zuvor festgelegten Einstellungen, z.B. für Multiplexer, Ausgabewert oder Burst-Modus.

Modultyp	Aktion	Ersetzter Befehl
Analoge Eingänge	A/D-Wandlung auf allen ADC starten: `SyncEnable`(…, …, 1) oder Burst-Messreihe starten: `SyncEnable`(…, …, 3) und nur für Module Pro-AIn-F-x/14.	`Start_Conv` / `Start_ConvF` `Burst_Start`
Analoge Ausgänge	D/A-Wandlung starten mit dem Wert aus dem DAC-Register, siehe `WriteDAC`.	`Start_DAC`
Digitale Eingänge	Aktuellen Zustand der Eingänge in das Eingangs-Zwischenregister übertragen. Von dort auslesen mit `Dig_ReadLatch1/2`.	`Dig_Latch`
Digitale Ausgänge	Wert aus dem Ausgangs-Zwischenregister lesen und auf die digitalen Ausgänge schalten. Siehe `Dig_WriteLatch1/2`.	`Dig_Latch`
Zähler	Aktuelle Zählerstände in die Zähler-Zwischenregister übertragen. Von dort auslesen mit `Cnt_ReadLatch16`, `Cnt_ReadLatch32`, `CO4_ReadLatch`.	`Cnt_Latch`

Siehe auch

SyncEnable, SyncStat

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B, AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T, DIO-32 Rev. A, DIO-32 Rev. B, OPT-16 Rev. A, OPT-16 Rev. B, PWM-4(-I), REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Init:
  Rem Multiplexer der A/D Module 1-3 auf Eingang 1 setzen
  Set_Mux(1,0)
  Set_Mux(2,0)
  Set_Mux(3,0)
  Rem Synchronisation der A/D Module 1-3 aktivieren
  SyncEnable(1,ad,1)
  SyncEnable(2,ad,1)
  SyncEnable(3,ad,1)
  i=1                          'Index initialisieren

Event:
  Rem Wandlung für alle 3 Module synchron starten
  SyncAll()                    'Wandlung aller aktiven Module
                               'starten
  Wait_EOC(1)                  'Auf des Ende der Wandlung warten
  Data_1[i]=ReadADC(1)         'A/D Wandler Modul 1 auslesen
  Data_2[i]=ReadADC(2)         'A/D Wandler Modul 2 auslesen
  Data_3[i]=ReadADC(3)         'A/D Wandler Modul 3 auslesen
  If (i=1000) Then End         'Nach 1000 Durchläufen Prozess
                               'beenden
  Inc(i)                       'Index erhöhen
```

SyncEnable

SyncEnable aktiviert oder deaktiviert auf dem angegebenen Modul die Synchron-Option.

Syntax

```
#Include ADwinPro_All.Inc
SyncEnable(module,mod_class,enable)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
mod_class	Modulklasse: Eine der Konstanten ad, da, dio, cpu oder ext.	LONG
enable	Wert, um die Synchron-Option zu: 0: deaktivieren. 1: aktivieren (für „normale“ Aktion; siehe **SyncAll**). 3: aktivieren für Burst-Messreihe (nur für Module Pro-AIn-F-x/14).	LONG

Bemerkungen

Für jedes Modul muss die Synchron-Option separat aktiviert werden. Sie können beliebig viele Module synchronisieren.

Das Synchron-Signal wird mit **SyncAll** ausgelöst.

Siehe auch

SyncAll, SyncStat

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B, AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T, DIO-32 Rev. A, DIO-32 Rev. B, OPT-16 Rev. A, OPT-16 Rev. B, PWM-4(-I), REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_4[1000] As Long

Init:
  Rem Multiplexer der A/D Module 1-3 auf Eingang 1 setzen
  Set_Mux(1,0)              'Multiplexer auf Eingang 1 setzen
  Set_Mux(2,0)
  Set_Mux(3,0)
  Rem Synchronisation aktivieren: A/D Modul 1, DIO Module 1+2
  SyncEnable(1,ad,1)
  SyncEnable(1,dio,1)
  SyncEnable(2,ad,1)
  i=1                       'Index initialisieren

Event:
  Rem - Wandlung A/D Modul 1 starten
  Rem - Aktuellen Zustand der dig. Eingänge des Digital-
  Rem   I/O-Moduls 1 in das Eingangs-Zwischenregister
  Rem   übernehmen bzw. Wert aus dem Ausgangs-Zwischen-
  Rem   register auf die dig. Ausgänge geben
  Rem - Aktuelle Zählerstände der Zähler von Modul 2 in
  Rem   die Zähler-Zwischenregister übernehmen
  SyncAll()                 'Wandlung A/D Modul starten
  Wait_EOC(1)               'Auf des Ende der Wandlung warten
  Data_1[i]=ReadADC(1)      'A/D Wandler Modul 1 auslesen
  Rem Zwischenregister der DIO-Module auslesen
  Data_2[i]=Dig_ReadLatch1(1)
  Data_3[i]=CNT_Read32(2,1)'Zwischenregister Zähler 1
  Data_4[i]=CNT_Read32(2,2)'Zwischenregister Zähler 2
  If (i=1000) Then End      'Nach 1000 Durchläufen
                            ' den Prozess beenden
  Inc(i)                    'Index erhöhen
```

SyncStat

`SyncStat` gibt die Einstellung der Synchron-Option auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = SyncStat(module,mod_class)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`mod_class`	Modulklasse: Eine der Konstanten `ad`, `da`, `dio`, `cpu` oder `ext`.	LONG
`ret_val`	Einstellung der Synchron-Option: 0: deaktiviert. 1: aktiviert (für „normale“ Aktion; siehe `SyncAll`). 3: aktiviert für Burst-Messreihe (nur für Module Pro-AIn-F-x/14).	LONG

Siehe auch

SyncAll, SyncEnable

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B, AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, AOut-8/16 Rev. C, CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T, DIO-32 Rev. A, DIO-32 Rev. B, OPT-16 Rev. A, OPT-16 Rev. B, PWM-4(-I), REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000] As Long

Init:
  If (SyncStat(1,da)=0) Then 'Ist Synchron-Option deaktiviert?
    SyncEnable(1,da,1)       'Dann Synchronisation aktivieren für
    SyncEnable(2,da,1)       'D/A Module 1+2
  EndIf
  i=1                        'Index initialisieren

Event:
  WriteDAC(1,1,Data_1[i]) 'Ausgabe vorbereiten
  WriteDAC(2,1,Data_2[i])
  'Ausgabe auf den beiden Modulen synchron starten
  SyncAll()
  If (i=1000) Then End       'Nach 1000 Durchläufen Prozess
                             'beenden
  Inc(i)                     'Index erhöhen
```

3.2 Pro I: Analoge Eingänge

Dieser Abschnitt beschreibt folgende Befehle:

Analoge Eingänge mit Multiplexer

Analoge Eingänge mit Fast-ADC und Burst-Messreihe

Analoge Eingänge mit Fast-ADC

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für ein bestimmtes Modul anwendbar sind.

In den Anwendungsbeispielen wird davon ausgegangen, dass auf dem A/D-Modul die Adresse 1 eingestellt ist.

ADC

ADC führt eine komplette Messung auf dem 12 Bit-, 14 Bit- oder 16 Bit-ADC des angegeben Moduls durch.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ADC(module,input_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
input_no	Nummer des analogen Eingangs (je nach Modul: 1…8, 1…16 oder 1…32). Siehe auch Hinweise für die Module Pro-AIn-32/x.	LONG
ret_val	Wandlungsergebnis als 16 Bit Integer-Wert (0 … 65535). Bei 12 Bit ADC sind die 4 niederwertigsten Bits immer gleich 0, bei 14 Bit ADC die 2 niederwertigsten Bits.	LONG

Bemerkungen

Die Anweisung ADC ist eine Zusammenstellung aufeinander folgender Funktionen:

Set_Mux	→	…	→	Start_Conv	→	Wait_EOC	→	ReadADC
Multiplexer auf den gewünschten Eingangskanal setzen		Einschwingen des Multiplexers abwarten (3µs) bei AIn-x/16 ab Rev. B: 14µs		A/D-Wandlung starten		Ende der Wandlung abwarten		Gewandelten Wert auslesen

In folgenden Fällen sollten Sie anstelle der Anweisung **ADC** die oben genannten Anweisungen verwenden:

- Sehr kurze Zykluszeiten: **Processdelay** < 200.
- Hoher Innenwiderstand (>3kΩ) der Spannungsquelle des Messsignals: Dies verlängert die Einschwingzeit des Multiplexers über 3,0µs bzw. 14µs.
- Sie möchten unvermeidliche Wartezeiten für zusätzliche Programmschritte nutzen.
- Sie möchten die Verstärkung einstellen.

Diese Funktion darf nicht in zwei parallel laufenden Prozessen mit unterschiedlicher Priorität auf dasselbe A/D-Modul angewendet werden:

Ein niedrig priorisierter Prozess kann inmitten der **ADC**-Sequenz von einem höher priorisierten Prozess unterbrochen werden. Wenn der höher priorisierte Prozess den Multiplexer umstellt, liefert die Funktion im niedrig priorisierten Prozess den Messwert des falschen Kanals.
Mehrere parallel laufende hoch priorisierte Prozesse können die Anweisung **ADC** problemlos verwenden, da hoch priorisierte Prozesse nicht unterbrochen werden können.

Wenn die Module Pro-AIn-32/x mit differentiellen Eingängen betrieben werden (siehe **SE_Diff**), können für input_no die Nummern 1…8 und 17…24 angegeben werden. Die Nummern 9…16 und 25…32 werden auf diesen Modulen nur bei single-ended Eingängen benutzt.

Siehe auch

Set_Mux, Start_Conv, Wait_EOC, ReadADC, ADC16, SE_Diff

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#Include ADwinPro_All.Inc
Dim value As Long
Event:
  value = ADC(1, 4)          'Misst einen Wert am analogen Eingg. 4
```

ADC16

ADC16 führt eine komplette Messung auf einem 16 Bit ADC durch.
Die Angaben gelten ausschließlich für das Modul Pro-AIn-8/16 Rev. A.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ADC16(module,input_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
input_no	Nummer des analogen Eingangs (1…8).	LONG
ret_val	Wandlungsergebnis (0…65535).	LONG

Bemerkungen

Die Anweisung ADC16 gilt nicht für das Modul AIn-8/16 Rev. B (oder höher). Verwenden Sie stattdessen den Befehl ADC.

Die Anweisung ADC16 ist eine Zusammenstellung von aufeinander folgenden Funktionen:

Set_Mux	→ … →	Start_Conv	→ Wait_EOC →
Multiplexer auf den gewünschten Eingang setzen	Einschwingen des Multiplexers abwarten (5,0 µs)	A/D-Wandlung starten	Ende der Wandlung abwarten

→ Start_Conv	→ Wait_EOC	→ ReadADC
A/D-Wandlung starten	Ende der Wandlung abwarten	Gewandelten Wert auslesen

In folgenden Fällen sollten Sie anstelle der Anweisung ADC16 die oben genannten Anweisungen verwenden:

- Sehr kurze Zykluszeiten: Processdelay < 200 (s.o.).
- Hoher Innenwiderstand (>3kΩ) der Spannungsquelle des Messsignals: Dies verlängert die Einschwingzeit des Multiplexers über 3,0µs bzw. 14µs.
- Sie möchten unvermeidliche Wartezeiten für zusätzliche Programmschritte nutzen.
- Sie möchten die Verstärkung einstellen.

Wandlung 2fach starten

Das Starten der Wandlung auf dem ADC dieses Moduls löst parallel 2 Aktionen aus:

1. Der aktuell anliegende Spannungswert wird gewandelt und im ADC-Zwischenregister abgelegt.
2. Der Wert, welcher sich vor dem Wandlungsstart im Zwischenregister befand, wird seriell vom ADC zur Logik auf dem Modul übertragen und steht nach dem Ende der Konvertierung als Rückgabewert zur Verfügung.

Damit die Anweisung nicht den Wert der letzten Messung, sondern den aktuellen Wert liefert, muss die Wandlung also 2mal gestartet werden.

Diese Funktion darf nicht in zwei parallel laufenden Prozessen mit unterschiedlicher Priorität auf dasselbe A/D-Modul angewendet werden:

Ein niedrig priorisierter Prozess kann inmitten der ADC16-Sequenz von einem höher priorisierten Prozess unterbrochen werden. Wenn der höher priorisierte Prozess den Multiplexer umstellt, liefert die Funktion im niedrig priorisierten Prozess den Messwert des falschen Kanals.
Mehrere parallel laufende hoch priorisierte Prozesse können die Anweisung

ADC16 problemlos verwenden, da hoch priorisierte Prozesse nicht unterbrochen werden können.

Siehe auch

Set_Mux, Start_Conv, Wait_EOC, ReadADC, ADC

Gültig für

(LP)SH-8(-FI), AIn-8/16 Rev. A

Beispiel

```
#Include ADwinPro_All.Inc
Dim value As Long
Event:
  value = ADC16(1, 7)       'Misst einen Wert am analogen Eingg. 7
```

ADCF

ADCF führt eine komplette Messung auf einem Fast-ADC durch.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ADCF(module,input_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
input_no	Nummer des analogen Eingangs (1…4 oder 1…8).	LONG
ret_val	Wandlungsergebnis (0…65535). Bei 12 Bit und 14 Bit ADC werden die „fehlenden“ niederwertigsten Bits immer gleich 0 gesetzt.	LONG

Bemerkungen

Die Anweisung **ADCF** ist eine Zusammenstellung von aufeinander folgenden Funktionen:

Start_ConvF	→	Wait_EOCF	→	ReadADCF
A/D-Wandlung starten		Ende der Wandlung abwarten		Gewandelten Wert auslesen

Siehe auch

Start_ConvF, Wait_EOCF, ReadADCF, Read_ADCF4, Read_ADCF8, Read_ADCF4_Packed, Read_ADCF8_Packed

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim value As Long

Event:
  value = ADCF(1, 4)          'Misst einen Wert am analogen Eingg. 4
```

Burst_Abort

Burst_Abort bricht eine laufende Burst-Messreihe auf dem angegebenen Modul ab und gibt die Anzahl der bereits durchgeführten Messungen oder der gespeicherten Messwerte zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Burst_Abort(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Anzahl der gespeicherten Messwerte (0…1048575 = 220 - 1).	LONG

Bemerkungen

Wenn Sie die Burst-Messreihe abgebrochen haben, liefert Ihnen die Funktion **Burst_Status** die Anzahl der noch nicht durchgeführten Messungen zurück.

Bereits gespeicherte Messwerte können nach dem Abbruch der Burst-Messreihe mit dem Befehl **Burst_Read** vom Modul gelesen werden.

Siehe auch

Burst_Init, Burst_Read, Burst_Start, Burst_Status

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
Rem Messung im hochprioren Prozess, Auslesen im niederprioren
Rem Abschnitt Finish:. Abbruch der Messung bei
Rem einem Triggersignal am DIO32 Modul Nr.1
#Include ADwinPro_All.Inc

#Define samples 10000       'Anzahl durchzuführende Messungen
#Define ainadr 1            'Adresse AIn Modul
#Define dioadr 1            'Adresse DIO Modul
#Define sampleperiod 800    '50 kHz Messrate [=1/(25ns*800)]

Dim Data_1[samples] As Long
Dim Data_2[samples] As Long
Dim num_data As Long         'Anzahl der gewandelten Messwerte
Dim run_state As Long        'Ablaufstatus der Messreihe:
                             'steht/läuft/abgeschlossen
Dim cancel As Long           'Merker, ob Abbruch gewünscht

Init:
  Burst_Init(ainadr,1,sampleperiod,samples)
                             'Adresse, Modus, Messrate, Anzahl
                             'Messungen setzen
  run_state=0                'Status setzen: Messreihe steht
  DigProg1(dioadr,0)         'DIO 32: Bit 0…15 als Eingang

Event:
  If (run_state=0) Then      'wenn keine Messreihe läuft
    Burst_Start(ainadr)      'dann Messreihe starten
    run_state=1              'Messreihe läuft
  EndIf
  If (run_state=1) Then      'wenn Messreihe läuft
    num_data=Burst_Status(ainadr)'Anzahl restl. Messg. ermitteln
    cancel=Digin_Word1(dioadr) 'Abbruch von extern gewünscht?
    If (cancel And 1=1) Then 'Abbruchkriterium erfüllt
      num_data=Burst_Abort(ainadr)'Messreihe abbr./Anz. Messwerte
      End                    'Programmende
    EndIf
    If (num_data=0) Then End 'wenn alle Messungen fertig: beenden
  EndIf

Finish: 'Messwerte abholen im niederprioren Abschnitt
  Burst_Read(ainadr,1,1,num_data,Data_1,1)
```

Burst_CRead

Burst_CRead kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines bestimmten Kanals in ein Feld. Die Anzahl der zu kopierenden Messwerte muss angegeben werden.

Syntax

```
#Include ADwinPro_All.Inc
Burst_CRead(module,channel,count,array[],arr_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Kanal der übertragen werden soll (1…4; bei Pro-AIn-F-8/14 Rev. B: 1…8).	LONG
count	Anzahl der zu übertragenden Messwerte (1…n), dividiert durch 2.	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden; ein FIFO-Feld ist nicht erlaubt.	ARRAY LONG
arr_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1…n).	LONG

Bemerkungen

Die Messwerte werden jeweils in das untere Wort (Bits 15…0) eines Feldelements geschrieben (in das obere Wort eine Null).

Das Zielfeld muss mit mindestens arr_idx + count Elementen dimensioniert werden, um alle Messwerte aufnehmen zu können.

Lesen Sie mit **Burst_CRead** nur Messwerte einer kontinuierlichen Burst-Messreihe aus (siehe **Burst_CStart**).

Siehe auch

Burst_Init, Burst_CStart, Burst_Status, Set_Gain, Sync_Mode

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define count 10000
#Define module 1
#Define sampleperiod 20    '2000 kHz Messrate [=1/(25ns*20)]
Dim Data_1[count] As Long

Init:
  Rem Adresse, Modus, Messrate, Anzahl Messungen
  Burst_Init(module,1,sampleperiod,count)
  Rem kontinuierliche Burst-Messung starten
  Burst_CStart(module)
  Processdelay=80           'Mit 500 kHz den Trigger-Punkt finden

Event:
  Par_1=ReadADCF(module,1)  'Den aktuellsten Messwert holen
  If (Par_1>49152) Then     'Sind +5V überschritten?
    Par_9=Burst_Abort(module)  'Messung abbrechen und
    End                     'Prozess beenden (=FINISH ausführen)
  EndIf

Finish:
  Rem Die zuletzt gesammelten Daten in Data_1 kopieren
  Burst_CRead(module,1,count,Data_1,1)
```

Burst_CStart

Burst_CStart startet eine kontinuierliche Burst-Messreihe (Modus „Continuous“) auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
Burst_CStart(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Bemerkungen

Die Anweisung verwendet den moduleigenen Speicher als Ringspeicher. Während einer kontinuierlichen Burst-Messreihe werden in festen Zeitabständen Messungen durchgeführt und die Messwerte im Ringspeicher abgelegt.

Die gespeicherten Messwerte werden nach Abschluss der Messreihe mit **Burst_CRead** ausgelesen (nicht mit **Burst_Read**).

Mit dem Befehl ist es beispielsweise möglich, eine Anzahl von Messwerten direkt vor oder nach dem Eintreten einer bestimmten Trigger-Bedingung zu erfassen (und weiter zu verarbeiten).
Hierzu wird die Messung mit **Burst_Abort** sofort beim (oder um eine bestimmte Zeitspanne nach) Eintreten der Bedingung gestoppt.

Siehe auch

Burst_Init, Burst_CRead, Burst_Status, Set_Gain, Sync_Mode

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define count 10000
#Define module 1
#Define sampleperiod 20    '2000 kHz Messrate [=1/(25ns*20)]
Dim Data_1[count] As Long

Init:
  Burst_Init(module,1,sampleperiod,count) 'Adresse, Modus,
                          'Messrate, Anzahl Messungen
  Burst_CStart(module)    'kontin. Burst-Messreihe starten
  Processdelay=80         'Mit 500 kHz den Trigger-Punkt finden

Event:
  Par_1=ReadADCF(module,1) 'Den aktuellsten Messwert holen
  If (Par_1>49152) Then    'Sind +5V überschritten?
    Par_9=Burst_Abort(module) 'Messung abbrechen und
    End                     'Prozess beenden (=FINISH ausführen)
  EndIf

Finish:
  Rem Die zuletzt gesammelten Daten in Data_1 kopieren
  Burst_CRead(module,1,count,Data_1,1)
```

Burst_Init

Burst_Init legt die Kennwerte für eine Burst-Messreihe auf dem angegebenen Modul fest.

Die Kennwerte sind: Anzahl und Nummer der Messkanäle, Periodendauer der Messreihe und Anzahl der durchzuführenden Messungen.

Syntax

```
#Include ADwinPro_All.Inc
Burst_Init(module,mode,pulses,samples)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
mode	Messmodus (1…3; bei Pro-AIn-F-8/14 Rev. B: 1…4) legt die Anzahl der Messkanäle fest; die Kanalnummern werden automatisch festgelegt. Aus Messmodus und Speichergröße ergibt sich die max. Anzahl der Messwerte pro Kanal:	LONG

mode	Kanalnr.	max. Anzahl der Messwerte pro Kanal:
n	1…2^(n-1)	2^(21-n) - 1
1	1	220 - 1 = 1048575 = 0FFFFFH
2	1…2	219 - 1 = 524287 = 7FFFFH
3	1…4	218 - 1 = 262143 = 3FFFFH
4	1…8	217 - 1 = 131071 = 1FFFFH

pulses	legt die Periodendauer für eine Messreihe fest als Anzahl der Zeittakte: Periodendauer = pulses * 25ns (Mind.wert 20, entspricht 2MHz). Eine Periodendauer ist die Zeit vom Beginn einer Messung bis zum Beginn der nächsten Messung.	LONG
samples	Anzahl der durchzuführenden Messungen pro Kanal (der Maximalwert für samples wird durch mode bestimmt).	LONG

Bemerkungen

Auch wenn Sie nicht alle Messkanäle verwenden, werden bei jeder Messung einer Burst-Messreihe immer alle vorhandenen Kanäle gewandelt. Die Auswahl der Messkanäle mit **Burst_Init** legt nur fest, welche der gewandelten Messwerte gespeichert werden.

Sie können mit **ReadADCF** den aktuellen Messwert auch eines solchen Kanals einlesen, der nicht abgespeichert wird, z.B. zum Prüfen einer Trigger-Bedingung.

Sie können Burst-Messreihen auf mehreren Modulen synchron ausführen, wenn Sie die entsprechenden Module mit **Sync_Mode** zur Synchronisation frei geben. Dabei können alle Messungen der Messreihen zeitgleich (synchron) ausgeführt werden. Beachten Sie, dass die Anzahl der Burst-Messungen bei den verschiedenen Burst-Messreihen gleich sein sollte.

Die Befehle **ADCF** und **Start_ConvF** überschreiben eine mit **Burst_Init** vorgenommene Einstellung, d.h. sie setzen den Messmodus auf den Wert 1. Sie sollten die Befehle daher zur Sicherheit nicht zwischen **Burst_Init** und **Burst_Start** verwenden.

Siehe auch

Burst_Abort, Burst_Read, Burst_Read_Packed, Burst_Start, Burst_Status, ReadADCF, Set_Gain, Sync_Mode

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
Rem Messung im hochprioren Prozess; sobald eine Spannung größer
Rem 5V gemessen wird, Umschaltung in Burst-Modus und messen
Rem mit 1,0 MHz.
Rem Messwerte auslesen im niederprioren Abschnitt Finish:
#Include ADwinPro_All.Inc
#Define samples 10000       'Anzahl durchzuführende Messungen
#Define ainadr 1            'Adresse AIn Modul
#Define sampleperiod 40     '1,0 MHz Messrate [=1/(25ns*40)]
Dim Data_1[samples] As Long
Dim Data_2[samples] As Long
Dim dig_value As Long       'Anliegender Spannungswert
Dim remaining As Long       'Anzahl der restlichen Messwerte
Dim run_state As Long       'Ablaufstatus der Messreihe:
                            'steht/läuft/abgeschlossen
Init:
  run_state=0               'Status setzen: Messreihe steht

Event:
  If (run_state=0) Then     'keine Messreihe läuft?
    dig_value =ADCF(ainadr,1)
    If (dig_value>49151) Then 'Spannung >5 V
      'Adresse, Modus, Messrate, Anzahl Messungen
      Burst_Init(ainadr,2,sampleperiod,samples)
      Burst_Start(ainadr)   'dann Messreihe starten
      run_state=1           'Messreihe läuft
    EndIf
  EndIf
  If (run_state=1) Then     'Messreihe läuft?
    remaining=Burst_Status(ainadr)'Anzahl restliche Messungen
                            'ermitteln
    If (remaining=0) Then End'alle Messungen durchgeführt?
  EndIf

Finish: 'Daten auslesen im niederprioren Abschnitt Finish:
  Burst_Read(ainadr,1,1,samples,Data_1,1) 'Messwerte von Kanal 1
                            'auslesen
  Burst_Read(ainadr,2,1,samples,Data_2,1) 'Messwerte von Kanal 2
                            'auslesen
```

Burst_Read

Burst_Read kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines Kanals in ein bestimmtes Feld.

Die Anzahl der zu kopierenden Messwerte muss angegeben werden.

Syntax

```
#Include ADwinPro_All.Inc
Burst_Read(module, channel, startadr, count,
           array[], array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Kanal, der übertragen werden soll (1…4; bei Pro-AIn-F-8/14 Rev. B: 1…8).	LONG
startadr	Quell-Startadresse: Adresse, ab der die Messwerte gelesen werden.	LONG
count	Anzahl der zu übertragenden Messwerte (1…n).	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden; ein FIFO-Feld ist nicht erlaubt.	ARRAY LONG
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1…n).	LONG

Bemerkungen

Die Messwerte werden jeweils in das untere Wort (Bits 15…0) eines Feldelements geschrieben (in das obere Wort eine Null).

Das Zielfeld muss mit mindestens array_idx + count -1 Elementen dimensioniert werden, um alle Messwerte aufnehmen zu können.

Wenn Sie den Befehl **Burst_Read** während einer laufenden Burst-Messreihe ausführen, kann es zu Fehlfunktionen des Moduls kommen. Stellen Sie deshalb zuerst sicher, dass die Burst-Messreihe abgeschlossen ist. Hierzu gibt es 2 Möglichkeiten:

- Sie prüfen den Ablaufstatus der Messreihe mit dem Befehl **Burst_Status**. Der Rückgabewert 0 (Null) zeigt an, dass die Messreihe beendet ist (anderenfalls müssen Sie auf das Ende der Messreihe warten).
- Sie brechen die Messreihe mit **Burst_Abort** ab.

In einem hoch priorisierten Prozess dürfen Sie mit **Burst_Read** nur so viele Daten auf einmal vom Modul lesen, dass die Auslastung des *ADwin*-Systems nicht über 100% steigt. Falls dies dennoch geschieht, kann die Kommunikation mit dem PC in einen undefinierten Zustand geraten (Time-out). Diese Einschränkung besteht nicht im Abschnitt LowInit: und bei niederprioren Prozessen.

Für einen sicheren Betrieb empfehlen wir, dass Sie den Befehl **Burst_Read** nur in einem niedrig priorisierten Prozess oder in einem niedrig priorisierten Programmabschnitt (z.B. **Finish:**) verwenden.

Siehe auch

Burst_Abort, Burst_Init, Burst_Read_Packed, Burst_Start, Burst_Status, Set_Gain, Sync_Mode

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
Rem Achtung: Messung im hochprioren Prozess, Auslesen im
Rem niederprioren Abschnitt!
Rem Startet bei positiver Flanke an DI0 Bit 0 eine Analogmessung
Rem an Kanal 1
#Include ADwinPro_All.Inc
#Define samples 10000       'Anzahl durchzuführende Messungen
#Define sampleperiod 30     '1,33 MHz Messrate [=1/(25ns*30)]
#Define dioadr 1            'Moduladresse DIO Modul
#Define ainadr 1            'Moduladresse AIN Modul
#Define run_state Par_1     'Ablaufstatus der Messreihe:
                            'steht/läuft/abgeschlossen
Dim Data_1[samples] As Long
Dim remaining As Long       'Anzahl der restlichen Messwerte
Dim i As Long
Dim trigger As Long         'Merker für externes Trigger-Signal

Init:
  DigProg1(dioadr,0)        'DIO Bit 0…15 als Input
  run_state=0               'Status setzen: Messreihe steht
  Burst_Init(ainadr,1,sampleperiod,samples)
                            'Adresse,Modus,Messrate,Anz. Messng.

Event:
  If (run_state=0) Then     'wenn keine Messreihe läuft
    trigger=Digin_Word1(dioadr)'Trigger-Signal lesen
    If (trigger And 1=1) Then'Trigger vorhanden?
      Burst_Start(ainadr)   'dann Messreihe starten
      run_state=1           'Messreihe läuft
    EndIf
  EndIf
  If (run_state=1) Then     'wenn Messreihe läuft
    remaining=Burst_Status(ainadr)'Anzahl restliche Messungen
    If (remaining=0) Then End 'alle Messungen durchgeführt?
                            'dann Ende
  EndIf

Finish: 'Daten auslesen im niederprioren Abschnitt Finish:
  Burst_Read(ainadr,1,1,samples,Data_1,1)
                            'Moduladr., Kanal, Startadr., Länge,
                            'Zielfeld, Startindex im Zielfeld
```

Burst_Read_Packed

Burst_Read_Packed kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines Kanals in ein bestimmtes Feld, jedoch komprimiert und schnell.

Die Anzahl der zu kopierenden Messwerte muss angegeben werden.

Syntax

```
#Include ADwinPro_All.Inc

Burst_Read_Packed(module, channel, startadr, count,
                  array[], array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Kanal der übertragen werden soll (1…4; bei Pro-AIn-F-8/14 Rev. B: 1…8).	LONG
startadr	Quell-Startadresse: Adresse, ab der die Messwerte gelesen werden.	LONG
count	Anzahl der zu übertragenden Messwerte, dividiert durch 2 (1…n/2).	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden; ein FIFO-Feld ist nicht erlaubt.	ARRAY LONG
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1…n).	LONG

Bemerkungen

Die Messwerte werden abwechselnd in das untere und das obere Wort eines Feldelements geschrieben (siehe Tabelle). Das Zielfeld muss mit mindestens array_idx + count Elementen dimensioniert werden, um alle Messwerte aufnehmen zu können.

Adresse im Ziel-Feld array[]	oberes Wort (Bit 31…16)	unteres Wort (Bit 15…0)
array_idx	Messwert 2	Messwert 1
array_idx + 1	Messwert 4	Messwert 3
…	…	…
array_idx + count	Messwert n	Messwert (n-1)

Wenn eine ungerade Anzahl von Messwerten aus dem Speicher kopiert wird, ist der letzte reguläre Messwert im unteren Wort des letzten Feld-Elements zu finden, sowie ein nicht definierter Wert im oberen Wort.

Wenn Sie den Befehl **Burst_Read_Packed** während einer laufenden Burst-Messreihe ausführen, kann es zu Fehlfunktionen des Moduls kommen. Stellen Sie deshalb zuerst sicher, dass die Burst-Messreihe abgeschlossen ist. Hierzu gibt es 2 Möglichkeiten:

- Sie prüfen den Ablaufstatus der Messreihe mit dem Befehl **Burst_Status**. Der Rückgabewert 0 (Null) zeigt an, dass die Messreihe beendet ist (anderenfalls müssen Sie auf das Ende der Messreihe warten).
- Sie brechen die Messreihe mit **Burst_Abort** ab.

In einem hoch priorisierten Prozess dürfen Sie mit **Burst_Read_Packed** nur so viele Daten auf einmal vom Modul lesen, dass die Auslastung des *ADwin*-Systems nicht über 100% steigt. Falls dies dennoch geschieht, kann die Kommunikation mit dem PC in einen undefinierten Zustand geraten (Time-out). Diese Einschränkung besteht nicht im Abschnitt **LowInit:** und bei niederprioren Prozessen.

Für einen sicheren Betrieb wird deshalb empfohlen, dass Sie den Befehl **Burst_Read_Packed** nur in einem niedrig priorisierten Prozess oder in einem niedrig priorisierten Programmabschnitt (z.B. **Finish:**) verwenden.

Siehe auch

Burst_Abort, Burst_Init, Burst_Read, Burst_Start, Burst_Status, Set_Gain, Sync_Mode

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
Rem Achtung: Messung im hochprioren Prozess, Auslesen im
Rem niederprioren Abschnitt!
#Include ADwinPro_All.Inc
#Define samples 10000       'Anzahl durchzuführende Messungen
#Define ainadr 2            'Adresse AIN Modul
#Define sampleperiod 40     '1 MHz Messrate [=1/(25ns*40)]
Dim Data_1[samples] As Long
Dim Data_2[samples] As Long
Dim remaining As Long        'Anzahl der restlichen Messwerte
Dim run_state As Long        'Ablaufstatus der Messreihe:
                             'steht/läuft/abgeschlossen

Init:
  Burst_Init(ainadr,2,sampleperiod,samples)
                             ''Adresse,Modus,Messrate,Anz. Messng.
  run_state=0                'Messreihe läuft nicht
  Set_Gain(ainadr,1,1)       'Messbereich Kanal 1 +-5V
  Set_Gain(ainadr,2,2)       'Messbereich Kanal 2 +-2.5 V

Event:
  If (run_state=0) Then      'wenn keine Messreihe läuft
    Burst_Start(ainadr)      'dann Messreihe starten
    run_state=1              'Messreihe läuft
  EndIf
  If (run_state=1) Then      'wenn Messreihe läuft
    remaining=Burst_Status(ainadr) 'Anzahl restliche Messungen
    If (remaining=0) Then run_state=2 'Messreihe abgeschlossen,
                             'Merker setzen
  EndIf

Finish: 'Daten auslesen im niederprioren Abschnitt Finish:
  Burst_Read_Packed(ainadr,1,1,samples,Data_1,1) 'Messwerte von
                             'Kanal 1 lesen
  Burst_Read_Packed(ainadr,2,1,samples,Data_2,1) 'Messwerte von
                             'Kanal 2 lesen
```

Burst_Start

Burst_Start startet (unabhängig vom Prozessor) eine Burst-Messreihe auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
Burst_Start(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Bemerkungen

Burst-Messreihen können synchronisiert werden. Hierfür gibt es 2 Möglichkeiten:

1. Burst-Messreihe synchron zu anderen Messungen starten:

   Das Modul muss mit **SyncEnable** zur Synchronisierung frei gegeben sein. Die Burst-Messreihe wird dann mit dem Befehl **SyncAll** synchron zu anderen Messungen gestartet.

   Dieser Modus synchronisiert nur den Start, nicht die weitere Ausführung; sie erlaubt aber auch die Synchronisierung mit Einzelmessungen.

2. Die Messungen mehrerer Burst-Messreihen synchron ausführen:

   Das Modul muss mit **Sync_Mode** zur Synchronisierung (Modus 2 und 3) und mit **Burst_Start** zum Start freigegeben sein. Die Wandlungen der Messreihe werden durch Synchron-Signale ausgelöst.

   Geben Sie bei Master- / Slave-Modus (siehe **Sync_Mode**) den Start der Burst-Messungen auf den Slave-Modulen zuerst frei und auf dem Master-Modul zuletzt.

   Geben Sie bei Event-gesteuerten Modulen (siehe **Sync_Mode**) die gewünschten Event-Eingänge erst dann mit **EventEnable** frei, wenn Sie alle Burst-Messreihen mit **Burst_Start** zum Start freigegeben haben.

Siehe auch

Burst_Abort, Burst_Init, Burst_Read, Burst_Read_Packed, Burst_Status, Set_Gain, SyncEnable, Sync_Mode

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
Rem Messung im hochprioren Prozess; sobald eine Spannung größer
Rem 5 V gemessen wird, Umschaltung in Burst-Modus und messen mit
Rem 1,0 MHz.
Rem Messwerte auslesen im niederprioren Abschnitt Finish:
#Include ADwinPro_All.Inc
#Define samples 10000       'Anzahl durchzuführende Messungen
#Define ainadr 1            'Adresse AIn Modul
#Define sampleperiod 40     '1 MHz Messrate [=1/(25ns*40)]
Dim Data_1[samples] As Long
Dim Data_2[samples] As Long
Dim dig_value As Long        'Anliegender Spannungswert
Dim remaining As Long        'Anzahl der restlichen Messwerte
Dim run_state As Long        'Ablaufstatus der Messreihe:
                             'steht/läuft/abgeschlossen
Init:
  run_state=0                'Status setzen: Messreihe steht

Event:
  If (run_state = 0) Then    'wenn keine Messreihe läuft
    dig_value = ADCF(ainadr,1)
    If (dig_value > 49151) Then 'Spannung >5 V
      Burst_Init(ainadr,2,sampleperiod,samples) 'Adresse, Modus,
                             'Messrate, Anzahl Messungen
      Burst_Start(ainadr)    'dann Messreihe starten
      run_state=1            'Messreihe läuft
    EndIf
  Else                       'wenn Messreihe läuft
    remaining = Burst_Status(ainadr)'Anzahl restliche Messungen
                             'ermitteln
    If (remaining = 0) Then End 'alle Messungen durchgeführt
  EndIf

Finish: 'Daten auslesen im niederprioren Abschnitt Finish:
  Burst_Read(ainadr,1,1,samples,Data_1,1) 'Messwerte von
                             'Kanal 1 auslesen
  Burst_Read(ainadr,2,1,samples,Data_2,1) 'Messwerte von
                             'Kanal 2 auslesen
```

Burst_Status

Burst_Status ermittelt die Anzahl der noch durchzuführenden Burst-Messungen auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Burst_Status(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Anzahl der noch auszuführenden Messungen.	LONG

Bemerkungen

Wenn eine Messreihe bereits abgeschlossen ist, liefert die Funktion den Rückgabewert 0 (Null).

Siehe auch

Burst_Abort, Burst_Init, Burst_CStart, Burst_CRead, Burst_Read, Burst_Read_Packed, Burst_Start, Set_Gain, Sync_Mode

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
Rem Messung im hochprioren Prozess; sobald eine Spannung größer
Rem 5 V gemessen wird, Umschaltung in Burst-Modus und messen mit
Rem 1,0 MHz
Rem Messwerte auslesen im niederprioren Abschnitt Finish:
#Include ADwinPro_All.Inc
#Define samples 10000'Anzahl der durchzuführenden Messungen
#Define ainadr 1'Adresse AIn Modul
#Define sampleperiod 40'1 MHz Messrate [=1/(25ns*40)]

Dim Data_1[samples] As Long
Dim Data_2[samples] As Long
Dim volt_value As Long      'Anliegender Spannungswert
Dim remaining As Long       'Anzahl der restlichen Messwerte
Dim run_state As Long       'Ablaufstatus der Messreihe:
                            'steht/läuft/abgeschlossen

Init:
  run_state=0               'Status setzen: Messreihe steht

Event:
  If (run_state=0) Then     'wenn keine Messreihe läuft
    volt_value =ADCF(ainadr,1)
    If (volt_value>49151) Then 'Spannung >5 V
      Burst_Init(ainadr,2,sampleperiod,samples) 'Adresse, Modus,
                            'Messrate, Anzahl Messungen
      Burst_Start(ainadr)   'dann Messreihe starten
      run_state=1           'Messreihe läuft
    EndIf
  EndIf
  If (run_state=1) Then     'wenn Messreihe läuft
    remaining=Burst_Status(ainadr)'Anzahl restliche Messungen
                            'ermitteln
    If (remaining=0) Then End'alle Messungen durchgeführt
  EndIf

Finish: 'Daten auslesen im niederprioren Abschnitt Finish:
  Burst_Read(ainadr,1,1,samples,Data_1,1) 'Messwerte von
                            'Kanal 1 auslesen
  Burst_Read(ainadr,2,1,samples,Data_2,1) 'Messwerte von
                            'Kanal 2 auslesen
```

ReadADC

ReadADC liest das Ergebnis einer Wandlung aus dem ADC-Register des angegebenen Moduls aus.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ReadADC(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Im ADC-Register enthaltener Messwert (0…65535).	LONG

Bemerkungen

Bei dem Modul Pro-AIn-8/16 wird nicht der aktuelle, sondern der Messwert aus der vorhergehenden Messung ausgelesen (vgl. Anweisung **ADC16**).

Wenn die Anweisung zu früh, d.h. während einer Wandlung ausgeführt wird, hat der Rückgabewert eine entsprechend geringere Auflösung.

Siehe auch

ADC, ADC16, ReadADC_SConv, Set_Mux, Start_Conv, SyncAll, Wait_EOC

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Deklaration

Event:
  Set_Mux(1,0)              'Multiplexer auf Eingang 1 setzen
  Rem 3µs (12-Bit-ADC) bzw. 14µs (AIn-8/16 Rev.B, AIn-32/16 Rev.B)
  Rem auf das Einschwingen des Multiplexers warten*
  Start_Conv(1)             'Start AD-Wandlung
  Wait_EOC(1)               'Warten auf Wandlung-Ende
  value1 = ReadADC(1)       'Wert vom ADC einlesen
```

*Die Wartezeit kann z.B. durch eine Reihe von *ADbasic*-Befehlen überbrückt werden, die keine Messung auf dem selben A/D-Modul durchführen, auf dem der Multiplexer umgestellt wurde.

Besteht keine Notwendigkeit, die Wartezeit mit Befehlen zu nutzen, so kann die folgende Warteschleife programmiert werden, welche jedoch nur in hoch priorisierten Prozessen angewendet werden darf (mit T9 und T10; T11 siehe Kapitel 5.2.4):

```
  Start_Conv(1)             'Start AD-Wandlung
  Par_80 = Read_Timer()
  Do
  Until (Read_Timer() - Par_80 > 560) '25ns*560=14µs warten
```

ReadADC_SConv

ReadADC_SConv liest das Wandlungsergebnis aus dem ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ReadADC_SConv(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Im ADC-Register enthaltener Messwert (0…65535).	LONG

Bemerkungen

Bei dem Modul Pro-AIn-8/16 wird nicht der aktuelle, sondern der Messwert aus der vorhergehenden Messung ausgelesen (vgl. Anweisung **ADC16**).

Wenn die Anweisung zu früh, d.h. während einer Wandlung ausgeführt wird, hat der Rückgabewert eine entsprechend geringere Auflösung.

Bei Verwendung der Anweisung **SyncAll** kann **ReadADC_SConv** nicht mehr angewendet werden! Setzen Sie statt dessen nur den Befehl **ReadADC** ein, da der Start der Wandlung durch den folgenden Befehl **SyncAll** eingeleitet wird.

Siehe auch

ADC, ADC16, ReadADC, SE_Diff, Set_Mux, Start_Conv, Wait_EOC

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000] As Long   'Deklaration

Init:
  i = 1
  Set_Mux(1, 2)            'Multiplexer auf Eingang 3 setzen
  Rem 3µs (12-Bit ADC) bzw.
  Rem 14µs (AIn-8/16 Rev. B,Rem AIn-32/16 Rev. B) auf das
  Rem Einschwingen des Multiplexers warten
  Start_Conv(1)            'A/D-Wandler starten

Event:
  Wait_EOC(1)              'Wandlungsende abwarten
  Data_1[i] = ReadADC_SConv(1) 'A/D-Wandler auslesen und starten
  Inc(i)                   'Index erhöhen
  If (i = 1001) Then End   'Nach 1000 Messwerten Prozess beenden
```

ReadADCF

ReadADCF liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ReadADCF(module,adc_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
adc_no	Nummer des zu lesenden ADC (1…4 oder 1…8).	LONG
ret_val	Im F-ADC-Register enthaltener Messwert (0…65535).	LONG

Bemerkungen

Bei einem 12 Bit-Wandler sind die 4 niederwertigsten Bits immer 0, bei einem 14 Bit-Wandler die 2 niederwertigsten Bits.

Alternativ stehen die Befehle **Read_ADCF4**, **Read_ADCF8**, **Read_ADCF4_Packed**, **Read_ADCF8_Packed** zur Verfügung, die mehrere Ergebnisse sehr schnell auslesen.

Siehe auch

ADCF, Start_ConvF, Wait_EOCF, ReadADCF_32, ReadADCF_SConv, ReadADCF_SConv_32, Read_ADCF4, Read_ADCF8, Read_ADCF4_Packed, Read_ADCF8_Packed

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Deklaration

Event:
  Start_ConvF(1,1)          'Start AD-Wandlung
  Wait_EOCF(1,1)            'Warten auf Wandlung-Ende
  value1 = ReadADCF(1,1)    'Wert vom ADC einlesen
```

Read_ADCF4

Read_ADCF4 liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus.

Syntax

```
#Include ADWINPRO_ALL.Inc
Read_ADCF4(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
array[]	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY
		LONG
index	Index des Elements im Zielfeld, in dem der erste Messwert gespeichert wird.	LONG

Bemerkungen

Es werden immer die Messwerte der F-ADC 1…4 des Moduls gelesen. Die Messwerte werden nacheinander im Zielfeld array[] gespeichert, beginnend mit dem Element index.

Die Anweisung ist wesentlich schneller als das mehrfache Auslesen mit **ReadADCF**.

Bei einem 12 Bit-Wandler sind die 4 niederwertigsten Bits eines Messwerts immer 0, bei einem 14 Bit-Wandler die 2 niederwertigsten Bits.

Siehe auch

ADCF, Start_ConvF, ReadADCF, Read_ADCF8, Read_ADCF4_Packed, Read_ADCF8_Packed, ReadADCF_SConv, Wait_EOCF

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADWINPRO_ALL.Inc
Dim value[4] As Long        'Feld für Messwerte

Init:
  Start_ConvF(1,0Fh)        'Start AD-Wandlung Kanäle 1…4

Event:
  Wait_EOCF(1,0Fh)          'Warten auf Wandlungsende
  Read_ADCF4(1,value,1)     'Werte der ADC 1…4 lesen
  Start_ConvF(1,0Fh)        'Neue AD-Wandlung starten
```

Read_ADCF8

Read_ADCF8 liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus.

Syntax

```
#Include ADWINPRO_ALL.Inc
Read_ADCF8(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
array[]	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY LONG
index	Index des Elements im Zielfeld, in dem der erste Messwert gespeichert wird.	LONG

Bemerkungen

Es werden immer die Messwerte der F-ADC 1…8 des Moduls gelesen. Die Messwerte werden nacheinander im Zielfeld array[] gespeichert, beginnend mit dem Element index.

Die Anweisung ist wesentlich schneller als das mehrfache Auslesen mit **ReadADCF**.

Bei einem 12 Bit-Wandler sind die 4 niederwertigsten Bits eines Messwerts immer 0, bei einem 14 Bit-Wandler die 2 niederwertigsten Bits.

Siehe auch

ADCF, Start_ConvF, Wait_EOCF, Read_ADCF4, Read_ADCF4_Packed, Read_ADCF8_Packed, ReadADCF_SConv

Gültig für

AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A

Beispiel

```
#Include ADWINPRO_ALL.Inc
Dim value[8] As Long        'Feld für Messwerte

Init:
  Start_ConvF(1,0FFh)       'Start AD-Wandlung Kanäle 1…8

Event:
  Wait_EOCF(1,0FFh)         'Warten auf Wandlungsende
  Read_ADCF8(1,value,1)     'Werte der ADC 1…8 lesen
  Start_ConvF(1,0FFh)       'Neue AD-Wandlung starten
```

Read_ADCF4_Packed

Read_ADCF4_Packed liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus. Je 2 Messwerte werden gemeinsam in einem 32 Bit-Wert zurückgegeben.

Syntax

```
#Include ADWINPRO_ALL.Inc
Read_ADCF4_Packed(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
array[]	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY
		LONG
index	Index des Elements im Zielfeld, in dem der erste Messwert gespeichert wird.	LONG

Bemerkungen

Es werden immer die Messwerte der F-ADC 1…4 des Moduls gelesen. Der Messwert des F-ADC mit ungerader Nummer wird in das untere Wort geschrieben, der mit gerader Nummer in das obere Wort. Die Werte werden auf folgende Weise im Zielfeld array[] gespeichert:

Feldelement Nr.	Bitnr.	
	31…16	15…0
index	F-ADC 2	F-ADC 1
index+1	F-ADC 4	F-ADC 3

Die Anweisung ist schneller als das Auslesen mit **Read_ADCF4**.

Bei einem 12 Bit-Wandler sind die 4 niederwertigsten Bits eines Messwerts immer 0, bei einem 14 Bit-Wandler die 2 niederwertigsten Bits.

Siehe auch

ADCF, Start_ConvF, Wait_EOCF, Read_ADCF4, Read_ADCF8, Read_ADCF8_Packed, ReadADCF_SConv

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADWINPRO_ALL.Inc
Dim value[2] As Long        'Feld für Messwerte

Init:
  Start_ConvF(1,0Fh)        'Start AD-Wandlung Kanäle 1…4

Event:
  Wait_EOCF(1,0Fh)          'Warten auf Wandlungsende
  Read_ADCF4_Packed(1,value,1)  'Werte der ADC 1…4 lesen
  Start_ConvF(1,0Fh)        'Neue AD-Wandlung starten
```

Read_ADCF8_Packed

Read_ADCF8_Packed liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus. Je 2 Messwerte werden gemeinsam in einem 32 Bit-Wert zurückgegeben.

Syntax

```
#Include ADWINPRO_ALL.Inc
Read_ADCF8_Packed(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
array[]	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY
		LONG
index	Index des Elements im Zielfeld, in dem der erste Messwert gespeichert wird.	LONG

Bemerkungen

Es werden immer die Messwerte der F-ADC 1…8 des Moduls gelesen. Der Messwert des F-ADC mit ungerader Nummer wird in das untere Wort geschrieben, der mit gerader Nummer in das obere Wort. Die Werte werden auf folgende Weise im Zielfeld array[] gespeichert:

Feldelement Nr.	Bitnr.	
	31…16	15…0
index	F-ADC 2	F-ADC 1
index+1	F-ADC 4	F-ADC 3
index+2	F-ADC 6	F-ADC 5
index+3	F-ADC 8	F-ADC7

Die Anweisung ist schneller als das Auslesen mit **Read_ADCF8**.

Bei einem 12 Bit-Wandler sind die 4 niederwertigsten Bits eines Messwerts immer 0, bei einem 14 Bit-Wandler die 2 niederwertigsten Bits.

Siehe auch

ADCF, Start_ConvF, Wait_EOCF, Read_ADCF4, Read_ADCF8, Read_ADCF4_Packed, ReadADCF_SConv

Gültig für

AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A

Beispiel

```
#Include ADWINPRO_ALL.Inc
Dim value[4] As Long        'Feld für Messwerte

Init:
  Start_ConvF(1,0FFh)       'Start AD-Wandlung Kanäle 1…8

Event:
  Wait_EOCF(1,0FFh)         'Warten auf Wandlungsende
  Read_ADCF8_Packed(1,value,1)  'Werte der ADC 1…8 lesen
  Start_ConvF(1,0FFh)       'Neue AD-Wandlung starten
```

ReadADCF_32

ReadADCF_32 liest die Wandlungsergebnisse aus 2 aufeinander folgenden F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ReadADCF_32(module,adc_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
adc_no	Nummer des ersten zu lesenden F-ADC (1, 3 oder 1, 3, 5, 7).	LONG
ret_val	Die in dem F-ADC-Registern enthaltenen Messwerte (jeweils 0…65535); je ein Messwert ist im unteren und im oberen Wort.	LONG

Bemerkungen

Das Wandlungsergebnis des ADC mit der Nummer adc_no wird in das untere Wort geschrieben, das mit der Nummer adc_no+1 in das obere Wort.

Bei einem 12 Bit-Wandler sind die 4 niederwertigsten Bits immer 0, bei einem 14 Bit-Wandler die 2 niederwertigsten Bits.

Die Nummer des ersten F-ADC muss ungerade sein. Es ist also beispielsweise nicht möglich, die Wandlungsergebnisse der F-ADC 2 und 3 mit einem Befehl auszulesen.

Siehe auch

ADCF, ReadADCF, Read_ADCF4, Read_ADCF8, Read_ADCF4_Packed, Read_ADCF8_Packed, ReadADCF_SConv, ReadADCF_SConv_32, Start_ConvF, Wait_EOCF

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Deklaration

Event:
  Start_ConvF(1,3)          'Start AD-Wandlung auf ADC1 und ADC2
  Wait_EOCF(1,3)            'Warten auf das Ende der Wandlungen
  value1 = ReadADCF_32(1,1) 'Wert von ADC1 und ADC2 einlesen
```

ReadADCF_ SConv

ReadADCF_SConv liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ReadADCF_SConv(module,adc_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
adc_no	Nummer des zu lesenden ADC (1…4 oder 1…8).	LONG
ret_val	Im F-ADC-Register enthaltener Messwert (0…65535).	LONG

Bemerkungen

Bei einem 12 Bit-Wandler sind die 4 niederwertigsten Bits immer 0, bei einem 14 Bit-Wandler die 2 niederwertigsten Bits.

Siehe auch

ADCF, ReadADCF, Read_ADCF4, Read_ADCF8, Read_ADCF4_Packed, Read_ADCF8_Packed, ReadADCF_32, ReadADCF_SConv_32, Start_ConvF, Wait_EOCF

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000] As Long   'Deklaration

Init:
  i=1
  Start_ConvF(1,1)          'A/D-Wandler starten

Event:
  Wait_EOCF(1,1)
  Data_1[i] = ReadADCF_SConv(1,1)'A/D-Wandler auslesen + starten
  Inc(i)                    'Index erhöhen
  If (i=1001) Then End      'Nach 1000 Messwerten Prozess beenden
```

ReadADCF_SConv_32

ReadADCF_SConv_32 liest die Wandlungsergebnisse aus 2 F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.

Anschließend wird sofort eine neue Konvertierung gestartet.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = ReadADCF_SConv_32(module,adc_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
adc_no	Nummer des ersten zu lesenden F-ADC (1…2 oder 1…4).	LONG
ret_val	Der Rückgabewert (32 Bit) enthält die Messdaten von 2 aufeinanderfolgenden F-ADC (je 16 Bit: 0…65535); je ein Messwert ist im unteren und im oberen Wort.	LONG

adc_no	1	2	3	4
F-ADC-Nr.	1, 2	3, 4	5, 6	7, 8

Bemerkungen

Bei einem 12 Bit-Wandler sind die 4 niederwertigsten Bits immer 0, bei einem 14 Bit-Wandler die 2 niederwertigsten Bits.

Siehe auch

ADCF, ReadADCF, Read_ADCF4, Read_ADCF8, Read_ADCF4_Packed, Read_ADCF8_Packed, ReadADCF_32, ReadADCF_SConv, Start_ConvF, Wait_EOCF

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim value As Long           'Deklaration

Init:
  Start_ConvF(1,3)          'Start AD-Wandlung

Event:
  Wait_EOCF(1,3)            'Warten auf das Ende der Konvertierung
  value = ReadADCF_SConv_32(1,1)'Wert vom ADC1 und ADC2
                            'einlesen und die Wandlung
                            'beider ADC neu starten
```

SE_Diff

SE_Diff stellt die Betriebsart single ended oder differentiell für alle Analog-Eingänge auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.Inc
SE_Diff(module,choice)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
choice	Betriebsart der Analog-Eingänge. 0: single ended. 1: differentiell (Default).	LONG

Bemerkungen

In der Betriebsart single ended stehen Ihnen 32 Eingänge zur Verfügung, in der Betriebsart differentiell 16 Eingänge. Nach dem Einschalten des Systems befinden sich alle Eingänge im differentiellen Modus.

Achten Sie auf die unterschiedliche Pin-Belegung bei den entsprechenden Konfigurationen (siehe Hardware-Dokumentation des Moduls).
Im differentiellen Betrieb werden die analogen Eingänge nur mit den Nummern 1…8 und 17…24 angesprochen.

Siehe auch

ADC

Gültig für

AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  SE_Diff(1,0)              'Modul mit der Adresse 1 wird
                            'auf SE gesetzt
  SE_Diff(2,1)              'Modul mit der Adresse 2 wird
                            'auf DIFF gesetzt
```

Set_Gain

Set_Gain setzt für einen Kanal des angegebenen Moduls die Betriebsart fest und damit auch den Verstärkungsfaktor und den Messbereich.

Syntax

```
#Include ADwinPro_All.Inc
Set_Gain(module,channel,mode)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Kanal, dessen Verstärkung eingestellt werden soll (1…4 oder 1…8).	LONG
mode	Betriebsart (0…3) des Kanals: Legt die Verstärkung des Eingangssignals fest. Mit der Verstärkung ändert sich der Messbereich für die Eingangssignale umgekehrt proportional.	LONG

Betriebsart	Verstärkung	Messbereich
mode	2^n	±10V / 2^n
0	1	±10 V
1	2	±5 V
2	4	±2,5 V
3	8	±1,25 V

Bemerkungen

- / -

Siehe auch

ADCF, Burst_CStart, Burst_Start, ReadADCF, Start_ConvF, Wait_EOCF

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define ainadr 1'Moduladresse AIN Modul

Init:
  Set_Gain(ainadr,4,1)      'Spannungsbereich im Kanal 4 auf
                            'Betriebsart 1 stellen
                            '(Messbereich: +5V…-5V)

Event:
  Par_1 = ADCF(1,4)         'Wert vom analogen Eingang 4 messen
```

Set_Mux

`Set_Mux` stellt den Multiplexer des angegebenen Moduls auf einen bestimmten Eingang und eine bestimmte Verstärkung ein.

Syntax

```
#Include ADwinPro_All.Inc
Set_Mux(module,pattern)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`pattern`	Bitmuster zur Einstellung des Multiplexers (siehe Tabelle); 2 Bits stellen die Verstärkung ein, 5 Bits die Nummer des Eingangs.	LONG

Bitnr.							
31:7	6	5	4	3	2	1	0
ohne Funktion	Verstär-kung		Multiplexer-Eingang				
	1 = `00b`		Eingang 1: `00000b`				
	2 = `01b`		Eingang 2: `00001b`				
	4 = `10b`		Eingang 3: `00010b`				
	8 = `11b`		…				
			Eingang 32: `11111b`				

Bemerkungen

Kombinieren Sie für die gewünschte Multiplexer-Einstellung die passenden Bitkombinationen für Verstärkung und Multiplexer-Eingang. Sie können die Bits im Parameter `pattern` im angegebenen Binärformat verwenden oder sie in Hexadezimal- oder Dezimal-Format umrechnen. Beachten Sie die angehängten Buchstaben `h` und `b` für Hex- und Binär-Code.

Bitte beachten Sie die erforderliche Einschwingzeit des Multiplexers (3µs für 12 Bit-ADC, 14µs für AIn-8/16 Rev. B, AIn-32/16 Rev. B). Stellen Sie sicher, dass zwischen der Neueinstellung des Multiplexers und dem Konvertierungsbeginn mindestens diese Zeit vergeht.

Beim Modul Pro-AIn-8/16 kann keine Verstärkung eingestellt werden. Die Bits 5 und 6 haben hier keine Funktion.

Siehe auch

ADC, ADC16, Start_Conv, Wait_EOC, ReadADC

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Deklaration

Event:
  Set_Mux(1,0100010b)       'MUX auf Eing. 3, Verstärkung 2 setzen
                            '3µs (12-Bit ADC) bzw. 14µs
                            '(AIn-8/16 Rev.B,AIn-32/16 Rev.B) auf
                            'das Einschwingen des Multiplexers
                            'warten
  Start_Conv(1)             'Start AD-Wandlung
  Wait_EOC(1)               'Warten auf Wandlung-Ende
  value1 = ReadADC(1)       'Wert vom ADC einlesen
```

Seq_Mode

Seq_Mode initialisiert das angegebene Modul für den Betrieb mit Ablaufsteuerung.

Es werden der Arbeitsmodus der Ablaufsteuerung und der Verstärkungsfaktor eingestellt (für alle Kanäle gleich).

Syntax

```
#Include ADwinPro_All.Inc
Seq_Mode(module, mode, gain)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
mode	Arbeitsmodus der Ablaufsteuerung auf dem Modul: 0 Normaler Modus (Default). 1 Modus „single shot“. 3 Modus „continuous“.	LONG
gain	Verstärkungsfaktor (nur für die Modi 1 und 3): 0 Faktor = 1. 1 Faktor = 2. 2 Faktor = 4. 3 Faktor = 8.	LONG

Bemerkungen

Nach dem Einschalten ist der Modus 0 aktiv.

Die Modi 1 und 3 aktivieren die Ablaufsteuerung des Moduls. Diese ermöglicht, mit einem einzigen Befehl an mehreren Kanälen nacheinander eine Wandlung durchzuführen. Die Steuerung bezieht sich immer nur auf die mit **Seq_Select** definierte Auswahl an Kanälen, die Messgruppe.

Die Modi unterscheiden sich wie folgt:

Modus	Art der Messung
0 Normal:	Standard: Einzelne Messung an einem Kanal (siehe **ADC**).
1 „single shot“:	Die Ablaufsteuerung startet mit **Start_Conv** und endet, sobald die gewählten Kanäle je einmal gewandelt sind. Das Ende der Ablaufsteuerung kann mit **Seq_Status** abgefragt werden, die Messwerte der Ablaufsteuerung werden mit **Seq_Read** eingelesen.
3 „continuous“:	Die Ablaufsteuerung wandelt ununterbrochen neue Messwerte an den gewählten Kanälen, d.h. Wandlung und Prozesszyklus laufen asynchron. Die Wandlung wird mit **Seq_Start** gestartet. Mit **Seq_Read** werden die jeweils neuesten Messwerte gelesen.

Es können bis zu 32 Messwerte im moduleigenen Speicher abgelegt werden. Bei 14 Bit-Wandlern wird das Messergebnis linksbündig in einem 16 Bit-Wert zurückgegeben, d.h. die 2 niederwertigsten Bits sind immer Null.

Wenn der Innenwiderstand der Spannungsquelle des Messsignals zu groß ist (>3kΩ), genügt die voreingestellte Einschwingzeit des Multiplexers nicht mehr aus für eine genaue Messung. Sie können die Wartezeit bis zur nächsten Wandlung mit dem Befehl **Seq_Set_Delay** verändern.

Siehe auch

ADC, Seq_Select, Seq_Set_Delay, Seq_Status, Seq_Read

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Define module 1
#Include ADwinPro_All.Inc

Dim Data_1[16] As Long At DM_Local

Init:
  SE_Diff(module,0)          'Eingänge auf single ended stellen
  Seq_Mode(module,3,0)       'Ablaufsteuerung: Continuous Mode,
                             'Verstärkungsfaktor 1
  Seq_Select(module,55555555h) 'Alle ungeradzahligen Kanäle des
                             'Moduls AIN-32/.. messen
  Start_Conv(1)              'Messsequenzen starten
                             ' (Continuous Mode)
  Wait_EOC(1)                'Warten, bis alle angegebenen Kanäle
                             'einmal gemessen sind

Event:
  Seq_Read(module,16,Data_1,1) 'Messwerte von dem Modul holen und
                             'in Data_1 kopieren

Finish:
  Seq_Mode(module,0,0)       'rücksetzen in Standardmodus
```

Seq_Read

Seq_Read kopiert eine bestimmte Anzahl an Messwerten (16 Bit) von dem angegebenen Modul in ein Ziel-Feld.

In jedes Feldelement wird jeweils 1 Messwert kopiert.

Syntax

```
#Include ADwinPro_All.Inc
Seq_Read(module,count,array[],array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
count	Anzahl der zu lesenden Messwerte (1…32). Es sollen nur so viele Messwerte gelesen werden, wie Kanäle in der Messgruppe sind.	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden.	ARRAY LONG
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1…n).	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **Seq_Mode** die Ablaufsteuerung des Moduls aktiviert und mit **Seq_Select** eine Messgruppe festgelegt wurde.

Wenn mehr Werte gelesen werden als Kanäle in der Messgruppe definiert sind, sind die überzähligen Werte undefiniert und zu verwerfen.

Die Messwerte der Messgruppe werden von der kleinsten Kanalnummer an in aufsteigender Reihenfolge in das Zielfeld kopiert.

Siehe auch

Seq_Read_One, Seq_Read_Two, Seq_Read_Packed, Seq_Read32, Seq_Mode, Seq_Read

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim Data_1[16] As Long At DM_Local

Init:
  SE_Diff(1,0)              'Single-Ended Eingänge
  Seq_Mode(1,3,0)           'Ablaufsteuerung auf Continuous Mode,
                            'Verstärkungsfaktor 1
  Seq_Select(1,55555555h)   'Alle ungeradzahligen Kanäle einer
                            'AIN-32/.. messen
  Start_Conv(1)             'Messsequenzen im Continuous Mode
                            'starten
  Wait_EOC(1)               'Warten, bis einmal alle angegebenen
                            'Kanäle gemessen wurden

Event:
  Seq_Read(1,16,Data_1,1)   'Aktuelle Messwerte von dem Modul
                            'in Data_1 umkopieren

Finish:
  Seq_Mode(1,0,0)           'rücksetzen in Standardmodus
```

Seq_Read_One

Seq_Read_One liest einen bestimmten Messwert (16 Bit) einer Messgruppe auf dem angegebenen Modul aus.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Seq_Read_One(module,idxno)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
idxno	Indexnr., die die Position eines Kanals in der Messgruppe bezeichnet (1 … 32).	LONG
ret_val	Zuletzt gespeicherter Messwert des Kanals mit der Position idxno in der Messgruppe.	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **Seq_Mode** die Ablaufsteuerung des Moduls aktiviert und mit **Seq_Select** eine Messgruppe festgelegt wurde.

Die Indexnummer darf nicht als Nummer eines Kanals missverstanden werden, sondern sie ist die Positionsnummer eines Kanals in der mit **Seq_Select** definierten Messgruppe.

Beispielsweise wählen Sie mit der Indexnummer 4 in einer Messgruppe mit den Kanälen (1, 3, 7, 11, 12, 15) den Kanal 11 aus.

Siehe auch

Seq_Mode, Seq_Select, Seq_Status, Seq_Read

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim Data_1[32] As Float At DM_Local
Dim i As Long

Init:
  SE_Diff(1,0)              'Single-Ended Eingänge
  Seq_Mode(1,1,0)           'Ablaufsteuerung auf Single Shot,
                            'Verstärkungsfaktor 1
  Seq_Select(1,0FFFFFFFFh)  'Alle Eingänge 1…32 einer AIN-32/..
                            'messen

Event:
  Start_Conv(1)             'Messsequenz im Single Shot Modus
                            'starten
  For i=1 To 32             'Alle 32 Kanäle einer AIN-32/.. lesen
    Do                      'sobald die jeweilige
    Until(i<=Seq_Status(1)) 'Einzelmessung fertig ist:
    Data_1[i]=(Seq_Read_One(1,i)-32768)*20/65536
                            'Digit in Volt umrechnen und speichern
  Next i

Finish:
  Seq_Mode(1,0,0)           'rücksetzen in Standardmodus
```

Seq_Read_Two

Seq_Read_Two liest auf einmal 2 aufeinanderfolgende Messwerte (je 16 Bit) einer Messgruppe auf dem angegebenen Modul aus und gibt diese in einem 32 Bit-Wert zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Seq_Read_Two(module,idxno)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
idxno	Indexnummer, die die Position von je 2 Kanälen in der Messgruppe bezeichnet (0 … 15).	LONG

Indexnr.	0	1	2	…	15
Position	1, 2	3, 4	5, 6	…	31, 32

ret_val	32 Bit-Wert, der die 2 Messwerte der (mit idxno indirekt) gewählten Kanäle enthält.	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **Seq_Mode** die Ablaufsteuerung des Moduls aktiviert und mit **Seq_Select** eine Messgruppe festgelegt wurde.

Die Indexnummer darf nicht als Nummer eines Kanals missverstanden werden, sondern sie bezeichnet die Position zweier Kanäle in der mit **Seq_Select** definierten Messgruppe.

Der zurückgegebene 32 Bit-Wert enthält im unteren Wort den Messwert des Kanals mit der jeweils kleineren der beiden Kanalnummern, im oberen Wort den größeren.

Beispielsweise bedeutet die Indexnummer 1 bei einer Messgruppe mit den Kanälen (1,3, 7, 11, 12, 15), dass die Kanäle 7 und 11 ausgelesen werden. Dabei wird der Messwert des Kanals 7 im unteren Wort des Rückgabewerts, der des Kanals 11 im oberen Wort zurück gegeben.

Siehe auch

Seq_Mode, Seq_Select, Seq_Status, Seq_Read

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim Data_1[32] As Long At DM_Local
Dim i, value32 As Long

Init:
  SE_Diff(1,0)              'Single-Ended Eingänge
  Seq_Mode(1,1,0)           'Ablaufsteuerung auf Single Shot,
                            'Verstärkungsfaktor 1
  Seq_Select(1,0FFFFFFFFh)  'Alle 32 Eing. der AIN-32/.. messen

Event:
  Start_Conv(1)             'Messsequenz im Single Shot Modus
                            'starten
  For i=1 To 15 Step 2      'Alle 32 Kanäle einlesen
    Do                      'Sobald jeweils …
    Until( (i+1)<=Seq_Status(1) ) '2 Messungen fertig sind:
    value32=Seq_Read_Two(1,(i-1)/2)'Langwort abholen und
                            'speichern
    Data_1[i]=value32 And 0FFFFh 'unteres Wort speichern
    Data_1[i+1]=Shift_Right(value32,16) 'oberes Wort speichern
  Next i
  Wait_EOC(1)

Finish:
  Seq_Mode(1,0,0)           'rücksetzen in Standardmodus
```

Seq_Read_Packed

Seq_Read_Packed kopiert eine gerade Anzahl an Messwerten (16 Bit) paarweise von dem angegebenen Modul in ein Ziel-Feld.

In jedes Feldelement werden jeweils 2 Messwerte kopiert.

Syntax

```
#Include ADwinPro_All.Inc

Seq_Read_Packed(module,count,array[],array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
count	Anzahl der zu lesenden Messwerte-Paare (1…16). Es sollen nur so viele Messwerte gelesen werden, wie Kanäle in der Messgruppe sind.	LONG
array[]	Ziel-Feld, in das die Messwerte-Paare übertragen werden.	ARRAY LONG
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1…n).	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **Seq_Mode** die Ablaufsteuerung des Moduls aktiviert und mit **Seq_Select** eine Messgruppe festgelegt wurde.

Wenn mehr Werte gelesen werden als Kanäle in der Messgruppe definiert sind, sind die überzähligen Werte undefiniert und zu verwerfen. Wenn eine Messgruppe aus einer ungeraden Anzahl von Kanälen besteht, muss zwangsläufig ein überzähliger Wert gelesen werden.

Die Messwerte der Messgruppe werden von der kleinsten Kanalnummer an in aufsteigender Reihenfolge und paarweise in das Zielfeld kopiert. Ein Feldelement enthält im unteren Wort den Messwert des Kanals mit der jeweils kleineren der beiden Kanalnummern, im oberen Wort den größeren.

Siehe auch

Seq_Read_One, Seq_Read_Two, Seq_Read_Packed, Seq_Read32, Seq_Mode, Seq_Read, Seq_Select

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim Data_1[32] As Long At DM_Local

Init:
  SE_Diff(1,0)              'Single-Ended Eingänge
  Seq_Mode(1,3,0)           'Ablaufsteuerung auf Continuous Mode,
                            'Verstärkungsfaktor 1
  Seq_Select(1,0AAAAAAAAh)  'Alle geradzahligen Kanäle einer
                            'AIN-32/.. messen
  Start_Conv(1)             'Messsequenzen im Continuous Mode
                            'starten
  Wait_EOC(1)               'Warten bis einmal alle angegebenen
                            'Kanäle gemessen wurden

Event:
  Seq_Read_Packed(1,8,Data_1,1)
                            '16 Messwerte von dem Modul holen und
                            'in Data_1 kopieren

Finish:
  Seq_Mode(1,0,0)           'rücksetzen in Standardmodus
```

Seq_Read32

Seq_Read32 kopiert alle 32 Messwerte (je 16 Bit) von dem angegebenen Modul in ein Ziel-Feld.

In jedes Feldelement wird jeweils 1 Messwert kopiert.

Syntax

```
#Include ADwinPro_All.Inc
Seq_Read32(module, array[], array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden.	ARRAY LONG
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1…n).	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **Seq_Mode** die Ablaufsteuerung des Moduls aktiviert und mit **Seq_Select** eine Messgruppe festgelegt wurde.

Wenn mehr Werte gelesen werden als Kanäle in der Messgruppe definiert sind, sind die überzähligen Werte undefiniert und zu verwerfen.

Die Messwerte der Messgruppe werden von der kleinsten Kanalnummer an in aufsteigender Reihenfolge in das Zielfeld kopiert.

Wenn Sie mehr als sechzehn Messwerte benötigen, arbeitet diese Anweisung schneller als **Seq_Read**, obwohl **Seq_Read32** immer alle 32 Messwerte überträgt.

Siehe auch

Seq_Read_One, Seq_Read_Two, Seq_Read_Packed, Seq_Read32, Seq_Mode, Seq_Read, Seq_Select

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim Data_1[32] As Long At DM_Local

Init:
  SE_Diff(1,0)              'Single-Ended Eingänge
  Seq_Mode(1,3,0)           'Ablaufsteuerung auf Continuous Mode,
                            'Verstärkungsfaktor 1
  Seq_Select(1,0FFFFFFFFh)  'Alle Kanäle einer AIN-32/.. messen
  Start_Conv(1)             'Messsequenzen im Continuous Mode
                            'starten
  Wait_EOC(1)               'Warten bis einmal alle angegebenen
                            'Kanäle gemessen wurden

Event:
  Seq_Read32(1,Data_1,1)    'Messwerte von dem Modul holen
                            'und in Data_1 kopieren

Finish:
  Seq_Mode(1,0,0)           'rücksetzen in Standardmodus
```

Seq_Select

`Seq_Select` legt fest, welche Kanäle zu der Messgruppe gehören, die die Ablaufsteuerung auf dem angegebenen Modul wandeln soll.

Syntax

```
#Include ADwinPro_All.Inc
Seq_Select(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster, das die einzulesenden Kanäle bestimmt.	LONG

Bitnr.	31	…	8	…	2	1	0
Kanal-Nr.	32	…	9	…	3	2	1

Bemerkungen

Sie können in der Messgruppe eine beliebige Auswahl aus den 32 Kanälen des Moduls zusammenstellen. Die Kanäle einer Messgruppe werden automatisch in aufsteigender Reihenfolge der Kanalnummern sortiert, d.h. die Ablaufsteuerung wandelt den Kanal mit der niedrigsten Nummer zuerst. Nicht ausgewählte Kanäle werden übersprungen, ihnen ist kein Messwert zugeordnet.

Der Status- und die Lese-Anweisungen beziehen sich immer und ausschließlich auf die Gruppe der hier ausgewählten Kanäle. Wenn also die Messgruppe aus 3 Kanälen besteht, können auch nur 3 Messwerte abgeholt werden.

Bei den 32-kanaligen Modulen müssen die Eingänge mit `SE_Diff` als single ended oder differentiell eingestellt werden. Beachten Sie dabei die besondere Nummerierung der differentiellen Eingänge (1 … 8 und 17 … 24).

Siehe auch

SE_Diff, Seq_Mode, Seq_Status, Seq_Read

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim Data_1[32] As Long At DM_Local

Init:
  SE_Diff(1,1)              'Differentielle Eingänge
  Seq_Mode(1,1,0)           'Ablaufsteuerung auf Single Shot,
                            'Verstärkungsfaktor 1
  Seq_Select(1,0FF00FFh)    'Diff. Eingänge 1-8 und 17-24 einer
                            'AIN-32/.. messen

Event:
  Start_Conv(1)             'Messsequenz im Single Shot Modus
                            'starten
  Rem Hier könnte die Wartezeit sinnvoll genutzt werden
  Wait_EOC(1)               'Warten bis einmal alle angegebenen
                            'Kanäle gemessen wurden
  Seq_Read(1,16,Data_1,1)   'Messwerte von dem Modul holen
                            'und in Data_1 kopieren

Finish:
  Seq_Mode(1,0,0)           'rücksetzen in Standardmodus
```

Seq_Set_Delay

Seq_Set_Delay legt die Einschwingzeit der Ablaufsteuerung auf dem angegebenen Modul fest.

Syntax

```
#Include ADwinPro_All.Inc
Seq_Set_Delay(module,time)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
time	Anzahl der Zeiteinheiten für die Einschwingzeit der Ablaufsteuerung. 0: Werkseinstellung (Default). 1…2047: Zeit in Einheiten von 25 ns.	LONG

Bemerkungen

Die Einschwingzeit beeinflusst die Genauigkeit der Messergebnisse in starkem Maß. Tendenziell ergibt eine kürzere Einschwingzeit ungenauere und eine längere Einschwingzeit genauere Messergebnisse.

Die Wartezeit zwischen 2 Wandlungen berechnet sich nach folgender Formel:

$$\text{Wartezeit} = \text{Wandlerzeit} + 25\,\text{ns} \cdot \text{time}$$

Der Default-Wert für time ist so gewählt, dass die Wartezeit der Multiplexer-Einschwingzeit des Moduls entspricht. Beispiel: Wenn die Einschwingzeit 3 µs und die Wandlerzeit 0,5 µs betragen, entspricht die Einstellung 0 für time dem Wert 100.

Sie finden die Werte für Wandlerzeit und Einschwingzeit in der Hardware-Dokumentation Ihres Pro-Moduls.

Siehe auch

Seq_Mode, Seq_Read

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim Data_1[32] As Long At DM_Local

Init:
  SE_Diff(1,1)              'Differentielle Eingänge
  Seq_Mode(1,1,0)           'Ablaufsteuerung auf Single Shot
  Seq_Select(1,0FF00FFh)    'Diff. Eing. 1-8 und 17-24 einer
                            'AIN-32/.. messen
  Seq_Set_Delay(1,200)      'Dem Analogteil tconv + 200 * 25ns
                            'Zeit geben, um einzuschwingen

Event:
  Start_Conv(1)             'Messsequenz im Single Shot Modus
                            'starten
  Rem Hier könnte die Wartezeit sinnvoll genutzt werden
  Wait_EOC(1)               'Warten, bis einmal alle angegebenen
                            'Kanäle gemessen wurden
  Seq_Read(1,16,Data_1,1)   'Messwerte von dem Modul holen
                            'und in Data_1 kopieren

Finish:
  Seq_Mode(1,0,0)           'rücksetzen in Standardmodus
```

Seq_Status

Seq_Status ermittelt, wieviele Kanäle der Messgruppe die Ablaufsteuerung auf dem angegebenen Modul bereits gewandelt und gespeichert hat.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Seq_Status(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Anzahl der bereits gewandelten und gespeicherten Kanäle aus der Messgruppe (1 … 32).	LONG

Bemerkungen

Diese Anweisung ist nur im Modus 1 (single shot) sinnvoll einsetzbar.

Der Rückgabewert darf nicht als Nummer eines Kanals missverstanden werden, sondern bezieht sich immer auf die mit **Seq_Select** definierte Messgruppe.

Beispielsweise bedeutet der Rückgabewert 4 bei einer Messgruppe mit den Kanälen (1,3, 7, 11, 12, 15), dass Kanal 11 bereits gewandelt wurde und Kanal 12 als nächstes zur Messung ansteht. Anders gesagt: Es stehen schon 4 Werte im Zwischenspeicher des Moduls zur Abholung bereit. Sie können nun entweder diese Werte bereits auslesen und verarbeiten oder noch auf das Ende der Messsequenz warten.

Siehe auch

Seq_Mode, Seq_Select, Seq_Status, Seq_Read

Gültig für

AIn-16/14-C Rev. A, AIn-32/14 Rev. A, AIn-32/16 Rev. C, AIn-8/14 Rev. A, AIn-8/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim Data_1[32] As Float At DM_Local
Dim i As Long

Init:
  SE_Diff(1,0)               'Single-Ended Eingänge
  Seq_Mode(1,1,0)            'Ablaufsteuerung auf Single Shot,
                             'Verstärkungsfaktor 1
  Seq_Select(1,0FFFFFFFFh)   'Alle 32 Eing. einer AIN-32/.. messen

Event:
  Start_Conv(1)              'Messsequenz im Single Shot Modus
                             'starten
  For i=1 To 32              'Alle 32 Kanäle einlesen
    Do                       'sobald die
    Until(i<=Seq_Status(1))  'jeweilige Messung fertig ist:
    Data_1[i]=(Seq_Read_One(1,i)-32768)*20/65536
                             'Digit in Volt umrechnen und speichern
  Next i

Finish:
  Seq_Mode(1,0,0)            'rücksetzen in Standardmodus
```

SH_SetMode

SH_SetMode stellt den Modus der Sample- und Hold-Stufen auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.Inc
SH_SetMode(module,mode)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
mode	Modus der Sample-&Hold-Stufen: 0: Sample. 1: Hold.	LONG

Bemerkungen

Im Modus „Sample“ folgt die Ausgangsspannung des Moduls der Eingangsspannung, während im Modus „Hold“ die Ausgangsspannung auf dem Wert der Eingangsspannung gehalten wird.

Die Ausgangsspannung kann nur über eine begrenzte Zeit konstant gehalten werden (Modus „Hold“). Der entstehende Spannungsabfall („Droop-Rate“) beträgt typischerweise 1 µV/ms bei 25°C Betriebstemperatur; die Erfassungszeit (acquisition time) auf 0,01% beträgt ca. 20 µs.

Siehe auch

- / -

Gültig für

(LP)SH-8(-FI)

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  SH_SetMode(1,0)           'Modul mit der Adresse 1 wird auf
                            'Sample-Modus gestellt
  SH_SetMode(2,1)           'Modul mit der Adresse 2 wird auf
                            'Hold-Modus gestellt
```

Start_Conv

`Start_Conv` startet die A/D-Wandlung auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
Start_Conv(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Bemerkungen

Wenn das Modul mit `SyncEnable` zur Synchronisation freigeschaltet ist, hat die Anweisung `SyncAll` die gleiche Funktion wie `Start_Conv`.

Siehe auch

ADC, ADC16, ReadADC, SE_Diff, Set_Mux, SyncAll, Wait_EOC

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Deklaration

Event:
  Set_Mux(1,0)              'Multiplexer auf Eingang 1 setzen
```

3µs (12-Bit ADC) bzw. 14µs (AIn-8/16 Rev. B, AIn-32/16 Rev. B) auf das Einschwingen des Multiplexers warten*

```
  Start_Conv(1)             'Start AD-Wandlung
  Wait_EOC(1)               'Warten auf Wandlung-Ende
  value1 = ReadADC(1)       'Wert vom ADC einlesen
```

* Die Wartezeit kann z. B. durch eine Reihe von *ADbasic*-Befehlen überbrückt werden, die keine Messung auf dem selben A/D-Modul durchführen, auf dem der Multiplexer umgestellt wurde.

Eine weitere Möglichkeit zur Überbrückung bietet die folgende Warteschleife, welche jedoch nur in hoch priorisierten Prozessen angewendet werden darf (für T9 und T10; T11/T12 siehe Kapitel 5.2.4):

```
  Dim time As Long
  time = Read_Timer()       'Aktuellen Zählerstand ermitteln
  Do
  Until (Read_Timer()-time>120) '25ns*120=3µs warten
```

Alternativ für AIn-8/16 Rev. B, AIn-32/16 Rev. B einsetzen:

```
  Until (Read_Timer()-time>560)'25ns*560=14µs warten
```

Start_ConvF

`Start_ConvF` startet die Wandlung auf einem oder mehreren F-ADC des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Start_ConvF(module,adc_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
adc_no	Bitmuster, das die ADC festlegt, deren Konvertierung gestartet werden soll (siehe Tabelle): 1: ADC starten. 0: ADC nicht starten.	LONG

Bitnr.	31:8	7	6	5	4	3	2	1	0
Wandler-Nr.	–	8	7	6	5	4	3	2	1

Bemerkungen

Die Angabe der ADC erfolgt bitweise, so dass die Konvertierung von mehreren Wandlern gleichzeitig gestartet werden kann. Beispielsweise muss zum Starten von A/D-Wandler 1 und 3 das Bitmuster `0101b` (dezimal 5) übergeben werden.

Sie können eine Wandlung mit dem Befehl **SyncAll** synchron zu anderen Messungen starten, falls Sie das Modul mittels **SyncEnable** zur Synchronisation frei gegeben haben.

Sie können mehrere Wandlungen ebenfalls synchron ausführen, wenn Sie die entsprechenden Module mit **Sync_Mode** zur Synchronisation frei geben. Sobald Sie auf dem Master-Modul eine Wandlung starten, starten zeitgleich auch Wandlungen auf allen Kanälen der Slave-Module. Bei Event-gesteuerten Modulen geschieht das Gleiche, sobald am gewünschten Event-Eingang ein Signal eingeht.

Siehe auch

ADCF, ReadADCF, Read_ADCF4, Read_ADCF8, Read_ADCF4_Packed, Read_ADCF8_Packed, Wait_EOCF

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim value As Long           'Deklaration

Event:
  Start_ConvF(1,1)          'Start AD-Wandlung auf Kanal 1
  Wait_EOCF(1,1)            'Warten auf Wandlung-Ende
  value = ReadADCF(1,1)     'Wert vom ADC einlesen
```

Sync_Mode

Sync_Mode bestimmt auf dem angegebenen Modul die Art der Synchronisation mit anderen Modulen, insbesondere für Burst-Messreihen.

Syntax

```
#Include ADwinPro_All.Inc
Sync_Mode(module,mode)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
mode	Synchronisationsmodus des Moduls (0…3): 0: keine Synchronisation (voreingestellt). 1: Synchronisation als Master-Modul. 2: Synchronisation als Slave-Modul. 3: Synchronisation durch ein Signal am externen EVENT-Eingang eines beliebigen Moduls (außer am CPU-Modul).	LONG

Bemerkungen

Sobald synchronisierte Module (im Modus 2 oder 3) ein Synchron-Signal empfangen, starten sie gleichzeitig eine Wandlung auf allen Kanälen (entspricht der Funktion des Befehls **Start_ConvF**). Diese Wandlung kann Teil einer Einzelmessung oder einer Burst-Messreihe sein.

Modus „Master / Slave“

Sie dürfen nur ein einziges Master-Modul einstellen. Wenn das Master-Modul eine Wandlung startet – entweder bei einer einzelnen Wandlung (**Start_ConvF**) oder als Teil einer Burst-Messreihe – sendet es gleichzeitig ein Synchron-Signal.
Sobald Slave-Module das Signal des Master-Moduls empfangen, starten sie auf allen Kanälen eine Wandlung.

Für synchronisierte Burst-Messreihen müssen Sie den Befehl **Burst_Start** zuerst an die Slave-Module und zuletzt an das Master-Modul senden. Bei jeder Wandlung in der Messreihe sendet das Master-Modul ein Synchron-Signal, so dass alle Wandlungen der Messreihen auf allen synchronisierten Modulen zeitgleich ablaufen.

Modus „Event“

Event-synchronisierte Module starten eine Wandlung, wenn ein Signal an einem (freigegebenen) Event-Eingang eines beliebigen Moduls eingeht. Auch Signale von nicht synchronisierten Modulen sind zulässig; ein Signal am Event-Eingang des CPU-Moduls wird ignoriert.

Ein Event-Eingang wird mit dem Befehl **EventEnable** freigegeben.

Für jede Messung einer Burst-Messreihe ist ein eigenes Event-Signal erforderlich. Wenn Sie z.B. eine Burst-Messreihe mit 10000 Messungen eingestellt haben, müssen 10000 Events eingehen, damit die Messreihe vollständig abgearbeitet wird.

Wenn Sie Burst-Messreihen auf mehreren Modulen synchronisieren, sollten Sie mit **Burst_Init** auf jedem Modul die gleiche Zahl an Messungen einstellen. Dies gilt insbesondere bei Master-Slave-Synchronisierung: Die Zahl der Messungen auf dem Master-Modul muss gleich oder größer sein als auf den Slave-Modulen, sonst wird auf letzteren die Burst-Messreihe nicht gleichzeitig mit dem letzten Signal des Master-Moduls beendet.

Siehe auch

Burst_Init, Burst_CStart, Burst_Read, Burst_Read_Packed, Burst_Start, Burst_Status, Set_Gain

Gültig für

AIn-F-4/14 Rev. B, AIn-F-8/14 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define count 10000
#Define module 1
#Define sampleperiod 40    '1 MHz Messrate [=1/(25ns*40)]

Dim i As Long
Dim Data_1[count], Data_2[count], Data_3[count] As Long
Dim Data_4[count], Data_5[count], Data_6[count] As Long
Dim Data_7[count], Data_8[count], Data_9[count] As Long
Dim Data_10[count], Data_11[count], Data_12[count] As Long

Init:
  Sync_Mode(module,1)       'Master-Modul
  Sync_Mode(module+1,2)     'Slave-Modul
  Sync_Mode(module+2,2)     'Slave-Modul
  Rem Burst-Messungen für je 4 Kanäle vorbereiten und starten
  Burst_Init(module,3,sampleperiod,count)
  Burst_Init(module+1,3,sampleperiod,count)
  Burst_Init(module+2,3,sampleperiod,count)
  Burst_Start(module+1)     'Burst-Messung Slave starten
  Burst_Start(module+2)     'Burst-Messung Slave starten
  Burst_Start(module)       'Burst-Messung Master starten
  Processdelay=800          'Mit 50 kHz den Trigger-Punkt finden

Event:
  Par_1=Burst_Status(module) 'Anzahl der noch durchzuführenden
                             'Messungen
  If (Par_1=0) Then End      'Burst-Messung beendet - dann FINISH
                             'ausführen

Finish:
  Rem Die jeweils letzten gesammelten Daten aller 4 Kanäle
  Rem kopieren
  Burst_Read(module,1,1,count,Data_1,1)
  Burst_Read(module,2,1,count,Data_2,1)
  Burst_Read(module,3,1,count,Data_3,1)
  Burst_Read(module,4,1,count,Data_4,1)
  Burst_Read(module+1,1,1,count,Data_5,1)
  Burst_Read(module+1,2,1,count,Data_6,1)
  Burst_Read(module+1,3,1,count,Data_7,1)
  Burst_Read(module+1,4,1,count,Data_8,1)
  Burst_Read(module+2,1,1,count,Data_9,1)
  Burst_Read(module+2,2,1,count,Data_10,1)
  Burst_Read(module+2,3,1,count,Data_11,1)
  Burst_Read(module+2,4,1,count,Data_12,1)
```

Wait_EOC

`Wait_EOC` wartet, bis die zuletzt gestartete A/D-Wandlung abgeschlossen ist.

Syntax

```
#Include ADwinPro_All.Inc
Wait_EOC(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Siehe auch

ADC, ADC16, Set_Mux, Start_Conv, ReadADC

Gültig für

(LP)SH-8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#Include ADwinPro_All.Inc
Dim value1 As Long          'Deklaration

Init:
  Set_Mux(1,0)              'Multiplexer auf Eingang 1 setzen
  Rem An dieser Stelle das Einschwingen des Multiplexers abwarten[1]
  Rem Für 12-Bit ADC: 3µs;
  Rem Für AIn-8/16 Rev. B, AIn-32/16 Rev. B: 14µs

Event:
  Start_Conv(1)              'Start AD-Wandlung
  Wait_EOC(1)                'Warten auf Wandlung-Ende
  value1 = ReadADC(1)        'Wert vom ADC einlesen
```

Eine weitere Möglichkeit zur Überbrückung der Einschwingzeit bietet die folgende Warteschleife, welche jedoch nur in hoch priorisierten Prozessen angewendet werden darf (für T9 und T10; T11/T12 siehe Kapitel 5.2.4):

```
  Dim time As Long
  time = Read_Timer()
  Do
  Until (Read_Timer()-time>120) '25ns*120=3µs warten
```

Für AIn-8/16 Rev. B, AIn-32/16 Rev. B einsetzen:

```
  Until (Read_Timer()-time>560) '25ns*560=14µs warten
```

1. Die Wartezeit kann z. B. durch eine Reihe von *ADbasic*-Befehlen überbrückt werden, die keine Messung auf dem selben A/D-Modul durchführen, auf dem der Multiplexer umgestellt wurde.

Wait_EOCF

Wait_EOCF wartet, bis die Wandlung auf allen angegebenen F-ADC des Moduls abgeschlossen ist.

Syntax

```
#Include ADwinPro_All.Inc
Wait_EOCF(module,adc_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
adc_no	Bitmuster, das die ADC festlegt, auf deren Konvertierungsende gewartet werden soll (siehe Tabelle).	LONG

Bitnr.	31:8	7	6	5	4	3	2	1	0
Wandler-Nr.	–	8	7	6	5	4	3	2	1

Bemerkungen

Die Angabe der ADC erfolgt bitweise, so dass die Konvertierung von mehreren Wandlern gleichzeitig gestartet werden kann. Beispielsweise muss zum Starten von A/D-Wandler 1 und 3 das Bitmuster 101b (dezimal 5) übergeben werden.

Siehe auch

ADCF, Start_ConvF, ReadADCF, Read_ADCF4, Read_ADCF8, Read_ADCF4_Packed, Read_ADCF8_Packed

Gültig für

AIn-F-4/12 Rev. A, AIn-F-4/14 Rev. B, AIn-F-4/16 Rev. A, AIn-F-4/16 Rev. B, AIn-F-8/12 Rev. A, AIn-F-8/14 Rev. B, AIn-F-8/16 Rev. A, AIn-F-8/16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
Dim value As Long          'Deklaration

Event:
  Start_ConvF(1,1)         'Start AD-Wandlung
  Wait_EOCF(1,1)           'Warten auf das Ende der Konvertierung
  value = ReadADCF(1,1)    'Wert vom ADC einlesen
```

3.3 Pro I: Analoge Ausgänge

Dieser Abschnitt beschreibt folgende Befehle:

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für ein bestimmtes Modul anwendbar sind.

In den Anwendungsbeispielen wird davon ausgegangen, dass auf dem D/A-Modul die Adresse 1 eingestellt ist.

DAC

DAC gibt auf einem Kanal des angegebenen Moduls eine (analoge) Spannung aus, die dem angegebenen Digitalwert entspricht.

Syntax

```
#Include ADwinPro_All.Inc
DAC(module,dac_no,value)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dac_no	Nummer (1…4 oder 1…8) des Ausgangs.	LONG
value	Auszugebender Wert (0…65535).	LONG

Diese Prozedur besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.

WriteDAC	→	Start_DAC
Digitalwert in das DAC-Register übertragen		D/A-Wandlung starten

Siehe auch

Start_DAC, WriteDAC

Gültig für

AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, AOut-8/16 Rev. C

Beispiel

```
Rem Digitaler P-Regler
#Include ADwinPro_All.Inc
#Define set_to Par_1        'Sollwert
#Define gain FPar_2         'Verstärkungsfaktor
#Define dev Par_3           'Regelabweichung
#Define actuate Par_4       'Stellgröße

Event:
  dev = set_to - ADC(1,1)   'Regelabweichung berechnen
  actuate = dev * gain      'Stellgröße berechnen
  DAC(1,1,actuate)          'Ausgabe der Stellgröße
```

FG_Control

`FG_Control` startet oder stoppt den Funktionsgenerator (d.h. die Werteausgabe) auf den gewählten Ausgabekanälen des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
FG_Control(module,output,run_mode)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`output`	Bitmuster (0…1111b), das die einzustellenden Ausgabekanäle bestimmt, siehe Tabelle unten: Bit = 0: Betriebsmodus nicht verändern. Bit = 1: Betriebsmodus `run_mode` einstellen.	LONG
`run_mode`	Betriebsmodus (0…2) einstellen: 0: Ausgabe sofort starten (wenn Funktionsgenerator gestoppt war); Ausgabe weiter führen und vorhandenen soft stop löschen (wenn Funktionsgenerator läuft). 1: Ausgabe sofort stoppen (hard stop). 2: Ausgabe nach dem letzten Wert des zugehörigen Zwischenspeichers stoppen (soft stop).	LONG

Bitnr.	31…4	3	2	1	0
DAC-Nr.	–	4	3	2	1

Bemerkungen

Diese Anweisung hat nur dann eine Funktion, wenn der Funktionsgenerator-Modus des Moduls mit **`FG_Mode`** aktiviert wurde. Die Anweisung wirkt auf alle gewählten Kanäle gleichzeitig (synchron, s.u.).

Ein gestarteter Funktionsgenerator läuft – solange dessen Parameter unverändert bleiben – unabhängig vom Prozessor-Modul. Er wird daher von einem Boot-Vorgang nicht beeinflusst; jedoch stoppt **`FG_Mode`** in einem neu gestarteten Prozess alle Funktionsgeneratoren.

Wenn der Funktionsgenerator den letzten Wert des zugehörigen Zwischenspeichers ausgegeben hat, beginnt er im nächsten Schritt wieder mit dem ersten Wert des Zwischenspeichers (ohne Zeitverzug).

Bei einem „soft stop“ stoppt der Funktionsgenerator, sobald der letzte Wert des zugehörigen Zwischenspeichers ausgegeben wurde. Die Einstellung des Modus 0 (Start) oder 1 (hard stop) löschen einen vorhergehenden soft stop sofort; im Modus 1 stoppt die Ausgabe sofort, während im Modus 0 die Ausgabe einfach weiter läuft (kein Neustart vom Anfang des Zwischenspeichers).

Auf einem Kanal, auf dem der Funktionsgenerator gestoppt oder nicht aktiviert ist, kann ein Wert mit DAC ausgegeben werden. Hierbei kann ein Zeitverzug entstehen (siehe **`FG_Mode`**). Wird die Anweisung DAC nach einem soft stop gegeben, wird sie erst nach dem endgültigen Stopp des Funktionsgenerators ausgeführt.

Nach dem Einstellen des Betriebsmodus 0 startet die Werteausgabe innerhalb von 1 µs. Mit **`FG_Status`** kann abgefragt werden, ob der Start bereits erfolgt ist.

Siehe auch

DAC, FG_Def, FG_Delay, FG_Mode, FG_Read_Index, FG_Status, FG_Write

Gültig für

AOut-4/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim values[100] As Long     'Feld mit Ausgabedaten
Dim i As Long               'Schleifenvariable

LowInit:
  FG_Mode(1, 1)             'Modus Funktionsgenerator aktivieren
  FG_Def(1, 1, 200, 100)    'dac_no=1, startadr=200, memsize=100
  For i=1 to 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  next i
  FG_Write(1, 200, 100, values, 1) 'Startadr=200, length=100,
                            'array=values, array_idx=1
  FG_Delay(1, 1, 80000)     'dac_no = 1, scale=80000 (= 1kHz
                            'Ausgaberate); bei 100 Werten beträgt
                            'die Periodendauer 100ms

Init:
  FG_Control(1,01b, 0)      'nur DAC-Nr. 1 sofort starten

Event:
  Par_1=FG_Read_Index(1, 1) 'Ausgabeposition des
                            'Funktionsgenerators (DAC1) in Par_1
                            'übergeben

Finish:
  FG_Control(1, 01111b,2)   'softstop für alle Kanäle =
                            'Funktionsgenerator stoppen,
                            'sobald der letzte Wert des
                            'Zwischenspeichers ausgegeben wurde.
  DAC(1, 1, 49152)          'Ausgang 1 auf 5V setzen, sobald der
                            'Funktionsgenerator gestoppt ist

  Rem Warten, bis alle Funktionsgeneratoren gestoppt sind. Durch
  Rem diese Warteschleife ist sicher gestellt, dass alle Werte aus
  Rem den Zwischenspeichern ausgegeben werden, bevor der FG-Modus
  Rem ausgeschaltet wird.
  do
   until (FG_Status(1)=0)

   FG_Mode(1,0)             'Modus Funktionsgenerator aus
                            '= Standard-Modus ein.
```

FG_Def

FG_Def definiert für einen Ausgabekanal auf dem angegebenen Modul die Startadresse und die Größe des internen Zwischenspeichers für den Funktionsgenerator-Modus.

Syntax

```
#Include ADwinPro_All.Inc
FG_Def(module,dac_no,startadr,memsize)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dac_no	Nummer (1…4) des Ausgabekanals.	LONG
startadr	Startadresse (1…0FFFFFh) des Zwischenspeichers.	LONG
memsize	Größe (1…0FFFFFh) des Zwischenspeichers in 16 Bit-Werten.	LONG

Bemerkungen

Diese Anweisung hat nur dann eine Funktion, wenn der Funktionsgenerator-Modus des Moduls mit **FG_Mode** aktiviert wurde.

Für jeden verwendeten Kanal ist ein eigener Zwischenspeicher einzurichten. Der Funktionsgenerator eines Kanals darf erst gestartet werden, wenn der zugehörige Zwischenspeicher definiert ist.

Der interne Zwischenspeicher kann bis zu 1048575 (2^{20}-1) Werte zu je 16 Bit aufnehmen. Die Startadresse und Größe des Zwischenspeichers kann für jeden Kanal frei gewählt werden; eine Prüfung z.B. von Bereichs-Überschneidungen erfolgt nicht.

Die Definition eines Kanal-Zwischenspeichers kann bei laufendem Funktionsgenerator geändert werden. Die Änderung tritt (ohne Zeitverzögerung) in Kraft, sobald der Funktionsgenerator den letzten Wert aus dem bisherigen Zwischenspeicher-Bereich ausgegeben hat. Zu diesem Zeitpunkt muss der neue Speicherbereich mit Daten beschrieben sein.

Siehe auch

FG_Control, FG_Delay, FG_Mode, FG_Read_Index, FG_Status, FG_Write

Gültig für

AOut-4/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim values[100] As Long    'Feld mit Ausgabedaten
Dim i As Long              'Schleifenvariable

LowInit:
  FG_Mode(1, 1)            'Modus Funktionsgenerator aktivieren
  FG_Def(1, 1, 200, 100)   'dac_no=1, startadr=200, memsize=100
  For i=1 to 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  next i
  FG_Write(1, 200, 100, values, 1) 'Startadr=200, length=100,
                           'array=values, array_idx=1
  FG_Delay(1, 1, 80000)    'dac_no = 1, scale=80000 (= 1kHz
                           'Ausgaberate); bei 100 Werten beträgt
                           'die Periodendauer 100ms
```

FG_Delay

FG_Delay stellt auf dem angegebenen Modul die Ausgaberate des Funktionsgenerators ein.

Syntax

```
#Include ADwinPro_All.Inc
FG_Delay(module,dac_no,scale)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dac_no	Nummer (1…4) des Ausgabekanals.	LONG
scale	Skalierungsfaktor (80…2,68·108=229-1) zur Einstellung der Ausgaberate nach der Formel: Ausgaberate = 80MHz / scale.	LONG

Bemerkungen

Diese Anweisung hat nur dann eine Funktion, wenn der Funktionsgenerator-Modus des Moduls mit **FG_Mode** aktiviert wurde.

Die Ausgaberate kann über den Skalierungsfaktor im Bereich von 0,15Hz bis 1,0MHz mit einer Auflösung von 12,5ns eingestellt werden.

Die Anweisung ändert die Ausgaberate sofort.

Beachten Sie bitte, dass die Ausgaberate des Funktionsgenerators von einem eigenen Taktgeber des AOut-M2 Moduls gesteuert wird und sie damit nicht synchron zum Taktgeber des Prozessormoduls läuft.

Siehe auch

FG_Control, FG_Def, FG_Mode, FG_Read_Index, FG_Status, FG_Write

Gültig für

AOut-4/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim values[100] As Long    'Feld mit Ausgabedaten
Dim i As Long              'Schleifenvariable

LowInit:
  FG_Mode(1, 1)            'Modus Funktionsgenerator aktivieren
  FG_Def(1, 1, 200, 100)   'dac_no=1, startadr=200, memsize=100
  For i=1 to 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  next i
  FG_Write(1, 200, 100, values, 1) 'Startadr=200, length=100,
                           'array=values, array_idx=1
  FG_Delay(1, 1, 80000)    'dac_no = 1, scale=80000 (= 1kHz
                           'Ausgaberate); bei 100 Werten beträgt
                           'die Periodendauer 100ms
```

FG_Mode

FG_Mode aktiviert oder deaktiviert auf dem angegebenen Modul den Funktionsgenerator-Modus.

Syntax

```
#Include ADwinPro_All.Inc
FG_Mode(module,mode)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
mode	Betriebsmodus einstellen: 0: Funktionsgenerator aus = Normal (Default). 1: Funktionsgenerator ein.	LONG

Bemerkungen

Die Anweisungen zum Funktionsgenerator (FG_…) sollten unbedingt in der folgenden Reihenfolge verwendet werden. Verwenden Sie nach Möglichkeit auch den angegebenen Programmabschnitt:

– Funktionsgenerator-Modus aktivieren:	FG_Mode	LowInit:
– Parameter einstellen:	FG_Def, FG_Delay, FG_Write	LowInit:
– Funktionsgenerator starten:	FG_Control	Init:
– Abfrage nach Bedarf:	FG_Read_Index, FG_Status	Init: Event: Finish:
– Funktionsgenerator stoppen:	FG_Control	Finish:
– Funktionsgenerator-Modus deaktivieren:	FG_Mode	Finish:

Das Aktivieren wie auch das Deaktivieren beenden sofort alle Ausgabevorgänge der noch laufenden Funktionsgeneratoren. Wenn statt dessen eine vollständige Ausgabe der jeweiligen Zwischenspeicher-Daten gewünscht ist, muss eine Abfrage des Funktionsgenerator-Status mit FG_Status erfolgen.

Beachten Sie, dass nach jedem Aktivieren (auch ohne vorheriges Deaktivieren) alle Parameter mit FG_Def, FG_Delay und FG_Write erneut einzustellen sind. Die Initialisierung sollte nur ein einziger Prozess durchführen.

Bei inaktivem Funktionsgenerator-Modus können alle Kanäle mit den Anweisungen DAC, WriteDAC und Start_DAC angesprochen werden.

Bei aktivem Funktionsgenerator-Modus können mit den Anweisungen DAC, WriteDAC und Start_DAC nur solche Kanäle angesprochen werden, auf denen der Funktionsgenerator gestoppt ist. Bei jeder der Anweisungen kann es zu einem gelegentlichen Zeitverzug (Jitter) bis zu 1 µs kommen.

Siehe auch

FG_Control, FG_Def, FG_Delay, FG_Read_Index, FG_Status, FG_Write

Gültig für

AOut-4/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc
  Rem …
  Rem …
  Rem …

Dim values[100] As Long     'Feld mit Ausgabedaten
Dim i As Long               'Schleifenvariable

LowInit:
  FG_Mode(1, 1)             'Modus Funktionsgenerator aktivieren
  FG_Def(1, 1, 200, 100)    'dac_no=1, startadr=200, memsize=100
  For i=1 to 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  next i
  FG_Write(1, 200, 100, values, 1) 'Startadr=200, length=100,
                            'array=values, array_idx=1
  FG_Delay(1, 1, 80000)     'dac_no = 1, scale=80000 (= 1kHz
                            'Ausgaberate); bei 100 Werten beträgt
                            'die Periodendauer 100ms

Init:
  FG_Control(1,01b, 0)      'nur DAC-Nr. 1 sofort starten

Event:
  Par_1=FG_Read_Index(1, 1) 'Ausgabeposition des
                            'Funktionsgenerators (DAC1) in Par_1
                            'übergeben

Finish:
  FG_Control(1, 01111b,2)   'softstop für alle Kanäle =
                            'Funktionsgenerator stoppen,
                            'sobald der letzte Wert des
                            'Zwischenspeichers ausgegeben wurde.
  DAC(1, 1, 49152)          'Ausgang 1 auf 5V setzen, sobald der
                            'Funktionsgenerator gestoppt ist

  Rem Warten, bis alle Funktionsgeneratoren gestoppt sind. Durch
  Rem diese Warteschleife ist sicher gestellt, dass alle Werte aus
  Rem den Zwischenspeichern ausgegeben werden, bevor der FG-Modus
  Rem ausgeschaltet wird.
  do
   until (FG_Status(1)=0)

   FG_Mode(1,0)             'Modus Funktionsgenerator aus
                            '= Standard-Modus ein.
```

FG_Read_Index

FG_Read_Index gibt auf dem angegebenen Modul den Positionszeiger eines bestimmten Funktionsgenerators zurück.

Der Positionszeiger ist die absolute Speicheradresse des Wertes, der zuletzt ausgegeben wurde.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = FG_Read_Index(module,dac_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dac_no	Nummer (1…4) des Ausgabekanals.	LONG
ret_val	Speicheradresse (1…0FFFFFh) als Positionszeiger auf den zuletzt ausgegebenen Wert.	LONG

Bemerkungen

Der Positionszeiger ist nur gültig, wenn sowohl der Funktionsgenerator-Modus des Moduls mit **FG_Mode** aktiviert wurde als auch der Funktionsgenerator dieses Ausgabekanals läuft (Status-Abfrage siehe **FG_Status**). Nach dem Stoppen des Funktionsgenerators bleibt der Positionszeiger auf der Speicheradresse des zuletzt ausgegebenen Wertes stehen.

Die Ausgabeposition innerhalb des jeweiligen Zwischenspeichers eines Kanals ergibt sich aus der Differenz des Rückgabewerts `ret_val` und der bei **FG_Def** angegebenen Startadresse `startadr` des Zwischenspeichers: `ret_val` - `startadr`.

Siehe auch

FG_Control, FG_Def, FG_Delay, FG_Mode, FG_Status, FG_Write

Gültig für

AOut-4/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim values[100] As Long    'Feld mit Ausgabedaten
Dim i As Long              'Schleifenvariable

LowInit:
  FG_Mode(1, 1)            'Modus Funktionsgenerator aktivieren
  FG_Def(1, 1, 200, 100)   'dac_no=1, startadr=200, memsize=100
  For i=1 to 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  next i
  FG_Write(1, 200, 100, values, 1) 'Startadr=200, length=100,
                           'array=values, array_idx=1
  FG_Delay(1, 1, 80000)    'dac_no = 1, scale=80000 (= 1kHz
                           'Ausgaberate); Periodendauer 100ms

Init:
  FG_Control(1,01b, 0)     'nur DAC-Nr. 1 sofort starten

Event:
  Par_1=FG_Read_Index(1, 1) 'Ausgabeposition des
                           'Funktionsgenerators (DAC1) in Par_1
                           'übergeben
```

FG_Status

`FG_Status` gibt den Status aller Funktionsgeneratoren auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = FG_Status(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Bitmuster, das den Generator-Status wiedergibt. Jedem Ausgabekanal ist ein Bit zugeordnet. Bit=0: Funktionsgenerator aus. Bit=1: Funktionsgenerator läuft.	LONG

Bitnr.	31…4	3	2	1	0
DAC-Nr.	–	4	3	2	1

Bemerkungen

Diese Anweisung hat nur dann eine Funktion, wenn der Funktionsgenerator-Modus des Moduls mit `FG_Mode` aktiviert wurde.

Wenn ein Funktionsgenerator mit `FG_Control(…,2)` (soft stop) angehalten wird, wird dessen Bit in `ret_val` erst gelöscht, wenn der letzte Wert des Bereichs ausgegeben ist.

Siehe auch

FG_Control, FG_Def, FG_Delay, FG_Mode, FG_Read_Index, FG_Write

Gültig für

AOut-4/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

LowInit:
  FG_Mode(1, 1)              'Modus Funktionsgenerator aktivieren
  FG_Def(1, 1, 200, 100)     'dac_no=1, startadr=200, memsize=100

Init:
  FG_Control(1,01b, 0)       'nur DAC-Nr. 1 sofort starten

Event:
  Rem Programm …

Finish:
  do
   until (FG_Status(1)=0)    'Warten, bis alle
                             'Funktionsgeneratoren gestoppt sind.
  FG_Mode(1,0)               'Modus Funktionsgenerator aus
                             '= Standard-Modus ein.
```

FG_Write

FG_Write überträgt eine gerade Zahl von Daten eines Felds an eine bestimmte Adresse im internen Zwischenspeicher des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
FG_Write(module,startadr,length,array[],array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
startadr	Startadresse (0…0FFFFEh) im Zwischenspeicher, ab der die Daten abgelegt werden.	LONG
length	Anzahl (2…0FFFFEh) der zu übertragenden Feldelemente. Die Anzahl muss geradzahlig sein.	LONG
array[]	Quell-Feld, dessen Werte in den Speicher übertragen werden.	ARRAY LONG
array_idx	Index des ersten zu übertragenden Elements aus dem Quell-Feld.	LONG

Bemerkungen

Diese Anweisung hat nur dann eine Funktion, wenn der Funktionsgenerator-Modus des Moduls mit **FG_Mode** aktiviert wurde.

Die Parameter startadr und length müssen mit der Einstellung abgestimmt sein, die mit **FG_Def** festgelegt wurde.

Das Quell-Feld muss mindestens length+array_idx Elemente haben. Aus den Elementen des Quell-Felds wird jeweils nur das untere Wort (Bits 0…15) in den Zwischenspeicher übertragen. Die Bits 16…31 werden nicht berücksichtigt.

Die Anzahl der zu übertragenden Feldelemente muss geradzahlig sein; bei ungerader Anzahl kann das *ADwin*-System in einen instabilen Zustand geraten.

In einem hoch priorisierten Prozess dürfen Sie mit **FG_Write** nur so viele Daten auf einmal in das Modul übertragen, dass die Auslastung des *ADwin*-Systems nicht über 100% steigt. Falls dies dennoch geschieht, kann die Kommunikation mit dem PC in einen undefinierten Zustand geraten (Time-out). Diese Einschränkung besteht nicht im Abschnitt **LowInit:** und bei niederprioren Prozessen.

Siehe auch

FG_Control, FG_Def, FG_Delay, FG_Mode, FG_Read_Index, FG_Status

Gültig für

AOut-4/16 Rev. C

Beispiel

```
#Include ADwinPro_All.Inc

Dim values[100] As Long    'Feld mit Ausgabedaten
Dim i As Long              'Schleifenvariable

LowInit:
  FG_Mode(1, 1)            'Modus Funktionsgenerator aktivieren
  FG_Def(1, 1, 200, 100)   'dac_no=1, startadr=200, memsize=100

  For i=1 to 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  next i
  FG_Write(1, 200, 100, values, 1) 'Startadr=200, length=100,
                           'array=values, array_idx=1

  FG_Delay(1, 1, 80000)    'dac_no = 1, scale=80000 (= 1kHz
                           'Ausgaberate); bei 100 Werten beträgt
                           'die Periodendauer 100ms

Init:
  FG_Control(1,01b, 0)     'nur DAC-Nr. 1 sofort starten

Event:
  Rem Programm …
```

Start_DAC

Start_DAC startet die Wandlung bzw. Ausgabe aller DAC des angegebenen D/A-Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Start_DAC(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Siehe auch

WriteDAC, DAC

Gültig für

AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, AOut-8/16 Rev. C

Beispiel

```
Rem Simultane Ausgabe von zwei verschiedenen Signalverläufen
Rem auf den Ausgängen 1 und 2 eines D/A-Moduls
#Include ADwinPro_All.Inc
Dim i As Long                 'Deklaration
Init:
  i=0

Event:
  WriteDAC(1,1,i)             'Ausgaberegister DAC1 setzen
  WriteDAC(1,2,65535-i)       'Ausgaberegister DAC2 setzen
  Start_DAC(1)                'Ausgabe auf allen DAC starten
  Inc(i)
  If (i=65535) Then i=0
```

WriteDAC

WriteDAC schreibt einen Digitalwert in das Ausgaberegister eines bestimmten DAC des angegebenen Moduls.

Die Wandlung in eine Ausgangsspannung erfolgt durch den Aufruf des Befehls **Start_DAC**.

Syntax

```
#Include ADwinPro_All.Inc
WriteDAC(module,dac_no,value)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dac_no	Nummer (1…4 oder 1…8) des Ausgangs.	LONG
value	Auszugebender Wert (0…4095 bei 12 Bit-DAC, 0…65535 bei 16 Bit-DAC).	LONG

Siehe auch

Start_DAC, DAC

Gültig für

AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, AOut-8/16 Rev. C

Beispiel

```
Rem Simultane Ausgabe von vier verschiedenen Signalverläufen
Rem auf den Ausgängen 1, 2, 3 und 4 eines D/A-Moduls
Rem Die Signalverläufe sind in vier DATA-Feldern abgelegt und
Rem können vor dem Programmstart vom PC übergeben werden
#Include ADwinPro_All.Inc
Dim i As Long                'Deklaration
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_4[1000] As Long

Init:
  i=1

Event:
  WriteDAC(1,1,Data_1[i])    'Ausgaberegister DAC1 setzen
  WriteDAC(1,2,Data_2[i])    'Ausgaberegister DAC2 setzen
  WriteDAC(1,3,Data_3[i])    'Ausgaberegister DAC3 setzen
  WriteDAC(1,4,Data_4[i])    'Ausgaberegister DAC4 setzen
  Start_DAC(1)               'Ausgabe auf allen DAC starten
  Inc(i)
  If (i>1000) Then i=1
```

WriteDAC32

WriteDAC32 schreibt einen Digitalwert in das Ausgaberegister eines bestimmten DAC des angegebenen Moduls.

Die Wandlung in eine Ausgangsspannung erfolgt durch den Aufruf des Befehls **Start_DAC**.

Syntax

```
#Include ADwinPro_All.Inc
WriteDAC32(module,dac_no,value)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dac_no	Nummer (0…1 oder 0…3) zur Auswahl eines Ausgangspaars: 0: Ausgänge 1 und 2 1: Ausgänge 3 und 4 2: Ausgänge 5 und 6 3: Ausgänge 7 und 8	LONG
value	Auszugebender Wert (0…4095 bei 12 Bit-DAC, 0…65535 bei 16 Bit-DAC).	LONG

Bemerkungen

Start_DAC startet die Wandlung aller DAC-Werte in Ausgangsspannungen.

Das untere Wort von value (Bits 0…15) wird in das Register des ungerade nummerierten Ausgangs geschrieben, das obere Wort (Bits 16…31) in das Register des gerade nummerierten Ausgangs.

Siehe auch

Start_DAC, DAC

Gültig für

AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. B, AOut-8/16 Rev. C

Beispiel

```
Rem Simultane Ausgabe von vier verschiedenen Signalverläufen
Rem auf den Ausgängen 1, 2, 3 und 4 eines D/A-Moduls
Rem Die Signalverläufe sind in vier DATA-Feldern abgelegt und
Rem können vor dem Programmstart vom PC übergeben werden
#Include ADwinPro_All.Inc
Dim i As Long                 'Deklaration
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_4[1000] As Long

Init:
  i=1

Event:
  WriteDAC32(1,1,Data_1[i]) 'Ausgaberegister DAC1 setzen
  WriteDAC32(1,2,Data_2[i]) 'Ausgaberegister DAC2 setzen
  WriteDAC32(1,3,Data_3[i]) 'Ausgaberegister DAC3 setzen
  WriteDAC32(1,4,Data_4[i]) 'Ausgaberegister DAC4 setzen
  Start_DAC(1)              'Ausgabe auf allen DAC starten
  Inc(i)
  If (i>1000) Then i=1
```

3.4 Pro I: Digitale Ein- und Ausgänge

Dieser Abschnitt beschreibt Befehle, die für Pro II-Module mit digitalen Eingängen und Ausgängen gelten:

Zähler

Digitale Ein-/Ausgänge

PWM-Ausgänge

SSI-Encoder

Komparator

- Comp_Digin_Word (Seite 137)
- Comp_Digin_Word_Diff (Seite 138)
- Comp_Fifo_Read (Seite 139)
- Comp_Fifo_Select (Seite 140)
- Comp_Read (Seite 141)
- Comp_Reset (Seite 142)
- Comp_Set (Seite 143)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für ein bestimmtes Modul anwendbar sind.

In den Anwendungsbeispielen wird davon ausgegangen, dass auf dem A/D-Modul die Adresse 1 eingestellt ist.

Cnt_Clear

Cnt_Clear setzt einen oder mehrere Zähler auf Null, gemäß dem Bitmuster in pattern.

Syntax

```
#Include ADwinPro_All.Inc
Cnt_Clear(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster, Zuordnung zu Zählern siehe Tabelle. Bit = 0: keine Funktion. Bit = 1: Zähler zurücksetzen.	LONG

Modul	Bitnr.				
	15 … 4	3	2	1	0
Pro-CNT-16/16	–	4, 8, 12, 16	3, 7, 11, 15	2, 6, 10, 14	1, 5, 9, 13
Pro-CNT-8/32	–	4, 8	3, 7	2, 6	1, 5
Pro-CNT-16/32	16 … 5	4	3	2	1
Pro-CNT-PW4 Pro-CNT-VR2PW2 Pro-CNT-VR4 / -4L Pro-CO4-T Pro-CO4-I Pro-CO4-D	–	4	3	2	1

Siehe auch

Cnt_Enable, Cnt_SetMode

Gültig für

CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR2PW2, CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T

Beispiel

In dieser Form nur für das Modul Pro-CNT-VR4 gültig!

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  Cnt_SetMode(module,1111b) 'Zähler 1..4 auf
                            'Takt/Richtungsauswertung setzen
  Cnt_Clear(module,1111b)   'Zählerstand Zähler 1..4 auf 0 setzen
  Cnt_Enable(module,1111b)  'Zähler 1..4 aktivieren
```

Cnt_Enable

`Cnt_Enable` aktiviert oder deaktiviert einen oder mehrere Zähler auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
Cnt_Enable(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Zähleraktivierung nach Bitmuster, Zuordnung zu Zählern siehe Tabelle. Bit = 0: Zähler deaktivieren / sperren. Bit = 1: Zähler aktivieren / freigeben.	LONG

Modul	Bitnr.						
	15	…	4	3	2	1	0
Pro-CNT-16/16		–		4, 8, 12, 16	3, 7, 11, 15	2, 6, 10, 14	1, 5, 9, 13
Pro-CNT-8/32		–		4, 8	3, 7	2, 6	1, 5
Pro-CNT-16/32	16	…	5	4	3	2	1
Pro-CNT-VR4 Pro-CNT-VR2PW2 Pro-CNT-PW4 Pro-CO4-T Pro-CO4-I Pro-CO4-D		–		4	3	2	1

Siehe auch

Cnt_Clear, Cnt_SetMode

Gültig für

CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR2PW2, CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T

Beispiel

In dieser Form nur für das Modul Pro-CNT-VR4 gültig!

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  Cnt_SetMode(module,1000b) 'Zähler 4 auf Takt/Richtungs-
                            'auswertung, alle anderen Zähler auf
                            'Vierfachflankenauswertung setzen
  Cnt_Clear(module,1000b)   'Zählerstand Zähler 4 auf 0 setzen
  Cnt_Enable(module,1000b)  'Zähler 4 aktivieren, alle anderen
                            'deaktivieren
```

Cnt_Latch

Cnt_Latch übernimmt den aktuellen Zählerstand eines oder mehrerer Zähler auf dem angegebenen Modul in die jeweiligen Zwischenregister (= latchen).

Syntax

```
#Include ADwinPro_All.Inc
Cnt_Latch(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Zählerstand „latchen“ gemäß Bitmuster, Zuordnung zu Zählern siehe Tabelle. Bit = 0: keine Funktion. Bit = 1: Zählerstand ins Zwischenregister übernehmen.	LONG

Modul	Bitnr.						
	15	…	4	3	2	1	0
Pro-CNT-16/16		–		4, 8, 12, 16	3, 7, 11, 15	2, 6, 10, 14	1, 5, 9, 13
Pro-CNT-8/32		–		4, 8	3, 7	2, 6	1, 5
Pro-CNT-16/32	16	…	5	4	3	2	1
Pro-CNT-VR4 Pro-CNT-VR2PW2 Pro-CNT-PW4 Pro-CO4-T Pro-CO4-I Pro-CO4-D		–		4	3	2	1

Siehe auch

Cnt_ReadLatch16, Cnt_ReadLatch32

Gültig für

CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR2PW2, CNT-VR4(-I), CNT-VR4L(-I), CO4-D, CO4-I, CO4-T

Beispiel

In dieser Form nur für das Modul Pro-CNT-VR4 gültig!

```
#Include ADwinPro_All.Inc
#Define module 1
Dim value As Long

Init:
  value = ADC(module,1)     'Messwert aufnehmen
  If (value > 49151) Then
    Cnt_Latch(module, 0011b)'Zähler 1 u. 2 latchen
    Rem Differenz Zähler 1 u. 2 bilden
    Par_1 = Cnt_ReadLatch32(module,1) - Cnt_ReadLatch32(module,2)
  EndIf
```

Cnt_Read16

Cnt_Read16 gibt den aktuellen Zählerstand eines 16 Bit-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Cnt_Read16(module,cnt_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
cnt_no	Zählernummer (1…16).	LONG
ret_val	Aktueller Zählerstand (16 Bit-Wert).	LONG

Bemerkungen

Diese Anweisung besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.

Cnt_Latch	→	Cnt_ReadLatch16
Aktuellen Zählerstand in das Zwischenregister übernehmen		Zwischenregister auslesen

Für spezielle Anwendungen können Sie auch diese Befehle anstelle von **Cnt_Read16** verwenden.

Siehe auch

Cnt_Read32, Cnt_Latch, Cnt_ReadLatch16

Gültig für

CNT-16/16(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Event:
  Par_1 = Cnt_Read16(module,1)'Zählerstand von Zähler 1
  Par_2 = Cnt_Read16(module,2)'Zählerstand von Zähler 2
```

Cnt_Read32

Cnt_Read32 gibt den aktuellen Zählerstand eines 32 Bit-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Cnt_Read32(module,cnt_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
cnt_no	Zählernummer (1…4 / 1…8).	LONG
ret_val	Aktueller Zählerstand (32 Bit-Wert).	LONG

Bemerkungen

Diese Prozedur besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.

Cnt_Latch	→	Cnt_ReadLatch32
Aktuellen Zählerstand in das Zwischenregister übernehmen		Zwischenregister auslesen

Für spezielle Anwendungen können Sie auch diese Befehle anstelle von **Cnt_Read32** verwenden.

Siehe auch

Cnt_Read16, Cnt_Latch, Cnt_ReadLatch32

Gültig für

CNT-8/32(-I), CNT-PW4(-I), CNT-VR2PW2, CNT-VR4(-I), CNT-VR4L(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Event:
  Par_1 = Cnt_Read32(module,3)'Zählerstand von Zähler 3
  Par_2 = Cnt_Read32(module,4)'Zählerstand von Zähler 4
```

Cnt_ReadLatch16

Cnt_ReadLatch16 gibt den Wert aus dem Zwischenregister eines 16 Bit-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Cnt_ReadLatch16(module,cnt_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
cnt_no	Zählernummer (1…16).	LONG
ret_val	Aktueller Zählerstand (16 Bit-Wert).	LONG

Bemerkungen

Um den aktuellen Zählerstand zu erhalten, muss dieser zuerst mit **Cnt_Latch** in das Zwischenregister übernommen werden und kann dann ausgelesen werden.

Siehe auch

Cnt_Latch, Cnt_Read16

Gültig für

CNT-16/16(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim value As Long

Init:
  value = ADC(module,1)     'Messwert aufnehmen
  If (value > 49151) Then
    Cnt_Latch(module, 0011b)'Zähler 1 u. 2 latchen
    Rem Differenz Zähler 1 u. 2 bilden
    Par_1 = Cnt_ReadLatch16(module,1) - Cnt_ReadLatch16(module,2)
  EndIf
```

Cnt_ReadLatch32

Cnt_ReadLatch32 gibt den Wert aus dem Zwischenregister eines 32 Bit-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Cnt_ReadLatch32(module,cnt_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
cnt_no	Zählernummer (1…4 / 1…8).	LONG
ret_val	Aktueller Zählerstand (32 Bit-Wert).	LONG

Bemerkungen

Um den aktuellen Zählerstand zu erhalten, muss dieser zuerst mit **Cnt_Latch** in das Zwischenregister übernommen werden und kann dann ausgelesen werden.

Siehe auch

Cnt_Latch, Cnt_Read32

Gültig für

CNT-8/32(-I), CNT-PW4(-I), CNT-VR2PW2, CNT-VR4(-I), CNT-VR4L(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim value As Long

Init:
  value = ADC(module,1)     'Messwert aufnehmen
  If (value > 49151) Then
    Cnt_Latch(module, 0011b)'Zähler 1 u. 2 latchen
    Rem Differenz Zähler 1 u. 2 bilden
    Par_1 = Cnt_ReadLatch32(module,1) - Cnt_ReadLatch32(module,2)
  EndIf
```

Cnt_SetMode

Cnt_SetMode stellt die Betriebsart aller Zähler auf dem angegebenen Modul ein, Vierfach-Flankenauswertung oder Takt- und Richtungseingang.

Syntax

```
#Include ADwinPro_All.Inc
Cnt_SetMode(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster zur Einstellung des Betriebsmodus der Zähler: Bit = 0: Modus Vierfach-Flankenauswertung. Bit = 1: Modus Takt- und Richtungseingang.	LONG

Modul	Bit 3	Bit 2	Bit 1	Bit 0
Pro-CNT-VR4 Pro-CNT-VR2PW2	4	3	2	1

Bemerkungen

Die Vorwärts-/Rückwärtszähler können in zwei verschiedenen Modi betrieben werden. Der Modus Vierfach-Flankenauswertung wird für Geber mit zwei um 90° phasenverschobenen Signalen benutzt. Der Modus Takt- und Richtungseingang wird für alle anderen Geber verwendet.

Siehe auch

Cnt_Clear, Cnt_Enable

Gültig für

CNT-VR2PW2, CNT-VR4(-I), CNT-VR4L(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  REM Zähler 3+4 auf Takt-/Richtungsauswertung,
  REM Zähler 1+2 auf Vierfach-Flankenauswertung
  Cnt_SetMode(module,1100b)
  Cnt_Clear(module,1100b)   'Zähler 3+4 auf 0 (Null) setzen
  Cnt_Enable(module,1100b)  'Zähler 3+4 aktivieren,
                            'alle anderen deaktivieren
```

CO4_ClearEnable

CO4_ClearEnable schaltet den externen Eingang CLR eines oder mehrerer Zähler auf dem angegebenen Modul frei.

Syntax

```
#Include ADwinPro_All.Inc
CO4_ClearEnable(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster zur Zuordnung zu Zählern (siehe Tabelle). Bit = 0: keine Funktion. Bit = 1: Eingang CLR frei schalten.	LONG

	Bitnr.							
	7	6	5	4	3	2	1	0
Zählernummer	4*	3*	2*	1*	4	3	2	1

*Nur beim Betrieb im Modus „Vierfach-Flankenauswertung“:
Bit = 0: Zähler löschen, wenn die Signale A, B und CLR auf „High“ gehen.
Bit = 1: Zähler löschen, wenn CLR auf „High“ geht.

Bemerkungen

Der externe Eingang darf entweder mit der Anweisung **CO4_ClearEnable** oder mit **CO4_LatchEnable** freigeschaltet sein, nicht aber mit beiden Anweisungen gleichzeitig!

Siehe auch

CO4_LatchEnable, CO4_SetMode

Gültig für

CO4-D, CO4-I, CO4-T

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  CO4_SetMode(module,1,1)   'Zähler 1: CLK/DIR-Modus
  CO4_LatchEnable(module,0) 'Eingang Latch für alle Zähler sperren
  CO4_ClearEnable(module,1) 'Eingang Clear für Zähler 1 freigeben
  Cnt_Clear(module,1)       'Zähler 1 löschen
  Cnt_Enable(module,1)      'Zähler 1 starten
Event:
  Par_1 = CO4_Read(module,1)'Zähler 1 lesen
```

CO4_GetStatus

`CO4_GetStatus` gibt den Status der Eingangssignale eines Zählers auf dem angegebenen Modul in einem Bitmuster zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = CO4_GetStatus(module,cnt_no)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`cnt_no`	Zählernummer (1…4).	LONG
`ret_val`	Bitmuster, das den Status der Zählereingänge darstellt (Bits 31…7 sind ohne Bedeutung):	LONG

Bitnr.						
6	5	4	3	2	1	0
Aktueller Zustand eines Signaleingangs (bei Pro-CO4-D das Signal nach den Pegelumsetzern)			**Statusmeldung** (Bit = 1)			
			Signal: einmal lesbar		**Fehler: mehrmals lesbar**	
Eingang B	Eingang A	Eingang CLR/ LATCH	Signal LATCH liegt an	Signal CLR liegt an	Korre-lations-fehler	Kabel-fehler

Fehler (mehrmals lesbare Statusmeldung): Verwenden Sie zum Löschen der Meldung den Befehl `CO4_ResetStatus`.

Bit 0 = 1: Kabelfehler; die differentiellen Signale (A & /A oder B & /B oder C & /C; C = CLR-/LATCH) weisen einen der folgenden Fehler auf:
Offene Leitung (Kabelbruch), Kurzschluss, zu kleine Signalspannung, zu hohe Gleichtaktspannung oder zu kleine Slew-Rate (< 0,33 V/µs).

Bit 1 = 1: Korrelationsfehler; gleichzeitige Änderung von A und B anstelle einer Phasenverschiebung.

Signal (einmal lesbare Statusmeldung): Die Statusmeldung wird durch Auslesen gelöscht.

Bit 2 = 1: CLR-Signal liegt an (wenn es durch `CO4_ClearEnable` freigeschaltet wurde).

Bit 3 = 1: LATCH-Signal liegt an (wenn es durch `CO4_LatchEnable` freigeschaltet wurde).

Bemerkungen

Ein Kabelfehler (Bit 0) kann nur bei differentiellen Eingängen erkannt werden! Bei TTL-Eingängen sind diese Bits stets 0 (Null) und der Befehl `CO4_ResetStatus` hat keine Wirkung.

Auch wenn Fehler auftreten, werden die Zähler dadurch nicht gestoppt.

Siehe auch

CO4_ResetStatus

Gültig für

CO4-D, CO4-I, CO4-T

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim error As Long

Init:
  CO4_SetMode(module,1,0)   'Zähler 1 Vierflanken-Modus
  Cnt_Clear(module,1)       'Zähler 1 löschen
  Cnt_Enable(module,1)      'Zähler 1 starten
  CO4_ResetStatus(module,1)'Status-Register löschen
  error = 0                 'Fehlerindikator zurücksetzen

Event:
  Par_1 = CO4_Read(module,1)'Zähler 1 auslesen
  Par_2 = CO4_GetStatus(module,1)'Zähler 1 auslesen
  If (Par_2 And 1 = 1) Then'Kabelfehler?
  Inc Par_3                 'Anzahl Kabelfehler bisher
  error = 1                 'Fehlerindikator setzen
  EndIf
  If (Par_2 And 2 = 2) Then'Korrelationsfehler?
    Inc Par_4               'Anzahl Korrelationsfehler bisher
    error = 1               'Fehlerindikator setzen
  EndIf
  Par_5 = Shift_Right(Par_2 And 16,4)'akt. Zustand CLR-Eingang
  Par_6 = Shift_Right(Par_2 And 32,5)'akt. Zustand A-Eingang
  Par_7 = Shift_Right(Par_2 And 64,6)'akt. Zustand B-Eingang
  If (error = 1) Then       'Fehlerindikator gesetzt?
    CO4_ResetStatus(module,1)'Status-Register zurücksetzen
    error = 0               'Fehlerindikator zurücksetzen
  EndIf
```

CO4_LatchEnable

CO4_LatchEnable schaltet den externen Eingang LATCH eines oder mehrerer Zähler auf dem angegebenen Modul frei.

Syntax

```
#Include ADwinPro_All.Inc
CO4_LatchEnable(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster zur Zuordnung zu Zählern (siehe Tabelle): Bit = 0: keine Funktion. Bit = 1: Eingang LATCH freischalten.	LONG

Bitnr.	Bit 3	Bit 2	Bit 1	Bit 0
Zähler-Nr.	4	3	2	1

Bemerkungen

Der externe Eingang darf entweder mit der Anweisung CO4_ClearEnable oder mit CO4_LatchEnable freigeschaltet sein, nicht aber mit beiden Anweisungen gleichzeitig!

Siehe auch

CO4_ClearEnable

Gültig für

CO4-D, CO4-I, CO4-T

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  CO4_SetMode(module,1,1)   'Zähler 1: CLK/DIR-Modus
  CO4_ClearEnable(module,0) 'Eingang Clear für alle Zähler sperren
  CO4_LatchEnable(module,1) 'Eingang Latch (Zähler 1) freigeben
  Cnt_Clear(module,1)       'Zähler 1 löschen
  Cnt_Enable(module,1)      'Zähler 1 starten

Event:
  Par_1 = CO4_ReadLatch(module,1) 'Zähler 1 auslesen
```

CO4_Read

CO4_Read gibt den aktuellen Zählerstand eines Zählers des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = CO4_Read(module,cnt_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
cnt_no	Zählernummer (1…4 / 1…16).	LONG
ret_val	Aktueller Zählerstand des spezifizierten Zählers.	LONG

Bemerkungen

Diese Prozedur besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.

Cnt_Latch	→	CO4_ReadLatch
Aktuellen Zählerstand in das Zwischenregister übernehmen		Zwischenregister auslesen

Für spezielle Anwendungen können Sie diese Befehle auch anstelle von **CO4_Read** verwenden.

Siehe auch

Cnt_Latch, CO4_ReadLatch

Gültig für

CNT-16/32(-I), CO4-D, CO4-I, CO4-T

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  CO4_SetMode(module,1,1)   'Zähler 1: CLK/DIR-Modus
  Cnt_Clear(module,1)       'Zähler 1 löschen
  Cnt_Enable(module,1)      'Zähler 1 starten

Event:
  Par_1 = CO4_Read(module,1)'Zählerstand von Zähler 1
```

CO4_ReadLatch

CO4_ReadLatch gibt den Wert aus dem Zwischenregister (Latch) eines Zählers des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = CO4_ReadLatch(module,cnt_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
cnt_no	Zählernummer (1…4 / 1…16).	LONG
ret_val	Wert aus dem Zwischenregister des Zählers.	LONG

Bemerkungen

Um den aktuellen Zählerstand zu erhalten, muss dieser zuerst mit dem Befehl **Cnt_Latch** in das Zwischenregister übernommen werden und kann dann mit **CO4_ReadLatch** gelesen werden.

Siehe auch

Cnt_Latch, CO4_Read

Gültig für

CNT-16/32(-I), CO4-D, CO4-I, CO4-T

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim old, new As Long        'Variablen dimensionieren

Init:
  old = 0                   'Variable initialisieren
  CO4_SetMode(module,1,1)   'Zähler 1: CLK/DIR-Modus
  Cnt_Clear(module,1)       'Zähler 1 zurücksetzen
  Cnt_Enable(module,1)      'Zähler 1 starten

Event:
  Cnt_Latch(module,1)       'Zähler 1 latchen und…
  new = CO4_ReadLatch(module,1)'Latch auslesen
  Par_1 = new - old         'Differenz bilden (f = Impulse / Zeit)
  old = new                 'Neuen Zählerstand als alten
                            'speichern
```

CO4_ResetStatus

CO4_ResetStatus löscht das Statusregister eines oder mehrerer Zähler auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
CO4_ResetStatus(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster zur Zuordnung zu Zählern (siehe Tabelle). Bit = 0: keine Funktion. Bit = 1: Statusregister-Bits 0 und 1 löschen.	LONG

Bitnr.	Bit 3	Bit 2	Bit 1	Bit 0
Zähler-Nr.	4	3	2	1

Bemerkungen

Das Statusregister enthält den Status der Eingangssignale eines Zählers (Auslesen mit **CO4_GetStatus**).

Siehe auch

CO4_GetStatus

Gültig für

CO4-D, CO4-I, CO4-T

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim error As Long

Init:
  CO4_SetMode(module,1,0)   'Zähler 1: Vierflanken-Modus
  Cnt_Clear(module,0001b)   'Zähler 1 löschen
  Cnt_Enable(module,0001b)  'Zähler 1 starten
  CO4_ResetStatus(module,0001b)'Status-Register löschen
  error = 0                 'Fehlerindikator zurücksetzen

Event:
  Par_1 = CO4_Read(module,1)'Zähler 1 auslesen
  Par_2 = CO4_GetStatus(module,1)'Eingangs-Status Zähler 1 lesen
  If (Par_2 And 1 = 1) Then'Kabelfehler?
    Inc Par_3               'Anzahl Kabelfehler bisher
    error = 1               'Fehlerindikator setzen
  EndIf
  If (Par_2 And 2 = 2) Then'Korrelationsfehler?
    Inc Par_4               'Anzahl Korrelationsfehler bisher
    error = 1               'Fehlerindikator setzen
  EndIf
  Par_5 = Shift_Right(Par_2 And 16,4)'akt. Zustand CLR-Eingang
  Par_6 = Shift_Right(Par_2 And 32,5)'akt. Zustand A-Eingang
  Par_7 = Shift_Right(Par_2 And 64,6)'akt. Zustand B-Eingang
  If (error = 1) Then       'Fehlerindikator gesetzt?
    CO4_ResetStatus(module,0001b)'Status-Register zurücksetzen
    error = 0               'Fehlerindikator zurücksetzen
  EndIf
```

CO4_Set_Latch-Mode

`CO4_Set_LatchMode` bestimmt den Modus der Latch-Eingänge für alle Zähler des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
CO4_Set_LatchMode(module,pattern)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`pattern`	Bitmuster für den Latch-Modus der Zähler (siehe Tabellen); voreingestellt ist 00000000b.	LONG

	Zielregister für Software-Latch (`Cnt_Latch`)				Latch-Inhalt in Zusatz-Latch kopieren			
Bitnr. in `pattern`	7	6	5	4	3	2	1	0
Zähler Nr.	4	3	2	1	4	3	2	1

Bit-Wert	Zielregister für Software-Latch (`Cnt_Latch`)	Latch-Inhalt in Zusatz-Latch kopieren
Bit = 0	Zählerstand wird in das Latch für negative Flanken übertragen: Latch 5…8	Bei jeder positiven Flanke wird der Latch-Inhalt für negative Flanken (5…8) in das Zusatz-Latch 9…12 kopiert.
Bit = 1	Zählerstand wird in das Latch für positive Flanken übertragen: Latch 1…4	Bei jeder negativen Flanke wird der Latch-Inhalt für positive Flanken (1…4) in das Zusatz-Latch 9…12 kopiert.

Bemerkungen

Der Befehl ist nur im Zusammenhang mit einer PWM-Analyse sinnvoll einsetzbar.

Sie sollten bereits Erfahrung mit der PWM-Analyse auf einem *ADwin-Pro*-System gesammelt haben, bevor Sie diesen Befehl verwenden. Wenden Sie sich für Detailfragen bitte an unseren Support.

Der Befehl `CO4_SET_LATCHMODE` legt fest,

- ob entweder der Latch-Inhalt für positive Flanken oder derjenige für negative Flanken in einem Zusatz-Latch gesichert wird.
  Dies ermöglicht, einen einmaligen schnellen Wechsel von großer zu kleiner Pulsbreite zu erfassen.
- in welches Zielregister der Zählerstand des CO4-Moduls bei einem Software-Latch übertragen wird.

Siehe auch

Cnt_Latch

Gültig für

CO4-D, CO4-I, CO4-T

Beispiel

- / -

CO4_SetMode

CO4_SetMode stellt den Zählmodus eines Zählers auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.Inc
CO4_SetMode(module,cnt_no,mode)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
cnt_no	Zählernummer (1…4).	LONG
mode	Zähler-Modus: 0: Vierflankenauswertung (A- und B-Signaleingänge). 1: Takt- und Richtung (CLK- und DIR-Eingänge). 2: PWM-Analyse.	LONG

Bemerkungen

Die Zähler können in 3 verschiedenen Modi betrieben werden:

- Die Vierfachflankenauswertung wird für Inkrementalgeber mit zwei um 90° phasenverschobenen Signalen benutzt.
- Der Takt- u. Richtungseingang wird für allgemeine Anwendungen verwendet.
- Im PWM-Modus ist die Analyse eines PWM-Signales möglich, d.h. Frequenz, Periodendauer und Impuls- sowie Pausenzeiten (bzw. Tastverhältnis) können ermittelt werden.

Siehe auch

Cnt_Clear, Cnt_Enable

Gültig für

CO4-D, CO4-I, CO4-T

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
#Define refCLK 40E6
Dim rise, rise_old, fall, fall_old, T As Long

Init:
  rise_old = 0
  fall_old = 0
  CO4_SetMode(module,1,2)   'Zähler 1: PWM-Analyse
  Cnt_Clear(module,1)       'Zähler 1 löschen
  Cnt_Enable(module,1)      'Zähler 1 starten

Event:
  rise = CO4_ReadLatch(module,1)'Latch 1 auslesen
                            '(pos. Flanke, Zähler 1)
  fall = CO4_ReadLatch(module,5)'Latch 5 auslesen
                            '(neg. Flanke, Zähler 1)
  If (rise <> rise_old) Then'Pos. Flanke detektiert?
    If (fall <> fall_old) Then 'Neg. Flanke detektiert,
                            'd.h. PWM-Signal = low?
      Par_1 = fall - rise    'Impulsdauer in Anzahl Perioden des
                            'Ref.-Taktes
      Par_2 = rise - fall_old'Pausendauer in Anzahl Perioden des
                            'Ref.-Taktes
    Else                     'Keine neg. Flanke detektiert,
                            'd.h. PWM-Signal = HIGH?
      Par_1 = fall - rise_old 'Impulsdauer in Anzahl Perioden des
                            'Ref.-Taktes
      Par_2 = rise - fall    'Pausendauer in Anzahl Perioden des
                            'Ref.-Taktes
    EndIf
  EndIf
  T = Par_1 + Par_2         'Periodendauer in Anzahl Perioden des
                            'Ref.-Taktes
  FPar_1 = refCLK / T       'Frequenz des PWM-Signals
  FPar_2 = Par_1 * 100 / T 'Tastverhältnis in Prozent
  rise_old = rise           'Latch-Inhalt speichern
  fall_old = fall           'Latch-Inhalt speichern
```

Dig_Latch

Dig_Latch überträgt auf dem angegebenen Modul Digital-Informationen von den Eingängen in die Eingangs-Latches und/oder von den Ausgangs-Latches zu den Ausgängen.

Syntax

```
#Include ADwinPro_All.Inc
Dig_Latch(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Bemerkungen

Bemerkungen

Wir empfehlen bei den Modulen Pro-DIO-32 und Pro-DIO-32 Rev. B, die Leitungen zunächst mit den Anweisungen **DigProg1** und **DigProg2** als Eingänge oder Ausgänge zu programmieren.

Abhängig vom Modul überträgt die Anweisung folgende Informationen:

Modul	Eingangs-Signal an Eingangs-Latches	Ausgangs-Latches an Ausgänge
Pro-OPT-16	x	–
Pro-REL-16 Pro-TRA-16	–	x
Pro-DIO-32 Pro-DIO-32 Rev. B	x	x

Wenn das angegebene Modul durch **SyncEnable** zur Synchronisation freigeschaltet ist, hat der Befehl **SyncAll** die gleiche Funktion wie der Befehl **Dig_Latch**.

Siehe auch

Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32, ExtLch_Enable

DigProg1, DigProg2, Digin_Word1, Digin_Word2, Digout, Digout_Word1, Digout_Word2

Digin_Long_F, Digout_Bits_F, Digout_F, Digout_Long_F

Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

SyncAll, SyncEnable

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B, OPT-16 Rev. A, OPT-16 Rev. B, REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  DigProg1(module,0FFFFh)   'DIO15:00 (Low-Word) des DIO-32-
                            'Moduls als Ausgang
  DigProg2(module,0000h)    'DIO31:16 (High-Word) des DIO-32-
                            'Moduls als Eingang
  Dig_WriteLatch1(module,0000h)'Alle Ausgangs-Bits auf 0 setzen

Event:
  Dig_Latch(module)         'Eingänge latchen, Inhalt des
                            'Ausgangs-Latches ausgeben
  Rem weitere Programmschritte
  Par_1 = Dig_ReadLatch2(module)'High-Word einlesen und
  REM .. beim nächsten Event im Low-Word ausgeben
  Dig_WriteLatch1(module,Par_1)
```

Dig_ReadLatch1

Dig_ReadLatch1 liefert die unteren 16 Bit (Bit 0…Bit 15) aus dem Zwischenregister für die digitalen Eingänge des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Dig_ReadLatch1(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **DigProg1** als Eingänge zu programmieren.

Sie können den aktuellen Zustand der digitalen Eingänge mit folgenden Anweisungen in das Zwischenregister übernehmen:

- **Digin_Word1**
- **Digin_Word2**
- **SyncAll** (falls für das Modul aktiviert).

Siehe auch

Dig_Latch, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32, ExtLch_Enable

DigProg1, DigProg2, Digin_Word1, Digin_Word2, Digout, Digout_Word1, Digout_Word2

Digin_Long_F

SyncAll, SyncEnable

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B, OPT-16 Rev. A, OPT-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module1 3
#Define module2 4
Dim value As Long

Init:
  Rem DIO15:00 der Module 1+2 als Eingänge setzen
  DigProg1(module,0000h)
  DigProg1(module2,0000h)
  SyncEnable(module,dio,1) 'Synchron. Dig.-Modul 1 freigeben
  SyncEnable(module2,dio,1)'Synchron. Dig.-Modul 2 freigeben

Event:
  Rem Pegel an den digitalen Eingängen von beiden Modulen synchron
  Rem in die Zwischenregister übernehmen
  SyncAll()
  Par_1 = Dig_ReadLatch1(module)'Zwischenregister Modul 1 lesen
  Par_2 = Dig_ReadLatch1(module2)'Zwischenregister Modul 2 lesen
```

Dig_ReadLatch2

Dig_ReadLatch2 liefert die oberen 16 Bit (Bit 16…Bit 31) aus dem Zwischenregister für die digitalen Eingänge des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Dig_ReadLatch2(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **DigProg2** als Eingänge zu programmieren.

Sie können den aktuellen Zustand der digitalen Eingänge mit folgenden Anweisungen in das Zwischenregister übernehmen:

- **Digin_Word1**
- **Digin_Word2**
- **SyncAll** (falls für das Modul aktiviert).

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32, ExtLch_Enable

DigProg1, DigProg2Digin_Long_F, Digin_Word1, Digin_Word2

SyncAll, SyncEnable

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module1 3
#Define module2 5
Dim value As Long

Init:
  Rem DIO31:16 der Module 1+2 als Eingänge setzen
  DigProg2(module1,0000h)
  DigProg2(module2,0000h)
  SyncEnable(module1,dio,1)'Synchr. Digital-Modul 1 freigeben
  SyncEnable(module2,dio,1)'Synchr. Digital-Modul 2 freigeben

Event:
  Rem Pegel an den digitalen Eingängen von beiden Modulen synchron
  Rem in die Zwischenregister übernehmen
  SyncAll()
  Par_1 = Dig_ReadLatch2(module1)'Zwischenregister Modul 1 lesen
  Par_2 = Dig_ReadLatch2(module2)'Zwischenregister Modul 2 lesen
```

Dig_WriteLatch1

Dig_WriteLatch1 schreibt einen Wert in die unteren 16 Bit (Bit 0…15) des Zwischenregisters für die digitalen Ausgänge des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Dig_WriteLatch1(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster. Jedes Bit entspricht einem digitalen Ausgang (siehe Tabelle).	LONG

Bitnr.	31:16	15	14	13	…	2	1	0
Ausgang	–	15	14	13	…	2	1	0

Bemerkungen

Bei den Modulen Pro-DIO-32 und Pro-DIO-32 Rev. B müssen die angesprochenen Leitungen zunächst mit den Anweisungen **DigProg1** als Ausgänge programmiert werden.

Sie können den Wert des Zwischenregisters für die digitalen Ausgänge mit folgenden Anweisungen setzen:

- **Digout**
- **Digout_Word1**
- **Digout_Word2**
- **Dig_WriteLatch1**
- **Dig_WriteLatch2**
- **Dig_WriteLatch32**

Diese Anweisung darf nicht in Kombination mit **Dig_WriteLatch2** verwendet werden. Falls Sie Bits sowohl im unteren als auch im oberen Wort des Zwischenregisters verändern wollen, benutzen Sie bitte den Befehl **Dig_WriteLatch32**.

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch2, Dig_WriteLatch32, ExtLch_Enable

DigProg1, DigProg2, Digout, Digout_Word1, Digout_Word2

Get_Digout_Word1, Get_Digout_Word2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B, REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define adc_module 2
#Define d_module1 3
#Define d_module2 5
Dim value As Long

Init:
  Rem Nur DIO-32: DIO15:00 der Module als Ausgänge setzen
  DigProg1(d_module1,0FFFFh)
  DigProg1(dio_module2,0FFFFh)

  SyncEnable(d_module1,dio,1)'Synchr. Digital-Modul 1 freigeben
  SyncEnable(d_module2,dio,1)'Synchr. Digital-Modul 2 freigeben
  Dig_WriteLatch1(d_module1,1)'Unterstes Bit im Ausgangs-
                              'Zwischenregister setzen
  Dig_WriteLatch1(d_module2,1)'Unterstes Bit im Ausgangs-
                              'Zwischenregister setzen

Event:
  value = ADC(adc_module,1)'Messwerterfassung
  If (value > 3000) Then    'Grenzwert überschritten?
    SyncAll()               'Werte aus den Zwischenregistern
                            'ausgeben
  EndIf
```

Dig_WriteLatch2

Dig_WriteLatch2 schreibt einen Wert in die oberen 16 Bit (Bit 16…31) des Zwischenregisters für die digitalen Ausgänge des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Dig_WriteLatch2(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster. Jedes Bit entspricht einem digitalen Ausgang (siehe Tabelle).	LONG

Bitnr.	31:16	15	14	13	…	2	1	0
Ausgang	–	31	30	29	…	18	17	16

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **DigProg2** als Ausgänge programmiert werden.

Sie können den Wert des Zwischenregisters für die digitalen Ausgänge mit folgenden Anweisungen setzen:

- **Digout**
- **Digout_Word1**
- **Digout_Word2**
- **Dig_WriteLatch1**
- **Dig_WriteLatch2**
- **Dig_WriteLatch32**

Diese Anweisung darf nicht in Kombination mit **Dig_WriteLatch1** verwendet werden. Falls Sie Bits sowohl im unteren als auch im oberen Wort des Zwischenregisters verändern wollen, benutzen Sie bitte den Befehl **Dig_WriteLatch32**.

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch32, ExtLch_Enable

DigProg1, DigProg2, Digout, Digout_Word1, Digout_Word2

Get_Digout_Word1, Get_Digout_Word2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define adc_module 2
#Define d_module1 3
#Define d_module2 5
Dim value As Long

Init:
  Rem Nur DIO-32: DIO31:16 der Module als Ausgänge setzen
  DigProg2(d_module1,0FFFFh)
  DigProg2(d_module2,0FFFFh)

  SyncEnable(d_module1,dio,1)'Synchr. Digital-Modul 1 freigeben
  SyncEnable(d_module2,dio,1)'Synchr. Digital-Modul 2 freigeben
  Dig_WriteLatch2(d_module1,1)'Bit 16 im Ausgangs-Latch setzen
  Dig_WriteLatch2(d_module2,16)'Bit 20 im Ausgangs-Latch setzen

Event:
  value = ADC(adc_module,1)'Messwerterfassung
  If (value > 3000) Then   'Grenzwert überschritten?
    SyncAll()              'Werte aus den Zwischenregistern
                           'ausgeben
  EndIf
```

Dig_WriteLatch32

Dig_WriteLatch32 schreibt einen 32 Bit-Wert in das Langwort (Bits 31…00) des Latches auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
Dig_WriteLatch32(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster. Jedes Bit entspricht einem digitalen Ausgang (siehe Tabelle).	LONG

Bitnr.	31	30	29	…	2	1	0
Ausgang	31	30	29	…	2	1	0

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit den Anweisungen **DigProg1** und **DigProg2** als Ausgänge programmiert werden.

Sie können den Wert des Zwischenregisters für die digitalen Ausgänge mit folgenden Anweisungen setzen:

- **Digout**
- **Digout_Word1**
- **Digout_Word2**
- **Dig_WriteLatch1**
- **Dig_WriteLatch2**
- **Dig_WriteLatch32**

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, ExtLch_Enable

DigProg1, DigProg2, Digout, Digout_Word1, Digout_Word2

Get_Digout_Word1, Get_Digout_Word2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  DigProg1(module,0FFFFh)   'DIO15:00 (Low-Word) des DIO-32-
                            'Moduls als Ausgang
  DigProg2(module,0FFFFh)   'DIO31:16 (High-Word) des DIO-32-
                            'Moduls als Ausgang

Event:
  Dig_Latch(module)         'Informationen des Ausgangs-Latches
                            'auf einem DIO-32-Modul ausgeben
  Dig_WriteLatch32(module,Par_1)'Long-Word ins Ausgangs-Latch
                            'schreiben
```

Digin_Long_F

Digin_Long_F gibt den Zustand der Eingänge (Bits 31…00) des angegebenen Moduls als Bitmuster zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Digin_Long_F(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Bitmuster. Jedes Bit (31…0) entspricht dem Eingangszustand eines digitalen Eingangs (siehe Tabelle). Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an.	LONG

Bitnr.	31	30	29	…	2	1	0
Eingang	31	30	29	…	2	1	0

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit den Anweisungen **DigProg1** und **DigProg2** als Eingänge zu programmieren.

Wenn ein oder mehrere der Kanäle 0…31 als Ausgang programmiert sind, sind die zugehörigen Bits nicht definiert.

Siehe auch

Dig_Latch, Digin_Word1, Digin_Word2, DigProg1, DigProg2, Digout_Long_F, Digout_Word1, Digout_Word2

Gültig für

DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  DigProg1(module,0000h)    'DIO15:00 (Low-Word) als Eingang
  DigProg2(module,0000h)    'DIO31:16 (High-Word) als Eingang

Event:
  Par_1 = Digin_Long_F(module)'Alle Eingänge einlesen
```

Digin_Word1

Digin_Word1 gibt den Zustand der Eingänge 0…15 des angegebenen Moduls als Bitmuster zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Digin_Word1(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Bitmuster. Jedes der Bits 15…0 entspricht dem Eingangszustand eines digitalen Eingangs (siehe Tabelle). Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an.	LONG

Bitnr.	31:16	15	14	…	2	1	0
Eingang	–	15	14	…	2	1	0

Bemerkungen

Wir empfehlen, bei den Modulen Pro-DIO-32 und Pro-DIO-32 Rev. B die angesprochenen Leitungen zunächst mit der Anweisung **DigProg1** als Eingänge zu programmieren.

Wenn ein oder mehrere der Kanäle 0…15 als Ausgang programmiert sind, sind die zugehörigen Bits nicht definiert.

Siehe auch

Digin_Word2, Digout, Digout_Word1, Digout_Word2, DigProg1, DigProg2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B, OPT-16 Rev. A, OPT-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define adc_module 1
#Define dio_module 4
Dim Data_1[10000] As Long As FIFO

Init:
  Rem nur für DIO32: DIO15:00 (Low-Word) als Eingang setzen
  DigProg1(dio_module,0000h)

Event:
  Rem Abfrage, ob Eingänge 1, 2 und 3 des Moduls 4 gesetzt sind
  If (Digin_Word1(dio_module) And 14 = 14) Then
    Data_1 = ADC(adc_module,1) 'Messwerterfassung
  EndIf
```

Am Bitmuster des Dezimalwerts 14 (nämlich …01110b) können Sie erkennen, welche digitalen Eingänge gesetzt sind, hier die Eingänge 1, 2 und 3.

Sie können in *ADbasic* Zahlen auch in binärer Schreibweise eingeben. Fügen Sie hierzu an das Bitmuster den Buchstaben b an. Die Zeile im obigen Beispiel würde dann lauten:

```
If (Digin_Word1(dio_module) And 1110b = 1110b) Then
  Data_1 = ADC(adc_module,1) 'Messwerterfassung
EndIf
```

Digin_Word2

Digin_Word2 gibt den Zustand der Eingänge 16…31 des angegebenen Moduls als Bitmuster zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Digin_Word2(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Bitmuster. Jedes der Bits 15…0 entspricht dem Eingangszustand eines digitalen Eingangs (siehe Tabelle). Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an.	LONG

Bitnr.	31:16	15	14	…	2	1	0
Eingang	–	31	30	…	18	17	16

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **DigProg2** als Eingänge zu programmieren.

Wenn ein oder mehrere der Kanäle 16…31 als Ausgang programmiert sind, sind die zugehörigen Bits nicht definiert.

Siehe auch

Digin_Word1, Digout, Digout_Word1, Digout_Word2, DigProg1, DigProg2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define adc_module 1
#Define dio_module 4
#Define module 1
Dim Data_1[10000] As Long As FIFO

Init:
  DigProg2(dio_module,0000h)'DIO31:16 (High-Word) als Eingang

Event:
  Rem Abfrage, ob Eingänge 16, 18 und 21 auf Modul 4 gesetzt sind
  If (Digin_Word2(dio_module) And 39 = 39) Then
    Data_1 = ADC(adc_module,1)'Messwerterfassung
  EndIf
```

Am Bitmuster des Dezimalwerts `39` (nämlich …`0100101b`) können Sie erkennen, welche digitalen Eingänge gesetzt sind, hier die Eingänge 16, 18 und 21.

Sie können in *ADbasic* Zahlen auch in binärer Schreibweise eingeben. Fügen Sie hierzu an das Bitmuster den Buchstaben `b` an. Die Zeile im obigen Beispiel würde dann lauten:

```
If (Digin_Word2(dio_module) And 0100101b = 0100101b) Then
  Data_1 = ADC(adc_module,1)'Messwerterfassung
EndIf
```

Digout

Digout setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.

Syntax

```
#Include ADwinPro_All.Inc
Digout(module,output,value)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
output	Nummer des Ausgangs, der angesprochen werden soll (0…31).	LONG
value	Neuer Zustand für den gewählten Ausgang: 0: Pegel Low. 1: Pegel High.	LONG

Bemerkungen

Für das Modul Pro-DIO-32 Rev. B ist **Digout** anwendbar, wir empfehlen aber die Verwendung der Anweisung **Digout_F**, die schneller ist und erheblich weniger Programmspeicher benötigt.

Der angesprochene Kanal muss zuerst mit der Anweisung **DigProg1** oder **DigProg2** als Ausgang programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Die Anweisung **Digout** besteht aus einer Sequenz von Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.

Get_Digout_ Word1/2	→	…	→	Digout_ Word1/2
Aktuellen Status aller digitalen Ausgänge einlesen		Gewählte Bit-Position setzen oder rücksetzen		Manipulierten Wert zurückschreiben

Die Anweisung darf in zwei parallel laufenden Prozessen mit unterschiedlicher Priorität nicht auf dasselbe Modul angewendet werden:

Ein niedrig priorisierter Prozess kann inmitten der **Digout** Sequenz von einem höher priorisierten Prozess unterbrochen werden. Wenn der höher priorisierte Prozess während dieser Unterbrechung die Stellung der digitalen Ausgänge ändert, geht diese Änderung verloren, wenn der niederpriore Prozess den manipulierten Wert mit **Digout_Word** zurückschreibt.

Mehrere parallel laufende hoch priorisierte Prozesse können die Anweisung **Digout** problemlos verwenden, da sich hoch priorisierte Prozesse nicht gegenseitig unterbrechen.

Siehe auch

DigProg1, DigProg2

Digout_F, Digout_Bits_F, Digout_Word1, Digout_Word2, Get_Digout_Word1, Get_Digout_Word2

Gültig für

DIO-32 Rev. B, REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim value As Long

Init:
  Rem nur für DIO32: Kanäle als Ausgang setzen
  DigProg1(module,0FFFFh)   'DIO15:00 (Low-Word) als Ausgang
  DigProg2(module,0FFFFh)   'DIO31:16 (High-Word) als Ausgang

Event:
  value = ADC(module,1)     'Messwerterfassung
  If (value < 100) Then     'Grenzwert unterschritten
    Digout(module,2,0)      'Dig. Ausgang 2 zurücksetzen
  EndIf
```

Digout_Bits_F

Digout_Bits_F setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel High oder den Pegel Low.

Syntax

```
#Include ADwinPro_All.Inc
Digout_Bits_F(module,set,clear)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
set	Bitmuster, mit dem einzelne digitale Ausgänge auf den Pegel High gesetzt werden. Bit = 0: Ausgang unverändert lassen. Bit = 1: Ausgang auf Pegel High setzen.	LONG
clear	Bitmuster, mit dem einzelne digitale Ausgänge auf den Pegel Low gesetzt werden. Bit = 0: Ausgang unverändert lassen. Bit = 1: Ausgang auf Pegel Low setzen.	LONG

Bitnr.	31	30	…	2	1	0
Ausgang	31	30	…	2	1	0

Bemerkungen

Die angesprochenen Kanäle müssen zunächst mit den Anweisungen **DigProg1** und **DigProg2** als Ausgänge programmiert werden. **Digout_Bits_F** beeinflusst keine Kanäle, die als Eingang programmiert sind.

Digout_Bits_F kann beliebige Ausgänge löschen oder setzen, ohne den Zustand der anderen Ausgänge zu ändern. Die logische Verknüpfung der entsprechenden Register erfolgt bei diesem Befehl auf Hardware-Ebene und läuft damit wesentlich schneller ab als beim Befehl **Digout**, der auf Software-Ebene arbeitet.

Zur Klarheit weisen wir darauf hin, dass Sie die im Bitmuster `set` gesetzten Bits nicht gleichzeitig im Bitmuster `clear` setzen dürfen und umgekehrt.

Siehe auch

Digout, Digout_F, Digout_Word1, Digout_Word2, DigProg1, DigProg2

Gültig für

DIO-32 Rev. B, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  Rem nur für DIO32: Kanäle als Ausgang setzen
  DigProg1(module,0FFFFh)   'DIO15:00 (Low-Word) als Ausgang
  DigProg2(module,0FFFFh)   'DIO31:16 (High-Word) als Ausgang

Event:
  If (Par_1 = 1) Then       'Bedingung abfragen
    Rem unteres Wort: MSB setzen, alle anderen Bits löschen
    Digout_Bits_F(module,8080h,7F7Fh)
  Else
    Rem unteres Wort: ungerade Bits setzen, gerade Bits löschen
    Digout_Bits_F(module,5555h,0AAAAh)
  EndIf
```

Digout_F

Digout_F setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.

Syntax

```
#Include ADwinPro_All.Inc
Digout_F(module,output,value)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
output	Nummer des Ausgangs, der angesprochen werden soll (0…31).	LONG
value	Neuer Zustand für den gewählten Ausgang: 0: Pegel Low. 1: Pegel High.	LONG

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit den Anweisungen **DigProg1** und **DigProg2** als Ausgänge programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Die logische Verknüpfung der entsprechenden Register erfolgt bei diesem Befehl auf Hardware-Ebene und läuft damit wesentlich schneller ab als beim Befehl **Digout**, der auf Software-Ebene arbeitet.

Siehe auch

Digout, Digout_Bits_F, Digout_Word1, Digout_Word2, DigProg1, DigProg2

Gültig für

DIO-32 Rev. B, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  Rem nur für DIO32: Kanäle als Ausgang setzen
  DigProg1(module,0FFFFh)   'DIO15:00 (Low-Word) als Ausgang
  DigProg2(module,0FFFFh)   'DIO31:16 (High-Word) als Ausgang

Event:
  Rem Low-Word (Bits 0…15) einlesen und prüfen, ob das MSB
  Rem gesetzt ist
  If (Digin_Word1(module) And 8000h = 8000h) Then
    Digout_F(module,31,0)   'Falls MSB gesetzt, Bit 31 löschen
  Else
    Digout_F(module,31,1)   'Falls MSB gelöscht, Bit 31 setzen
  EndIf
```

Digout_Long_F

Digout_Long_F setzt oder löscht durch den übergebenen 32 Bit-Wert alle Ausgänge des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Digout_Long_F(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster, nach dem die digitalen Ausgänge gesetzt werden: Bit = 0: Ausgang auf Pegel Low setzen. Bit = 1: Ausgang auf Pegel High setzen.	LONG

Bitnr.	31	30	…	2	1	0
Ausgang	31	30	…	2	1	0

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit den Anweisungen **DigProg1** und **DigProg2** als Ausgänge programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Siehe auch

Digout, Digout_F, Digout_Bits_F, Digout_Word1, Digout_Word2, DigProg1, DigProg2

Gültig für

DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  DigProg1(module,0FFFFh)   'DIO15:00 (Low-Word) als Ausgang
  DigProg2(module,0FFFFh)   'DIO31:16 (High-Word) als Ausgang

Event:
  Digout_Long_F(module,1000000)'Den Wert 1 Mio. als Binärwert
                             'auf die DIOs ausgeben
```

Digout_Word1

Digout_Word1 setzt gleichzeitig die digitalen Ausgänge 0…15 des angegebenen Moduls auf einen bestimmten Pegel.

Syntax

```
#Include ADwinPro_All.Inc
Digout_Word1(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster, nach dem die digitalen Ausgänge gesetzt werden: Bit = 0: Ausgang auf Pegel Low setzen. Bit = 1: Ausgang auf Pegel High setzen.	LONG

Bitnr.	31…16	15	…	1	0
Ausgang	–	15	…	1	0

Bemerkungen

Bei den Modulen Pro-DIO-32 und Pro-DIO-32 Rev. B müssen die angesprochenen Leitungen zunächst mit der Anweisung **DigProg1** als Ausgänge programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32

DigProg1, DigProg2, Digin_Word1, Digin_Word2, Digout, Digout_Word2

Digin_Long_F, Digout_Bits_F, Digout_F, Digout_Long_F

Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B, REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define adc_module 1
#Define dio_module 4
Dim value As Long

Init:
  Rem nur für DIO32: Kanäle als Ausgang setzen
  DigProg1(dio_module,0FFFFh)'DIO15:00 (Low-Word) als Ausgang


Event:
  value = ADC(adc_module,1)'Messwerterfassung
  If (value > 3000) Then    'Grenzwert überschritten?
    Digout_Word1(dio_module,111b)'Ausgänge 0, 1 u. 2 des
                            'DIO-Moduls setzen, alle anderen
                            'Ausgänge werden zurückgesetzt
  EndIf
```

Digout_Word2

Digout_Word2 setzt gleichzeitig die digitalen Ausgänge 16…31 des angegeben Moduls auf einen bestimmten Pegel.

Syntax

```
#Include ADwinPro_All.Inc
Digout_Word2(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster, nach dem die digitalen Ausgänge gesetzt werden: Bit = 0: Ausgang auf Pegel Low setzen. Bit = 1: Ausgang auf Pegel High setzen.	LONG

Bitnr.	31…16	15	…	1	0
Ausgang	–	31	…	17	16

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **DigProg2** als Ausgänge programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32

Digin_Word1, Digin_Word2, Digout, Digout_Word1, Digout_Word2

DigProg1, DigProg2, Digin_Long_F, Digout_Bits_F, Digout_F, Digout_Long_F

Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Siehe auch

Digin_Word2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define adc_module 1
#Define dio_module 4
Dim value As Long

Init:
  DigProg2(dio_module,0FFFFh)'DIO31:16 (High-Word) als Ausgang

Event:
  value = ADC(adc_module,1)'Messwerterfassung
  If (value > 3000) Then    'Grenzwert überschritten?
    Digout_Word2(dio_module,1011b)'Ausgänge 16, 17 u. 19 setzen,
                            'alle anderen Ausgänge werden
                            'zurückgesetzt
  EndIf
```

DigProg1

DigProg1 programmiert die digitalen Kanäle 0…15 des angegebenen Moduls als Ein- oder Ausgang.

Syntax

```
#Include ADwinPro_All.Inc
DigProg1(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster, nach dem die Kanäle als Ein- oder Ausgang gesetzt werden: Bit = 0: Kanal als Eingang setzen. Bit = 1: Kanal als Ausgang setzen.	LONG

Modul	Bitnr.	31…16	15	…	8	7	…	0
DIO-32 Rev. A	Kanalnr.	–	15	…	08	07	…	00
DIO-32 Rev. B	Kanalnr.	–	–	–	15:08	–	–	07:00

Bemerkungen

Nach dem Einschalten des Systems sind alle Kanäle als Eingänge konfiguriert.

Beachten Sie die unterschiedliche Programmierung der Module:

- Pro-DIO-32 Rev. A: Jeder einzelne Kanal kann als Eingang oder Ausgang gesetzt werden.
- Pro-DIO-32 Rev. B: Die Kanäle können nur in Gruppen zu je 8 als Ein- oder Ausgang gesetzt werden (nur 2 relevante Bits, die anderen Bits werden ignoriert).

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32

Digin_Word1, Digin_Word2, Digout, Digout_Word1, Digout_Word2

DigProg2, Digin_Long_F, Digout_Bits_F, Digout_F, Digout_Long_F

Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  DigProg1(module, 11111111b)'Konfiguriert Kanäle 0…7 des DIO-
                             'Moduls Nr. 1 als Ausgänge und Kanäle
                             '8…15 als Eingänge
```

DigProg2

DigProg2 programmiert die Kanäle 16…31 des angegebenen Moduls als Ein- oder Ausgang.

Syntax

```
#Include ADwinPro_All.Inc
DigProg2(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Bitmuster, nach dem die Kanäle als Ein- oder Ausgang gesetzt werden: Bit = 0: Kanal als Eingang setzen. Bit = 1: Kanal als Ausgang setzen.	LONG

Modul	Bitnr.	31…16	15	…	8	7	…	0
DIO-32 Rev. A	Kanalnr.	–	31	…	24	23	…	16
DIO-32 Rev. B	Kanalnr.	–	–	–	31:24	–	–	23:16

Bemerkungen

Nach dem Einschalten des Systems sind alle Kanäle als Eingänge konfiguriert.

Beachten Sie die unterschiedliche Programmierung der Module:

- Pro-DIO-32 Rev. A: Jeder einzelne Kanal kann als Eingang oder Ausgang gesetzt werden.
- Pro-DIO-32 Rev. B: Die Kanäle können nur in Gruppen zu je 8 als Ein- oder Ausgang gesetzt werden (nur 2 relevante Bits, die anderen Bits werden ignoriert).

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32

Digin_Word1, Digin_Word2, Digout, Digout_Word1, Digout_Word2

DigProg1, Digin_Long_F, Digout_Bits_F, Digout_F, Digout_Long_F

Get_Digout_Long, Get_Digout_Word1, Get_Digout_Word2

Gültig für

DIO-32 Rev. A, DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  DigProg2(module, 11111111b)'Konfiguriert Kanäle 16…23 des
                             'DIO-Moduls Nr. 1 als Ausgänge und
                             'Kanäle 24…31 als Eingänge
```

ExtLch_Enable

ExtLch_Enable gibt alle Latch-Eingänge auf dem angegebenen Modul entweder frei oder sperrt sie. Die Latch-Eingänge werden über die Nummer des zugehörigen Zählers ausgewählt.

Syntax

```
#Include ADwinPro_All.Inc
ExtLch_Enable(module,cnt_select)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
cnt_select	Bitmuster zur Auswahl der Zähler und zur Einstellung des Latch-Zustands: Bit = 0: Latch-Eingang sperren. Bit = 1: Latch-Eingang freigeben.	LONG

Bitnr.	31:5	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

- / -

Siehe auch

Cnt_Clear, Cnt_Enable, Cnt_Latch, Cnt_Read32, Cnt_ReadLatch32, Cnt_SetMode

Gültig für

CNT-VR4L(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  Rem Latch-Eingänge der Zähler 1+2 freigeben,
  Rem Latch-Eingänge der Zähler 3+4 sperren
  ExtLch_Enable(module, 0011b)
```

Get_Digout_Long

Get_Digout_Long gibt den Inhalt des Ausgangs-Latches (Register für digitale Ausgänge) auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Get_Digout_Long(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Inhalt des Ausgangs-Latches (Bits 31:00).	LONG

Bemerkungen

Ein Rücklesen der aktuellen Zustände der Ausgänge anstatt des Ausgangs-Latches ist technisch nicht möglich.

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32, ExtLch_Enable

DigProg1, DigProg2, Digin_Word1, Digin_Word2, Digout, Digout_Word1, Digout_Word2

Digin_Long_F, Digout_Bits_F, Digout_F, Digout_Long_F

Get_Digout_Word1, Get_Digout_Word2

Gültig für

DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Event:
  REM Bits 31:00 aus dem Latch zurücklesen
  Par_1 = Get_Digout_Long(module)
```

Get_Digout_Word1

Get_Digout_Word1 gibt das untere Wort (Bit 0…15) des Ausgangs-Latches (Register für digitale Ausgänge) des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Get_Digout_Word1(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Inhalt des Ausgangs-Latches (Bits 15:00).	LONG

Bemerkungen

Ein Rücklesen der aktuellen Zustände der Ausgänge anstatt des Ausgangs-Latches ist technisch nicht möglich.

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32, ExtLch_Enable

DigProg1, DigProg2, Digin_Word1, Digin_Word2, Digout, Digout_Word1, Digout_Word2

Digin_Long_F, Digout_Bits_F, Digout_F, Digout_Long_F

Get_Digout_Long, Get_Digout_Word2

Gültig für

DIO-32 Rev. B, REL-16 Rev. A, REL-16 Rev. B, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim value As Long

Init:
  value = Get_Digout_Word1(module) 'aktuellen Wert des Ausgangs-
                                   'registers lesen
  value = value And 0FFFEh 'letztes Bit (Bit 0) auf 0 setzen
  Digout_Word1(module, value) 'geänderten Wert zurückschreiben
```

Get_Digout_Word2

Get_Digout_Word2 gibt das obere Wort (Bit 16…31) des Ausgangs-Latches (Register für digitale Ausgänge) des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Get_Digout_Word2(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Inhalt des Ausgangs-Latches (Bits 31:16).	LONG

Bemerkungen

Ein Rücklesen der aktuellen Zustände der Ausgänge anstatt des Ausgangs-Latches ist technisch nicht möglich.

Siehe auch

Dig_Latch, Dig_ReadLatch1, Dig_ReadLatch2, Dig_WriteLatch1, Dig_WriteLatch2, Dig_WriteLatch32, ExtLch_Enable

DigProg1, DigProg2, Digin_Word1, Digin_Word2, Digout, Digout_Word1, Digout_Word2

Digin_Long_F, Digout_Bits_F, Digout_F, Digout_Long_F

Get_Digout_Long, Get_Digout_Word1

Gültig für

DIO-32 Rev. B

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim value As Long

Init:
  value = Get_Digout_Word2(module)'aktuellen Wert der
                                  'Ausgangsregister lesen
  value = value And 0FFFDh 'vorletztes Bit (Bit 17) auf 0 setzen
  Digout_Word2(module, value)'geänderten Wert zurückschreiben
```

PWM_Enable

PWM_Enable gibt alle internen Zähler frei oder stoppt sie. Die Zähler werden über die Nummer des zugehörigen PWM-Ausgangs gewählt.

Syntax

```
#Include ADwinPro_All.Inc
PWM_Enable(module,output)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
output	Bitmuster zur Auswahl der PWM-Ausgänge und zur Einstellung des Zähler-Zustands: Bit = 0: Zähler sperren. Bit = 1: Zähler freigeben.	LONG

Bitnr.	31:5	3	2	1	0
PWM-Kanal	–	4	3	2	1

Bemerkungen

Diese Anweisung ändert nicht den Pegel der PWM-Ausgänge.
Wenn Sie den Pegel der PWM-Ausgänge ändern möchten, tun Sie dies mit dem Befehl **PWM_Out** und stoppen anschließend mit **PWM_Enable** die zugehörigen internen Zähler.

Sobald und solange die Zähler gestoppt sind, können die Pegel der PWM-Ausgänge nicht geändert werden.

Siehe auch

PWM_Out, PWM_Set

Gültig für

PWM-4(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  PWM_Enable(module,1)      'PWM-Ausgang 1 für die Ausgabe
                            'aktivieren
  PWM_Set(module,1,0,2500,2500)'PWM-Signal für Ausgang 1 setzen
```

PWM_Out

`PWM_Out` setzt einen bestimmten PWM-Ausgabekanal des angegebenen Moduls auf den Pegel High oder Low.

Syntax

```
#Include ADwinPro_All.Inc
PWM_Out(module,channel,level)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer des PWM-Ausgabekanals (1…4).	LONG
level	Pegel, auf den der Kanal gesetzt werden soll: 0: auf Pegel Low setzen. 1: auf Pegel High setzen.	LONG

Bemerkungen

Dieser Befehl hat nur dann eine Funktion, wenn der zugehörige Zähler des gewählten Kanals freigegeben ist.

Siehe auch

PWM_Enable, PWM_Set

Gültig für

PWM-4(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  PWM_Enable(module,11b)    'Zähler der PWM-Ausgänge 1 und 2
                            'freigeben
  PWM_Set(module,1,0,2500,2500)'PWM-Signal für Ausgang 1 setzen
  PWM_Out(module,2,1)       'PWM-Ausgang 2 auf den logischen Wert
                            '1 setzen
```

PWM_Set

`PWM_Set` setzt die Voreinstellungen eines bestimmten PWM-Ausgabekanals auf dem angegebenen Modul.

Die Voreinstellungen sind:
- Faktor des Vorteilers (Prescaler)
- Zählwert der Low-Zeit
- Zählwert der High-Zeit

Syntax

```
#Include ADwinPro_All.Inc
PWM_Set(module,channel,prescale,low,high)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer des PWM-Ausgabekanals (1…4).	LONG
prescale	Exponent (0…7) für den Faktor des Vorteilers, siehe Formel.	LONG
low	Zählwert der Low-Zeit (1…32768), siehe Formel.	LONG
high	Zählwert der High-Zeit (1…32768), siehe Formel.	LONG

Bemerkungen

Die Ausgangsfrequenz (nach dem Vorteiler) wird nach folgender Formel berechnet:

$$f_{out} = \frac{5\,\mathrm{MHz}}{2^{prescale} \cdot (low + high)}$$

Die Zählwerte für Low- und High-Zeit stehen für die Anzahl der Impulse nach dem Vorteiler, die der interne Zähler erreichen muss, um den Logik-Pegel zu wechseln.

Die niedrigste Ausgangsfrequenz – bei noch einstellbarem Tastverhältnis von annähernd 0…100% – beträgt ca. 0,6 Hz.

Die höchste Ausgangsfrequenz, bei der das Tastverhältnis noch in 1%-Schritten einstellbar ist, beträgt ca. 50 kHz.

Siehe auch

PWM_Enable, PWM_Out

Gültig für

PWM-4(-I)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  PWM_Enable(module,11b)     'PWM-Ausgang 1 und 2 für die Ausgabe
                             'aktivieren
  PWM_Set(module,1,0,2500,2500)'PWM-Signal für Ausgang 1 setzen
  PWM_Out(module,2,1)        'PWM-Ausgang 2 auf den logischen Wert
                             '1 setzen
```

SSI_Mode

SSI_Mode stellt auf dem angegebenen Modul den Modus aller SSI-Decoder ein, entweder „single shot“ (einzelne lesen) und „continuous“ (kontinuierlich lesen).

Syntax

```
#Include ADwinPro_All.Inc
SSI_Mode(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
pattern	Betriebsmodus der SSI-Decoder, angegeben als Bitmuster. Jedem Decoder ist ein Bit zugeordnet (siehe Tabelle). Bit = 0: Modus „single shot“, der Encoder wird einmal ausgelesen. Bit = 1: Modus „continuous“, der Encoder wird kontinuierlich ausgelesen.	LONG

Bitnr.	31:2	1	0
SSI-Decoder	–	2	1

Bemerkungen

Wenn Sie den Modus „continuous“ wählen, startet das Auslesen des entsprechenden Encoders sofort. Die Anweisung **SSI_Start** ist hierzu nicht erforderlich.

Manche Encoder-Typen liefern im Modus „continuous“ gelegentlich den falschen Messwert 0 (Null) anstelle des korrekten Messwerts zurück. Im Modus „single shot“ tritt dieser Fehler nicht auf.

Siehe auch

SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Start, SSI_Status

Gültig für

CO4-D

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  SSI_Set_Clock(module,200) 'CLK (Taktrate) = 50 kHz
  SSI_Mode(module,11b)      'Continuous-Mode einstellen
                            '(für beide Encoder)
  SSI_Set_Bits(module,1,23) 'Anzahl Encoder-Bits = 23 (Encoder 1)
  SSI_Set_Bits(module,2,23) 'Anzahl Encoder-Bits = 23 (Encoder 2)

Event:
  Par_1 = SSI_Read(module,1) 'Positionswert (Encoder 1) auslesen
                             'und anzeigen
  Par_2 = SSI_Read(module,2) 'Positionswert (Encoder 2) auslesen
                             'und anzeigen
```

SSI_Read

SSI_Read gibt den zuletzt gespeicherten Zählerstand eines bestimmten SSI-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = SSI_Read(module,dcdr_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dcdr_no	Nummer (1, 2) des SSI-Decoders, dessen Zählerstand auszulesen ist.	LONG
ret_val	Letzter Zählerstand des SSI-Zählers (= Absolutwert-Position des Encoders).	LONG

Bemerkungen

Ein Encoder-Wert wird dann gespeichert, wenn die durch **SSI_SET_BITS** angegebene Anzahl von Bits eingelesen wurde.

Es wird immer diejenige Anzahl an Bits zurückgegeben, die mit der Anweisung **SSI_Set_Bits** eingestellt wurde, auch wenn dies nicht mit der Auflösung des Encoders übereinstimmt.
In diesem Fall ist der zurückgegebene Zählerstand abhängig vom Encoder (siehe Dokumentation des Herstellers). In der Regel gilt:

- Wenn der Encoder eine größere Auflösung besitzt, werden dessen überzählige niederwertigste Bits nicht genutzt.
- Besitzt der Encoder eine kleinere als die eingestellte Auflösung, wird für jedes fehlende höchstwertige Bit eine 0 (Null) gelesen.

Siehe auch

SSI_Mode, SSI_Set_Bits, SSI_Set_Clock, SSI_Start, SSI_Status

Gültig für

CO4-D

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim m, n, y As Long

Init:
  SSI_Set_Clock(module,50)  'CLK (Taktrate) = 200 kHz
  SSI_Mode(module,01b)      'Continuous-Mode setzen (Encoder 1)
  SSI_Set_Bits(module,1,23) 'Anzahl Encoder-Bits = 23 (Encoder 1)

Event:
  Par_1 = SSI_Read(module,1) 'Positionswert (Encoder 1) auslesen
                             'und anzeigen.
  Rem Falls es sich um einen Encoder mit Gray-Code handelt:
  m = 0                      'Werte der letzten Wandlung löschen
  y = 0                      ' -"-
  For n = 1 To 32            'Alle 32 mögl. Bits durchgehen
    m = (Shift_Right(Par_1,(32 - n)) And 1) XOr m
    y = (Shift_Left(m,(32 - n))) Or y
  Next n
  Par_9 = y                  'Das Ergebnis der Gray-/Binär-
                             'Wandlung in Par_9
```

SSI_Set_Bits

SSI_Set_Bits stellt für einen bestimmten SSI-Zähler auf dem angegebenen Modul die Anzahl der zu Bits ein, die einen vollständigen Encoder-Wert bilden.

Die Zahl der Bits sollte mit der Auflösung des Encoders identisch sein.

Syntax

```
#Include ADwinPro_All.Inc

SSI_Set_Bits(module,dcdr_no,bit_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dcdr_no	Nummer (1, 2) des SSI-Decoders, dessen Auflösung einzustellen ist.	LONG
bit_no	Anzahl der Bits (1…32) der zu lesenden Bits für einen Encoder-Wert (entspricht der Encoder-Auflösung).	LONG

Bemerkungen

Die Auflösung (Anzahl der Bits) des SSI-Encoders sollte mit der Anzahl der zu übertragenden Bits übereinstimmen.

Es wird immer diejenige Anzahl an Bits für einen Encoder-Wert erwartet, die mit **SSI_Set_Bits** eingestellt wurde, auch wenn dies nicht mit der Auflösung des Encoders übereinstimmt.
In diesem Fall ist der zurückgegebene Zählerstand abhängig vom Encoder (siehe Dokumentation des Herstellers). In der Regel gilt:

- Wenn der Encoder eine größere Auflösung besitzt, werden dessen überzählige niederwertigste Bits nicht genutzt.
- Besitzt der Encoder eine kleinere als die eingestellte Auflösung, wird für jedes fehlende höchstwertige Bit eine 0 (Null) gelesen.

Siehe auch

SSI_Mode, SSI_Read, SSI_Set_Clock, SSI_Start, SSI_Status

Gültig für

CO4-D

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  SSI_Set_Clock(module,50)  'CLK (Taktrate) = 200 kHz
  SSI_Mode(module,11b)      'Continuous-Mode einstellen (für
                            'beide Encoder)
  SSI_Set_Bits(module,1,10) '10 Encoder-Bits für Encoder 1
  SSI_Set_Bits(module,2,25) '25 Encoder-Bits für Encoder 2

Event:
  Par_1 = SSI_Read(module,1) 'Positionswert (Encoder 1) auslesen
                             'und anzeigen
  Par_2 = SSI_Read(module,2) 'Positionswert (Encoder 2) auslesen
                             'und anzeigen
```

SSI_Set_Clock

`SSI_Set_Clock` stellt die Taktrate (ca. 40 kHz bis 1 MHz) auf dem angegebenen Modul ein, mit der der Encoder getaktet wird.

Syntax

```
#Include ADwinPro_All.Inc
SSI_Set_Clock(module,prescale)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`prescale`	Teilerwert (10…255) zur Einstellung der Taktrate nach der Formel: Taktrate = 10 MHz / `prescale`.	LONG

Bemerkungen

Die Einstellung der Taktrate ist immer für beide Encoder, die an dem angesprochenen Modul angeschlossen sind, identisch und kann nicht getrennt eingestellt werden. Gegebenenfalls muss sich der Takt am langsameren Encoder orientieren.

Nach dem Einschalten des Moduls wird als Voreinstellung der Teilerfaktor 128 verwendet, das entspricht einer Taktrate von 78 kHz.

Teilerfaktoren kleiner 10 werden automatisch auf den Wert 10 korrigiert; von Werten über 255 werden die niederwertigsten 8 Bits als Teilerfaktor verwendet.

Die mögliche Taktfrequenz ist abhängig von Kabellänge, Kabeltyp und den jeweils verwendeten Sende- und Empfangsbausteinen des Encoders bzw. Decoders. Grundsätzlich gilt als Regel: Je höher die Taktfrequenz, desto kürzer die mögliche Kabellänge.

Siehe auch

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Start, SSI_Status

Gültig für

CO4-D

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  SSI_Set_Clock(module,10)  'CLK (Taktrate) = 1 MHz
  SSI_Mode(module,11b)      'Continuous-Mode setzen
                            '(für beide Encoder)
  SSI_Set_Bits(module,1,10) 'Anzahl Encoder-Bits = 10 (Encoder 1)
  SSI_Set_Bits(module,2,25) 'Anzahl Encoder-Bits = 25 (Encoder 2)

Event:
  Par_1 = SSI_Read(module,1) 'Positionswert (Encoder 1) auslesen
                             'und anzeigen
  Par_2 = SSI_Read(module,2) 'Positionswert (Encoder 2) auslesen
                             'und anzeigen
```

SSI_Start

SSI_Start startet auf dem angegebenen Modul das Auslesen eines oder beider SSI-Encoder (nur im Modus single shot).

Syntax

```
#Include ADwinPro_All.Inc
SSI_Start(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dcdr_no	Bitmuster zur Auswahl der SSI-Decoder, die gestartet werden sollen: Bit = 0: keine Funktion. Bit = 1: Auslesen des SSI-Decoders starten.	LONG

Bitnr.	31:2	1	0
SSI-Decoder	–	2	1

Bemerkungen

Im Modus „continuous“ ist die Anweisung ohne Funktion, weil dann die Encoder-Werte ohnehin kontinuierlich ausgelesen werden.

Ein Encoder-Wert wird dann gespeichert, wenn die durch **SSI_SET_BITS** angegebene Anzahl von Bits eingelesen wurde.

Es wird immer ein vollständiger Encoder-Wert übertragen, auch wenn währenddessen der Modus umgestellt wird.

Siehe auch

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Status

Gültig für

CO4-D

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  SSI_Set_Clock(module,250)'CLK (Taktrate) = ca. 40 kHz
  SSI_Mode(module,00b)     'Single shot-Mode einstellen
                           '(beide Zähler)
  SSI_Set_Bits(module,1,23)'Anzahl Encoder-Bits = 23 (Encoder 1)
  SSI_Set_Bits(module,2,23)'Anzahl Encoder-Bits = 23 (Encoder 2)

Event:
  SSI_Start(module,11b)    'Positionswert von Encoder 1 & 2 lesen
  Do                       'Für Encoder 1:
  Until (SSI_Status(module,1) = 0)
  Rem Wenn Positionswert komplett gelesen ist, dann …
  Par_1 = SSI_Read(module,1)'Positionswert auslesen und anzeigen
  Do                       'Für Encoder 2:
  Until (SSI_Status(module,2) = 0)
  Rem Wenn Positionswert komplett gelesen ist, dann …
  Par_1 = SSI_Read(module,2)'Positionswert auslesen und anzeigen
```

SSI_Status

`SSI_Status` liefert für einen bestimmten Decoder den aktuellen Lese-Status auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = SSI_Status(module,dcdr_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
dcdr_no	Nummer (1, 2) des SSI-Decoders, dessen Status gefragt ist.	LONG
ret_val	Lese-Status des Decoders: 0: Decoder ist bereit, d.h. ein vollständiger Wert wurde gelesen. 1: Decoder liest einen Encoder-Wert ein.	LONG

Bemerkungen

Verwenden Sie die Status-Abfrage nur im SSI-Modus „single shot“. Im Modus „continuous“ ist eine Status-Abfrage nicht sinnvoll.

Siehe auch

SSI_Mode, SSI_Read, SSI_Set_Bits, SSI_Set_Clock, SSI_Start

Gültig für

CO4-D

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1

Init:
  SSI_Set_Clock(module,250)'CLK (Taktrate) = ca. 40 kHz
  SSI_Mode(module,00b)     'Single shot-Mode einstellen
                           '(beide Zähler)
  SSI_Set_Bits(module,1,23)'Anzahl Encoder-Bits = 23 (Encoder 1)
  SSI_Set_Bits(module,2,23)'Anzahl Encoder-Bits = 23 (Encoder 2)

Event:
  SSI_Start(module,11b)    'Positionswert von Encoder 1 & 2 lesen
  Do                       'Für Encoder 1:
  Until (SSI_Status(module,1) = 0)
  Rem Wenn Positionswert komplett gelesen ist, dann …
  Par_1 = SSI_Read(module,1)'Positionswert auslesen und anzeigen
  Do                       'Für Encoder 2:
  Until (SSI_Status(module,2) = 0)
  Rem Wenn Positionswert komplett gelesen ist, dann …
  Par_1 = SSI_Read(module,2)'Positionswert auslesen und anzeigen
```

Comp_Digin_Word

Comp_Digin_Word gibt den aktuellen Status des Schwellenwert-Vergleichs für alle Kanäle auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Comp_Digin_Word(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Bitmuster, das den Status des Schwellenwert-Vergleichs der Kanäle wiedergibt (Zuordnung zu den Kanälen siehe Tabelle): Bit = 0: Messwert < unterer Schwellenwert. Bit = 1: Messwert > oberer Schwellenwert.	LONG

Bitnr.	31…16	15	14	…	1	0
Kanalnr.	–	16	15	…	2	1

Bemerkungen

Die Bewertung der Messwerte erfolgt über Schaltschwellen (Hysterese), die mit **Comp_Set** festgelegt werden. Der Status ist umso stabiler, je weiter die Schaltschwellen auseinander liegen.

Diese Anweisung kann z. B. verwendet werden, wenn Analogsignale im Modus single-ended an den Eingängen angelegt sind.

Siehe auch

Comp_Read, Comp_Reset, Comp_Fifo_Select, Comp_Set, Comp_Digin_Word_Diff, Comp_Fifo_Read

Gültig für

Comp-16 Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Rem Schwellenwerte des Kanals 3 auf +3V und +1V setzen (bei
  Rem Spannungsbereich -2V … +8,23V)
  Comp_Set(1,3,500,300)
  Rem Schwellenwerte des Kanals 4 auf +4V und +1,5V setzen (bei
  Rem Spannungsbereich -2V … +8,23V)
  Comp_Set(1,4,600,350)

Event:
  Rem Vergleichs-Status des Kanals 3 abfragen
  Par_1=(Comp_Digin_Word(1) And 00100b)
  Rem Vergleichs-Status des Kanals 4 abfragen
  Par_2=(Comp_Digin_Word(1) And 01000b)
```

Comp_Digin_Word_Diff

Comp_Digin_Word_Diff gibt den aktuellen Status des Schwellenwert-Vergleichs auf dem angegebenen Modul zurück.

Der Vergleich wird auf den Differenzwert von je 2 Kanälen (differentielles Signal) angewendet.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Comp_Digin_Word_Diff(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ret_val	Bitmuster, das den Status des Schwellenwert-Vergleichs der Differenzwerte wiedergibt (Zuordnung zu den Kanalpaaren siehe Tabelle): Bit = 0: Differenz < unterer Schwellenwert. Bit = 1: Differenz > oberer Schwellenwert.	LONG

Bitnr.	31…8	7	6	…	1	0
Kanalpaar	–	15,16	13,14	…	3,4	1,2

Bemerkungen

Die Bewertung der Differenzwerte erfolgt über Schaltschwellen (Hysterese), die mit **Comp_Set** festgelegt werden. Der Status ist umso stabiler, je weiter die Schaltschwellen auseinander liegen.

Die Kanalpaare sind in aufsteigender Reihenfolge (1/2, 3/4, …, 15/16) festgelegt, die Differenz wird berechnet als Wert1 - Wert2, Wert3 - Wert4, …, Wert15 - Wert16. Für den Vergleich werden jeweils die Schwellenwerte des kleineren Kanals verwendet (= ungerade Kanalnummer).

Siehe auch

Comp_Read, Comp_Reset, Comp_Fifo_Select, Comp_Set, Comp_Digin_Word, Comp_Fifo_Read

Gültig für

Comp-16 Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Comp_Set(1,3,500,300)     'Schwellenwerte des Kanals 3
                            'auf +3V und +1V setzen
                            '(bei Spannungsbereich -2V … +8,23V)
  Comp_Set(1,13,450,50)     'Schwellenwerte des Kanals 13
                            'auf +2,5V und -1,5V setzen
                            '(bei Spannungsbereich -2V … +8,23V)
Event:
  Rem Different. Vergleichs-Status des Kanalpaars 3/4 abfragen
  Par_1=(Comp_Digin_Word_Diff(1) And 00000010b)
  Rem Different. Vergleichs-Status des Kanalpaars 13/14 abfragen
  Par_2=(Comp_Digin_Word_Diff(1) And 01000000b)
```

Comp_Fifo_Read

Comp_Fifo_Read liest die letzten 2 x 1024 Messwerte eines Kanalpaars aus dem internen FIFO-Speicher des angegebenen Moduls und überträgt sie in 2 Felder.

Syntax

```
#Include ADwinPro_All.Inc
Comp_Fifo_Read(module,array1[],array2[])
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
array1[]	Zielfeld für 1024 Messwerte des Kanals mit der kleineren der beiden Kanalnummern.	ARRAY LONG
array2[]	Zielfeld für 1024 Messwerte des Kanals mit der größeren der beiden Kanalnummern.	ARRAY LONG

Bemerkungen

Im internen FIFO-Speicher des Moduls sind die jeweils letzten 1024 Messwerte von 2 Kanälen abgelegt. Sie wählen das Kanalpaar mit **Comp_Fifo_Select** aus.

Die Anweisung beschreibt in beiden Zielfeldern die Feldelemente arrayx[1]…arrayx[1024]. Beachten Sie bitte, dass die Zielfelder entsprechend groß dimensioniert sein müssen. Die übertragenen Messwerte sind Werte aus dem Bereich 0…1023.

Während diese Anweisung die Messdaten in die Zielfelder überträgt, werden keine neuen Messwerte in den Speicher geschrieben. Damit ist sichergestellt, dass ein zusammenhängendes Messwert-Paket übertragen wird.
Nach der Datenübertragung wird der Speicher automatisch wieder mit aktuellen Messwerten beschrieben.

Siehe auch

Comp_Read, Comp_Reset, Comp_Fifo_Select, Comp_Set, Comp_Digin_Word, Comp_Digin_Word_Diff

Gültig für

Comp-16 Rev. A

Beispiel

```
#Include ADwinPro_All.Inc
Dim array1[1024] As Long
Dim array2[1024] As Long

Init:
  Comp_Fifo_Select(1,2)      'Werte der Kanäle 5,6 im FIFO-Speicher
                             'ablegen

Event:
  Rem 1024 Messwerte für beide Kanäle abholen: Kanal 5 in array1,
  Rem Kanal 6 in array2
  Comp_Fifo_Read(1,array1,array2)
```

Comp_Fifo_Select

Comp_Fifo_Select legt das Kanalpaar fest, dessen Daten im internen FIFO-Speicher des angegebenen Moduls gespeichert werden.

Syntax

```
#Include ADwinPro_All.Inc
Comp_Fifo_Select(module,ch_pair)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
ch_pair	Zahl (0…7) zur Festlegung eines Kanalpaars; die Zuordnung zu den Kanälen siehe Tabelle.	LONG

ch_pair	7	…	1	0
Kanalnr.	15, 16	…	3, 4	1, 2

Bemerkungen

Es kann immer nur ein Kanalpaar gewählt werden, mehr oder weniger Kanäle können nicht festgelegt werden.

Im FIFO-Speicher werden die jeweils letzten 1024 Messwerte der beiden gewählten Kanäle abgelegt (= 2 × 1024 Werte). Sie können den FIFO-Speicher mit **Comp_Fifo_Read** auslesen.

Siehe auch

Comp_Read, Comp_Reset, Comp_Set, Comp_Digin_Word, Comp_Digin_Word_Diff, Comp_Fifo_Read

Gültig für

Comp-16 Rev. A

Beispiel

```
#Include ADwinPro_All.Inc
Dim array1[1024] As Long
Dim array2[1024] As Long

Init:
  Comp_Fifo_Select(1,2)     'Werte der Kanäle 5,6 im FIFO-Speicher
                            'ablegen

Event:
  Rem 1024 Messwerte für beide Kanäle abholen: Kanal 5 in array1,
  Rem Kanal 6 in array2
  Comp_Fifo_Read(1,array1,array2)
```

Comp_Read

Comp_Read gibt den aktuellen, minimalen oder maximalen Messwert eines bestimmten Kanals auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Comp_Read(module,channel,value)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Kanalnummer (1…16).	LONG
value	Auswahl des Messwerttyps: 0: Aktueller Messwert. 1: Minimaler Messwert. 2: Maximaler Messwert.	LONG
ret_val	Messwert (0…1023) am gewählten Kanal in Digits.	LONG

Bemerkungen

Der Rückgabewert 0 Digits entspricht dem minimalen Analogsignal U_{min}, der Wert 1023 Digits dem maximalen Analogsignal U_{max}. Zur Umrechnung von Digits in Volt gilt die Formel:

Umrechnung Digits in Volt

$$\text{Spannung} = \text{Digits} \cdot \frac{U_{max} - U_{min}}{1024} + U_{min}$$

Das Messen des Analogsignals wird durch diese Funktion nicht verzögert oder unterbrochen.

Der minimale und maximale Messwert bezieht sich auf die Messwerte im Zeitraum seit dem letzten Rücksetzen mit **Comp_Reset** (oder seit dem Einschalten des Systems).

Siehe auch

Comp_Reset, Comp_Fifo_Select, Comp_Set, Comp_Digin_Word, Comp_Digin_Word_Diff, Comp_Fifo_Read

Gültig für

Comp-16 Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Comp_Set(1,3,500,300)     'Schwellenwerte des Kanals 3
                            'auf +3V und +1V setzen
                            '(bei Spannungsbereich -2V … +8,23V)
  Comp_Reset(1,100b)        'Minimal- und Maximalwerte
                            'des Kanals 3 rücksetzen

Event:
  Par_1=Comp_Read(1,3,1)    'Minimum des Kanals 3 lesen
  Par_2=Comp_Read(1,3,2)    'Maximum des Kanals 3 lesen
```

Comp_Reset

`Comp_Reset` setzt auf dem angegebenen Modul die Messung des Minimal- und Maximalwerts für die ausgewählten Kanäle gleichzeitig zurück.

Syntax

```
#Include ADwinPro_All.Inc
Comp_Reset(module,pattern)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`channel_pattern`	Bitmuster zum Rücksetzen der Minimal- und Maximalwerte (Zuordnung zu Kanälen siehe Tabelle). Bit = 0: Minimal- und Maximalwerte beibehalten. Bit = 1: Minimal- und Maximalwerte rücksetzen.	LONG

Bitnr.	31…16	15	14	…	1	0
Kanalnr.	–	16	15	…	2	1

Bemerkungen

Beim Rücksetzen wird der Maximalwert auf den Wert 0, der Minimalwert auf den Wert 1023 gesetzt. Nach dem Rücksetzen wird die Messung der beiden Werte sofort weitergeführt.

Siehe auch

Comp_Read, Comp_Fifo_Select, Comp_Set, Comp_Digin_Word, Comp_Digin_Word_Diff, Comp_Fifo_Read

Gültig für

Comp-16 Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Rem Schwellenwerte der Kanäle 1, 3 und 4 auf +3V und +1V setzen
  Rem (bei Spannungsbereich -2V … +8,23V)
  Comp_Set(1,1,500,300)
  Comp_Set(1,3,500,300)
  Comp_Set(1,4,500,300)
  Comp_Reset(1,1101b)       'Minimal- und Maximalwerte
                            'der Kanäle 1,3,4 rücksetzen

Event:
  Par_1 = Comp_Read(1,1,1)  'Minimum des Kanals 1 lesen
  Par_2 = Comp_Read(1,1,2)  'Maximum des Kanals 1 lesen
  Par_3 = Comp_Read(1,3,1)  'Minimum des Kanals 3 lesen
  Par_4 = Comp_Read(1,4,2)  'Maximum des Kanals 4 lesen
```

Comp_Set

Comp_Set setzt den unteren und den oberen Schwellenwert für einen bestimmten Kanal des angegebenen Moduls fest.

Syntax

```
#Include ADwinPro_All.Inc
Comp_Set(module,channel,ValHigh,ValLow)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Kanalnummer (1…16).	LONG
ValHigh	oberer Schwellenwert (1…1023) des Kanals.	LONG
ValLow	unterer Schwellenwert (0…1022) des Kanals.	LONG

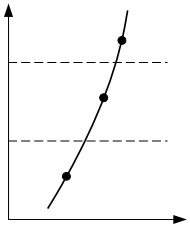
Bemerkungen

Mit der Festlegung der Schwellenwerte definieren Sie eine Hysterese bei der Bewertung des anliegenden Analogsignals.

Beispiel: Solange ein verrauschtes Analogsignal in der Nähe eines Schaltpunktes liegt, wechselt der Schaltzustand (ohne Hysterese) mehr oder weniger zufällig. Durch geschickte Wahl der Hysterese-Schwellenwerte kann dagegen der Schaltzustand stabilisiert werden.

Der obere Schwellenwert muss immer größer sein als der untere Schwellenwert, sonst funktioniert die Bewertung des Analogsignals nicht korrekt (Umrechnung Digits in Volt siehe **Comp_Read**).

Die anliegenden Analogsignale (auf allen Kanälen parallel) werden mit der festgelegten Messrate (20 MHz je Kanal) gemessen. Jeder einzelne Messwert wird nach folgendem Muster mit dem oberen und unteren Schwellenwert des zugehörigen Kanals verglichen:



Messwert > oberer Schwellenwert	⟶	Bit setzen
Messwert liegt auf oder zwischen den Schwellenwerten	⟶	Bit nicht ändern
Messwert < unterer Schwellenwert	⟶	Bit löschen

Die Vergleichs-Ergebnisse aller Kanäle werden in einem 16 Bit-Wort gespeichert (je Kanal 1 Bit); das aktuelle Vergleichs-Ergebnis kann mit **Comp_Digin_Word** ausgelesen werden.

In gleicher Weise wird der Differenzwert von je 2 Kanälen mit den Schwellenwerten verglichen und gespeichert (je Kanalpaar 1 Bit). Der Differenzwert wird berechnet als $Wert_1$ - $Wert_2$, $Wert_3$ - $Wert_4$, …, $Wert_{15}$ - $Wert_{16}$; es werden die Schwellenwerte des Kanals mit der ungeraden Nummer verwendet.
Das aktuelle Vergleichs-Ergebnis kann mit **Comp_Digin_Word_Diff** ausgelesen werden.

Siehe auch

Comp_Read, Comp_Reset, Comp_Fifo_Select, Comp_Digin_Word, Comp_Digin_Word_Diff, Comp_Fifo_Read

Gültig für

Comp-16 Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Comp_Set(1,3,500,300)      'Schwellenwerte des Kanals 3
                             'auf +3V und +1V setzen
                             '(bei Spannungsbereich -2V … +8,23V)
```

RTC_Set

RTC_Set setzt das Datum und die Zeit auf der Echtzeituhr des angegebenen Moduls. Ungültige Werte werden nicht akzeptiert.

Syntax

```
#Include ADwinPro_All.Inc
RTC_Set(module,year,month,day,hour,minute,second)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
year	Jahreszahl (0…63), entspricht 2000…2063.	LONG
month	Monat (1…12).	LONG
day	Tag (1…31); gültiger Wertebereich je nach Monat und Schaltjahr.	LONG
hour	Stunde (0…23).	LONG
minute	Minute (0…59).	LONG
second	Sekunde (0…59).	LONG

Bemerkungen

Die Jahresangabe bezieht sich auf den Zeitraum 2000 - 2063.

Siehe auch

RTC_Get

Gültig für

Storage Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Rem Echtzeituhr auf 4.7.2003 9:17:30 setzen
  RTC_Set(1, 3,7,4,9,17,30)'auf Modul 1
```

RTC_Get

RTC_Get gibt das Datum und die Zeit von der Echtzeituhr des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc
RTC_Get(module,year,month,day,hour,minute,second)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
year	Jahreszahl (0…63), entspricht 2000…2063.	LONG
month	Monat (1…12).	LONG
day	Tag (1…31); gültiger Wertebereich je nach Monat und Schaltjahr.	LONG
hour	Stunde (0…23).	LONG
minute	Minute (0…59).	LONG
second	Sekunde (0…59).	LONG

Bemerkungen

Alle Parameter (bis auf module) sind Rückgabeparameter.

Die Anweisung ersetzt die folgenden Befehle, die zeitweise verfügbar waren: **RTC_Get_Year**, **RTC_Get_Month**, **RTC_Get_Day**, **RTC_Get_Hour**, **RTC_Get_Minute**, **RTC_Get_Second**.

Siehe auch

RTC_Set

Gültig für

Storage Rev. A

Beispiel

```
#Include ADwinPro_All.Inc
Dim year,mon,day,h,m,s As Long

Init:
  Rem Echtzeituhr des Moduls 1 lesen
  RTC_Get(1, year,mon,day,h,m,s)
```

Media_WR_Blk_L

Media_WR_Blk_L kopiert eine Anzahl an Long-Datenblöcken aus einem Feld in eine der zehn Dateien auf dem Datenträger des angegebenen Moduls.

Der erste zu beschreibende Sektor der Datei ist frei wählbar.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Media_WR_Blk_L(module,f_info[],file,
          sec_idx,sec_cnt,array[],array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
f_info[]	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
file	Nummer (1…10) der Datei.	LONG
sec_idx	Index (1…m) des ersten zu beschreibenden Sektors, bezogen auf den Startsektor der Datei. Der Max.wert ist abhängig von der Dateigröße.	LONG
sec_cnt	Anzahl (1…12) der zu übertragenden Datenblöcke (=Sektoren) zu 128 Werten.	LONG
array[]	Feld, dessen Daten übertragen werden.	ARRAY LONG
array_idx	Index (1…n) des ersten zu übertragenden Feldelements.	LONG
ret_val	Status der Medienverwaltung als Bitmuster (siehe Tabelle).	LONG

Bitnr.	31:08	07	06	05	04	03	02	01	00
	–	A8	A7	A6	A5	A4	A3	A2	A1

Falls Bit = 1 (Bit = 0: ohne Bedeutung):
A1 Ein Medium ist im Schacht.
A2 Unbekannter Fehler aufgetreten.
A3 Medienverwaltung (Glue-Logik) ist beschäftigt.
A4 Daten können gelesen werden.
A5 Daten sind geschrieben.
A6 Reset wird gerade ausgeführt.
A7 Dateiende überschritten, es werden keine Daten übertragen.
A8 Time out, das Medium hat nicht reagiert.
Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Bevor diese Anweisung genutzt werden kann, muss das Feld f_info[] mit **Media_RD_FileInfo** initialisiert werden.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hoch-prioren Prozesses.

Der Start-Sektor ist frei wählbar. Dadurch können Daten an bereits gespeicherte Daten angehängt werden. Achten Sie darauf, nicht mehr Datenblöcke anzugeben als in die Datei passen. Anderenfalls werden die Datenblöcke nicht gespeichert!

Jeder Datenblock enthält 128 Long-Werte und wird in einem Sektor gespeichert. Wenn beispielsweise in einem ersten Schritt n Datenblöcke

ab dem Start-Sektor `x` gespeichert wurden, werden weitere Datenblöcke angehängt, indem sie ab Sektor `x+n+1` geschrieben werden.

Das Feld `array[]` muss mindestens `array_idx`+`s_cnt`×128 Werte enthalten.

Siehe auch

- / -

Gültig für

Storage Rev. A

Beispiel

```
Rem ########################################################
Rem Sinus-Tabelle mit 3600 Punkten in Daten-Datei speichern
Rem ########################################################
#Include ADwinPro_All.Inc

#Define pstom 1                 'Adresse des Pro-STORAGE-Modules
#Define f_info Data_197         'Feld mit Datei-Informationen
#Define LUT Data_1              'Look-Up-Tabelle für den Sinus
#Define file 1                  'Datei-Nummer zum Abspeichern
                                'des Sinus
#Define nds 3600                'Anzahl der Stützstellen
Rem Die Länge des Felds mit dem Sinus muss ein Vielfaches
Rem von 128 und damit größer oder gleich „nds" sein:
#Define lng 3712                'hier: 29x128
#Define pi2 6.2831853           'Zahlenwert für 2*pi

Dim f_info[22] As Long At DM_Local 'Dimensionierung des Felds
Dim LUT[lng] As Long            'Look-Up-Tabelle mit LONGs
Dim sec_p[128] As Long          'Sektor für Schreib-Pointer
Dim idx, n, ret As Long         'Lokale Variablen
Dim sec_s, sec_e, sec_c, sec_n As Long 'Sektor-Variablen
Dim sptr_a, sptr_b As Long      'Pointer-Variablen (Sektor-#)

Init:
  Par_52 = Media_RD_FileInfo(pstom,f_info)  'Datei-Info holen
  Rem Überprüfen, ob Medium gesteckt und bereit zum Lesen.
  Rem Sonst -> Exit
  If ((Par_52 And 1001b) <> 1001b) Then Exit

  sec_s = f_info[file*2 + 1]    'Erster und ...
  sec_e = f_info[file*2 + 2]    'letzter Sektor der Daten-Datei
  sec_c = 1                     'Aktueller (rel.) Sektor ist
                                '1. Sektor der Datei
  sec_n = Shift_Right(nds,7)    'Anz. kompl. Sektoren der
  If ((sec_n*128) < nds) Then 'Tabelle, ... plus ein Sektor,
    Inc sec_n                   'falls angefangen
  EndIf
  If ((sec_s+sec_n-1) > sec_e) Then 'Tab. größer als Datei?
    Exit                        ' dann Exit
  EndIf

  For idx = 1 To lng            'Werte der Feld-Elemente
                                'berechnen
    If (idx <= nds) Then        'Sinus-Tabellen berechnen:
      LUT[idx] = 32767.5 * Sin((idx-1) * pi2 / nds)
    Else                        'restl. Elemente ...
      LUT[idx] = 0              'mit 0 auffüllen
    EndIf
  Next idx

  'Alle Daten-Sektoren schreiben:
  For n=1 To sec_n
    Rem Alle notwend. Sekt. einzeln schreiben:
    ret = Media_WR_Blk_L(pstom,f_info,file,sec_c,1,LUT,
       (128*n - 127))
    Inc sec_c
  Next n
  Rem 1. Sektor, in dem WR-Ptr gespeichert wird
  sptr_a = f_info[1] + file - 1
  Rem 10. Sektor, in dem WR-Ptr gespeichert wird
  sptr_b = sptr_a + 10
  sec_p[1] = nds                '1. Long im Sektor = Pointer
  sec_p[2] = Not(nds)           '2. Long im Sektor = /Pointer
  For idx = 3 To 128            'Die restlichen 126 LONGs ...
```

```
  sec_p[idx] = 0              'mit 0 füllen
Next idx

Rem Beide Pointer-Sektoren schreiben:
ret = Media_WR_Blk_L(pstom,f_info,0,file,1,sec_p,1)
ret = Media_WR_Blk_L(pstom,f_info,0,file+10,1,sec_p,1)
```

Media_WR_Blk_F

Media_WR_Blk_F kopiert eine Anzahl an Float-Datenblöcken aus einem Feld in eine der zehn Dateien auf dem Datenträger des angegebenen Moduls.

Der erste zu beschreibende Sektor der Datei ist frei wählbar.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Media_WR_Blk_F(module,f_info[],file,
          sec_idx,sec_cnt,array[],array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
f_info[]	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
file	Nummer (1…10) der Datei.	LONG
sec_idx	Index (1…m) des ersten zu beschreibenden Sektors, bezogen auf den Startsektor der Datei. Der Max.wert ist abhängig von der Dateigröße.	LONG
sec_cnt	Anzahl (1…12) der zu übertragenden Datenblöcke (=Sektoren) zu 128 Werten.	LONG
array[]	Feld, dessen Daten übertragen werden.	ARRAY FLOAT
array_idx	Index (1…n) des ersten zu übertragenden Feldelements.	LONG
ret_val	Status der Medienverwaltung als Bitmuster (siehe Tabelle).	LONG

Bitnr.	31:08	07	06	05	04	03	02	01	00
	–	A8	A7	A6	A5	A4	A3	A2	A1

Falls Bit = 1 (Bit = 0: ohne Bedeutung):
A1 Ein Medium ist im Schacht.
A2 Unbekannter Fehler aufgetreten.
A3 Medienverwaltung (Glue-Logik) ist beschäftigt.
A4 Daten können gelesen werden.
A5 Daten sind geschrieben.
A6 Reset wird gerade ausgeführt.
A7 Dateiende überschritten, es werden keine Daten übertragen.
A8 Time out, das Medium hat nicht reagiert.
Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Bevor diese Anweisung genutzt werden kann, muss das Feld f_info[] mit **Media_RD_FileInfo** initialisiert werden.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hoch-prioren Prozesses.

Der Start-Sektor ist frei wählbar. Dadurch können Daten an bereits gespeicherte Daten angehängt werden. Achten Sie darauf, nicht mehr Datenblöcke anzugeben als in die Datei passen. Anderenfalls werden die Datenblöcke nicht gespeichert!

Jeder Datenblock enthält 128 Float-Werte und wird in einem Sektor gespeichert. Wenn beispielsweise in einem ersten Schritt n Datenblöcke

ab dem Start-Sektor x gespeichert wurden, werden weitere Datenblöcke angehängt, indem sie ab Sektor x+n+1 geschrieben werden.

Das Feld array[] muss mindestens array_idx+s_cnt×128 Werte enthalten.

Siehe auch

- / -

Gültig für

Storage Rev. A

Media_RD_Blk_L

Media_RD_Blk_L kopiert eine Anzahl an Datenblöcken mit Long-Werten aus einer der zehn Dateien auf dem Datenträger des angegebenen Moduls in ein Feld.

Der erste zu lesende Sektor der Datei ist frei wählbar.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Media_RD_Blk_L(module,f_info[],file,
          sec_idx,sec_cnt,array[],array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
f_info[]	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
file	Nummer (1…10) der Datei.	LONG
sec_idx	Index (1…m) des ersten zu lesenden Sektors, bezogen auf den Startsektor der Datei. Der Max.wert ist abhängig von der Dateigröße.	LONG
sec_cnt	Anzahl (1…12) der zu lesenden Datenblöcke (=Sektoren) zu 128 Long-Werten.	LONG
array[]	Zielfeld, in das die Daten übertragen werden.	ARRAY LONG
array_idx	Index (1…n) des ersten zu beschreibenden Feldelements.	LONG
ret_val	Status der Medienverwaltung als Bitmuster (s.u.).	LONG

Bitnr.	31:08	07	06	05	04	03	02	01	00
	–	A8	A7	A6	A5	A4	A3	A2	A1

Falls Bit = 1 (Bit = 0: ohne Bedeutung):
A1 Ein Medium ist im Schacht.
A2 Unbekannter Fehler aufgetreten.
A3 Medienverwaltung (Glue-Logik) ist beschäftigt.
A4 Daten können gelesen werden.
A5 Daten sind geschrieben.
A6 Reset wird gerade ausgeführt.
A7 Dateiende überschritten, es werden keine Daten übertragen.
A8 Time out, das Medium hat nicht reagiert.
Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Bevor diese Anweisung genutzt werden kann, muss das Feld f_info[] mit **Media_RD_FileInfo** initialisiert werden.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hoch-prioren Prozesses.

Der Start-Sektor ist frei wählbar. Achten Sie darauf, nicht mehr Datenblöcke zu lesen als in der Datei vorhanden sind. Anderenfalls werden keine Datenblöcke gelesen!

Jeder Datenblock enthält 128 Long-Werte und ist in einem Sektor gespeichert. Das Zielfeld array[] muss daher mindestens array_idx+s_cnt×128 Feldelemente enthalten.

Siehe auch

- / -

Gültig für

Storage Rev. A

Beispiel

```
Rem ########################################################
Rem Sinus-Tabelle mit 3600 Punkten aus Daten-Datei lesen
Rem ########################################################
#Include ADwinPro_All.Inc

#Define pstom 1                 'Adresse des Pro-STORAGE-Modules
#Define f_info Data_197         'Feld mit Datei-Informationen
#Define LUT Data_1              'Look-Up-Tabelle für den Sinus
#Define file 1                  'Datei-Nummer zum Lesen
                                 'des Sinus
Rem Die Länge des Felds mit dem Sinus muss ein Vielfaches
Rem von 128 und damit größer oder gleich „nds" sein:
#Define lng 3712                'hier: 29x128
#Define pi2 6.2831853           'Zahlenwert für 2*pi

Dim f_info[22] As Long At DM_Local 'Dimensionierung des Felds
Dim LUT[lng] As Long            'Look-Up-Tabelle mit LONGs
Dim sec_p[128] As Long          'Sektor für Schreib-Pointer
Dim idx, n, ret, nds As Long    'Lokale Variablen
Dim sec_c, sec_n As Long        'Sektor-Variablen
Dim dzn, rmn As Long            'Blöcke zu 12 Sek., restl. Werte

Init:
  Par_52 = Media_RD_FileInfo(pstom,f_info)   'Datei-Info holen
  Rem Überprüfen, ob Medium gesteckt und bereit zum Lesen.
  Rem Sonst -> Exit
  If ((Par_52 And 1001b) <> 1001b) Then Exit

  Rem Länge der gesp. Tabelle ermitteln - dazu 1. Pointer
                                 'Sektor laden:
  ret = Media_RD_Blk_L(pstom,f_info,(file-1),1,1,sec_p,1)
  If ((ret And 22h) > 0) Then Exit 'Exit bei Reset o. Error

  Rem 1. Daten-Pointer auf Gültigkeit überprüfen
  If ((sec_p[1] XOr sec_p[2]) <> -1) Then

    Rem 1. Daten-Ptr. ist ungültig -> 2. Daten-Pointer auswerten
    ret = Media_RD_Blk_L(pstom,f_info,(file-1),11,1,sec_p,1)
    If ((ret And 22h) > 0) Then Exit  'Exit bei RESET o. ERROR

    Rem 2. Daten-Pointer auf Gültigkeit überprüfen
    If ((sec_p[1] XOr sec_p[2]) <> -1) Then
      Rem 2. Daten-Ptr. ebenfalls ungültig: Exit
      Exit
    Else
      nds = sec_p[1]             '2. Daten-Pointer übernehmen
    EndIf
  Else
    nds = sec_p[1]               '1. Daten-Pointer übernehmen
  EndIf

  Rem Anzahl zu ladender Sektoren und Pakete (á 12 Sektoren)
                                 'ermitteln:
  dzn = nds/1536                 'Anz. von Paketen á 12 Sektoren
  rmn = nds - dzn*1536           'Anz. restl. LONGs errechnen
  sec_n = Shift_Right(rmn,7)     'Anz. weitere zu ladende
                                 'Sektoren ...
  If ((128*sec_n) < rmn) Then
    Rem Sektor war angefangen: plus eins
    Inc sec_n
  EndIf

  Rem Sektoren der Daten-Datei ins Feld kopieren
  If (dzn>0) Then                'Pakete á 12 Sektoren vorhanden?
```

```
  For n=1 To dzn              'Alle Pakete kopieren ...
   sec_c = 12*n - 11          'aktuellen Sektor errechnen
   idx = 1536*n - 1535        'Pointer im Ziel-Array errechnen
   Rem 12 Sektoren lesen
   ret = Media_RD_Blk_L(pstom,f_info,file,sec_c,12,LUT,idx)
   If ((ret And 22h) > 0) Then Exit 'Exit bei Reset o. Error
  Next n
EndIf

Rem restliche Sektoren kopieren
ret = Media_RD_Blk_L(pstom,f_info,file,(12*dzn+1),sec_
 n, LUT,(1536*dzn+1))
If ((ret And 22h) > 0) Then Exit 'Exit bei Reset o. Error
```

Media_RD_Blk_F

Media_RD_Blk_F kopiert eine Anzahl an Datenblöcken mit Float-Werten aus einer der zehn Dateien auf dem Datenträger des angegebenen Moduls in ein Feld.

Der erste zu lesende Sektor der Datei ist frei wählbar.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Media_RD_Blk_F(module,f_info[],file,
          sec_idx,sec_cnt,array[],array_idx)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
f_info[]	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
file	Nummer (1…10) der Datei.	LONG
sec_idx	Index (1…m) des ersten zu lesenden Sektors, bezogen auf den Startsektor der Datei. Der Max.wert ist abhängig von der Dateigröße.	LONG
sec_cnt	Anzahl (1…12) der zu lesenden Datenblöcke (=Sektoren) zu 128 Long-Werten.	LONG
array[]	Zielfeld, in das die Daten übertragen werden.	ARRAY FLOAT
array_idx	Index (1…n) des ersten zu beschreibenden Feldelements.	LONG
ret_val	Status der Medienverwaltung als Bitmuster (s.u.).	LONG

Bitnr.	31:08	07	06	05	04	03	02	01	00
	–	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1

Falls Bit = 1 (Bit = 0: ohne Bedeutung):
A_1 Ein Medium ist im Schacht.
A_2 Unbekannter Fehler aufgetreten.
A_3 Medienverwaltung (Glue-Logik) ist beschäftigt.
A_4 Daten können gelesen werden.
A_5 Daten sind geschrieben.
A_6 Reset wird gerade ausgeführt.
A_7 Dateiende überschritten, es werden keine Daten übertragen.
A_8 Time out, das Medium hat nicht reagiert.
Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Bevor diese Anweisung genutzt werden kann, muss das Feld f_info[] mit **Media_RD_FileInfo** initialisiert werden.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hochprioren Prozesses.

Der Start-Sektor ist frei wählbar. Achten Sie darauf, nicht mehr Datenblöcke zu lesen als in der Datei vorhanden sind. Anderenfalls werden keine Datenblöcke gelesen!

Jeder Datenblock enthält 128 Float-Werte und ist in einem Sektor gespeichert. Das Zielfeld array[] muss daher mindestens array_idx+s_cnt×128 Feldelemente enthalten.

Siehe auch

- / -

Gültig für

Storage Rev. A

Media_RD_FileInfo

Media_RD_FileInfo initialisiert die Mediumverwaltung (Glue-Logik) auf dem angegebenen Modul und gibt die Datei-Informationen (Start- und Endsektor) in einem Feld zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Media_RD_FileInfo(module,f_info[])
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
f_info[]	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
ret_val	Status der Medienverwaltung als Bitmuster (siehe Tabelle).	LONG

Bitnr.	31:06	05	04	03	02	01	00
	–	A_6	A_5	A_4	A_3	A_2	A_1

Falls Bit = 1 (Bit = 0: ohne Bedeutung):
A_1 Ein Medium ist im Schacht.
A_2 Unbekannter Fehler aufgetreten.
A_3 Medienverwaltung (Glue-Logik) ist beschäftigt.
A_4 Daten können gelesen werden.
A_5 Daten sind geschrieben.
A_6 Reset wird gerade ausgeführt.
Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Das Feld f_info[] muss mit **Media_RD_FileInfo** initialisiert werden, bevor Daten auf das Medium geschrieben oder von dort gelesen werden können. Das Feld muss mindestens 22 Elemente groß sein.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LowInit:** oder **Finish:** eines hochprioren Prozesses.

Die Feldelemente mit ungeradem Index enthalten jeweils den Startsektor einer Datei, diejenigen mit geradem Index die Endsektoren. Die folgende Tabelle zeigt die Zuordnung zwischen den Indexnummern des Felds und den Dateien.

Indexnr.	Datei
1, 2	Fileinfo.dat
3, 4	ADWIN1.dat
…	…
21, 22	ADWIN10.dat

Siehe auch

- / -

Gültig für

Storage Rev. A

Beispiel

```
Rem ########################################################
Rem Auslesen und Interpretation der Datei <FILEINFO.DAT>
Rem ########################################################
Rem Par_1 ... Par_10:  Erster Sektor der Datei 1 ... 10
Rem Par_11 ... Par_20:  Letzter Sektor der Datei 1 ... 10
Rem Par_21 ... Par_30:  Anzahl Sektoren der Datei 1 ... 10
Rem Par_31 ... Par_40:  Anzahl LONGs/FLOATs der Datei 1 ... 10
Rem ########################################################
#Include ADwinPro_All.Inc

#Define pstom 1                'Adresse des Pro-STORAGE-Modules
#Define f_info Data_197        'Feld mit Datei-Informationen

Dim f_info[22] As Long At DM_Local 'Dimensionierung des Felds
Dim n As Long                  'lokale Variable

Init:
  Par_52 = Media_RD_FileInfo(pstom,f_info)  'Datei-Info holen
  Rem Überprüfen, ob Medium gesteckt und bereit zum Lesen.
  Rem Sonst -> Exit
  If ((Par_52 And 1001b) <> 1001b) Then Exit

  Rem alle 10 Dateien lesen
  For n = 1 To 10
    Rem Nummer des ersten und letzten Dateisektors berechnen
    PAR[n] = f_info[n*2 + 1]
    PAR[n+10] = f_info[n*2 + 2]
    Rem Dateilänge > 1 Sektor?
    If ((PAR[n+10] - PAR[n]) <> 0) Then
      PAR[n+20] = PAR[n+10] - PAR[n] + 1 'Anz. Sektoren
      PAR[n+30] = PAR[n+20] * 128 'Anz. Datenwerte
    EndIf
  Next n

  Exit
```

3.5 Pro I: Signalwandlung und Schnittstellen

Dieser Abschnitt enthält Befehle, die für Pro II-Module zur Signalwandlung oder für Pro II-Module mit Schnittstellen gelten.

Die Befehle sind nach Schnittstellentyp sortiert:

- Thermo-Module, Seite 161
- CAN-Bus, Seite 176
- Feldbus, Seite 194
- RSxxx, Seite 206
- LS-Bus + ADwin-Pro I, Seite 227

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für ein bestimmtes Modul anwendbar sind.

In den Anwendungsbeispielen wird davon ausgegangen, dass auf dem A/D-Modul die Adresse 1 eingestellt ist.

3.5.1 Thermo-Module

Dieser Abschnitt beschreibt folgende Befehle:

- PT100_Dig_To_Temp (Seite 162)
- PT100_Dig_To_R (Seite 163)
- TC_Select (Seite 164)
- TCJ_Dig_To_Temp (Seite 165)
- TCK_Dig_To_Temp (Seite 166)
- TC_Read_B (Seite 167)
- TC_Read_E (Seite 168)
- TC_Read_J (Seite 169)
- TC_Read_K (Seite 170)
- TC_Read_N (Seite 171)
- TC_Read_R (Seite 172)
- TC_Read_S (Seite 173)
- TC_Read_T (Seite 174)
- TC_Set_Rate (Seite 175)

PT100_Dig_To_Temp

PT100_Dig_To_Temp berechnet aus dem Digitalwert eines PT100-Fühlers die zugehörige Temperatur in Celsius oder Fahrenheit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = PT100_Dig_To_Temp(dig_val, t_unit)
```

Parameter

dig_val	Digitalwert (0…65535).	LONG
t_unit	Temperatur-Einheit: 1: Grad Celsius. 2: Grad Fahrenheit.	LONG
ret_val	Temperatur in Grad Celsius oder in Grad Fahrenheit.	FLOAT

Bemerkungen

Für die Ermittlung der Temperaturwerte in °C und °F aus der Thermospannung wird die Grundwertreihe der IEC 751 (= EN 60751: 1990) verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Alternativ kann die Temperatur von Hand mit einer Konvertierungstabelle berechnet werden: Temperaturfühler PT100-x.

Sie finden die Konvertierungstabellen in der Online-Hilfe von *ADbasic*, Menüpunkt „Hardware-Informationen“ (unter Contents) oder auch im Datenblatt des Herstellers: Analog Devices.

Siehe auch

TC_Select, PT100_Dig_To_R

Gültig für

- / -

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des PT100-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

Beispiel

```
#Include ADwinPro_All.Inc
Dim temp[8] As Float
Dim channel As Long

Init:
  Processdelay=100000

Event:
  For channel = 1 To 8
    TC_Select(1,channel)     'nächsten Kanal wählen
    Sleep(50)                'Einschwingzeit abwarten
    Rem Thermospannung lesen und in °C umwandeln
    temp[channel] = PT100_Dig_To_Temp(ADC(1,1),1)
  Next
```

PT100_Dig_To_R

PT100_Dig_To_R berechnet aus dem Digitalwert eines PT100-Fühlers den zugehörigen Widerstand in Ohm.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = PT100_Dig_To_R(dig_val)
```

Parameter

dig_val	Digitalwert (0…65535).	LONG
ret_val	Widerstand in Ohm.	FLOAT

Bemerkungen

Für die Ermittlung der Widerstandswerte aus der Thermospannung wird folgende Formel verwendet:

$$\text{ret\_val} = 100\,\Omega + 200\,\Omega \cdot \frac{\text{dig\_val} - 32768}{65536}$$

Siehe auch

TC_Select, PT100_Dig_To_Temp

Gültig für

PT100-4, PT100-8

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des Temperaturfühler-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

Beispiel

```
#Include ADwinPro_All.Inc
Dim temp[8] As Float
Dim channel As Long

Init:
  Processdelay=100000

Event:
  For channel = 1 To 8
    TC_Select(1,channel)      'nächsten Kanal wählen
    Sleep(50)                 'Einschwingzeit abwarten
    Rem Thermospannung lesen und in °C umwandeln
    temp[channel] = PT100_Dig_To_R(ADC(1,1))
  Next
```

TC_Select

`TC_Select` schaltet den angegebenen Thermoelement-Kanal über Multiplexer auf den analogen Ausgang des Moduls.

Syntax

```
#Include ADwinPro_All.Inc
TC_Select(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Kanalnummer: TC-4, PT100-4: 1…4. TC-8, PT100-8: 1…8. TC-16: 1…16.	LONG

Bemerkungen

Die Einschwingzeit des Multiplexers muss durch entsprechende Programmierung überbrückt werden, damit ein korrekter Messwert erreicht wird. Die Einschwingzeit ist im Handbuch Pro-Hardware angegeben.

Zur Umrechnung der gemessenen Spannung in die Einheit [°C] wird die Konvertierungstabelle des Thermoelement-Verstärkers benötigt:

- TC-x Typ J (AD594)
- TC-x Typ K (AD595) oder
- PT100-x.

Sie finden die Konvertierungstabellen in der Online-Hilfe von *ADbasic*, Menüpunkt Hardware-Informationen (unter Contents) oder auch im Datenblatt des Herstellers: Analog Devices.

Siehe auch

PT100_Dig_To_Temp, PT100_Dig_To_R, TCJ_Dig_To_Temp, TCK_Dig_To_Temp

Gültig für

- / -

Beispiel

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des Thermoelement-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

Beispiel

```
#Include ADwinPro_All.Inc
Dim value, i As Long

Init:
  value = 0                   'Variablen…
  i = 1                       ' initialisieren

Event:
  TC_Select(1,3)              'Thermoelement-Kanal 3 wählen
  Rem Fügen Sie Programmcode ein, damit die Temperaturmessung
  Rem erst nach der Einschwingzeit des Multiplexers erfolgt.
  Rem Der nachfolgende Befehl ADC benötigt bereits einen Teil
  Rem dieser Zeit.
  value = value + ADC(1,1)  'Wert messen
  If (i = 10) Then
    Par_1 = value / 10        'Mittelwert aus 10 Messungen
    value = 0
    i = 0
  EndIf
  Inc i
```

TCJ_Dig_To_Temp

TCJ_Dig_To_Temp berechnet aus einem Digitalwert eines Thermoelements vom Typ J die zugehörige Temperatur in Celsius oder Fahrenheit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TCJ_Dig_To_Temp(dig_val, t_unit)
```

Parameter

dig_val	Digitalwert (0…65535).	LONG
t_unit	Temperatur-Einheit: 1: Grad Celsius. 2: Grad Fahrenheit.	LONG
ret_val	Temperatur in Grad Celsius oder in Grad Fahrenheit.	FLOAT

Bemerkungen

Für die Ermittlung der Temperaturwerte in °C und °F aus der Thermospannung wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Alternativ kann die Temperatur von Hand mit einer Konvertierungstabelle berechnet werden: Thermoelement-Verstärker TC-x Typ J (AD594).

Sie finden die Konvertierungstabellen in der Online-Hilfe von *ADbasic*, Menüpunkt „Hardware-Informationen“ (unter Contents) oder auch im Datenblatt des Herstellers: Analog Devices.

Siehe auch

TC_Select, TCK_Dig_To_Temp

Gültig für

- / -

Beispiel

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des Thermoelement-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

Beispiel

```
#Include ADwinPro_All.Inc
Dim temp[8] As Float
Dim channel As Long

Init:
  Processdelay = 100000

Event:
  For channel = 1 To 8
    TC_Select(1,channel)      'nächsten Kanal wählen
    Sleep(50)                 'Einschwingzeit abwarten
    Rem Thermospannung lesen und in °C umwandeln
    temp[channel] = TCJ_Dig_To_Temp(ADC(1,1),1)
  Next
```

TCK_Dig_To_Temp

`TCK_Dig_To_Temp` berechnet aus einem Digitalwert eines Thermoelements vom Typ K die zugehörige Temperatur in Celsius oder Fahrenheit.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TCK_Dig_To_Temp(dig_val, t_unit)
```

Parameter

dig_val	Digitalwert (0…65535).	LONG
t_unit	Temperatur-Einheit: 1: Grad Celsius. 2: Grad Fahrenheit.	LONG
ret_val	Temperatur in Grad Celsius oder in Grad Fahrenheit.	FLOAT

Bemerkungen

Für die Ermittlung der Temperaturwerte in °C und °F aus der Thermospannung wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Alternativ kann die Temperatur von Hand mit einer Konvertierungstabelle berechnet werden: Thermoelement-Verstärker TC-x Typ K (AD595).

Sie finden die Konvertierungstabellen in der Online-Hilfe von *ADbasic*, Menüpunkt „Hardware-Informationen“ (unter Contents) oder auch im Datenblatt des Herstellers: Analog Devices.

Siehe auch

TC_Select, TCJ_Dig_To_Temp

Gültig für

- / -

Beispiel

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des Thermoelement-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

Beispiel

```
#Include ADwinPro_All.Inc
Dim temp[8] As Float
Dim channel As Long

Init:
  Processdelay = 100000

Event:
  For channel = 1 To 8
    TC_Select(1,channel)      'nächsten Kanal wählen
    Sleep(50)                 'Einschwingzeit abwarten
    Rem Thermospannung lesen und in °C umwandeln
    temp[channel] = TCK_Dig_To_Temp(ADC(1,1),1)
  Next
```

TC_Read_B

TC_Read_B gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ B.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TC_Read_B(module, channel, ret_type)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1…8) des Kanals.	LONG
ret_type	Typ des Rückgabewerts ret_val: 0: Thermospannung in µV. 1: Temperatur in °C. 2: Temperatur in °F.	LONG
ret_val	Messwert des Kanals, abhängig von ret_type: Thermospannung: 291 µV … 13820 µV. Temperatur in °C: 250°C … 1820°C. Temperatur in °F: 482°F … 3329,6°F.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtastrate siehe **TC_Set_Rate**). Die Anweisung **TC_Read_B** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in °C und °F wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

TC_Read_E, TC_Read_J, TC_Read_K, TC_Read_N, TC_Read_R, TC_Read_S, TC_Read_T, TC_Set_Rate

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem Read temperature in °C from channel 5
  FPar_1 = TC_Read_B(1,5,1)
```

TC_Read_E

TC_Read_E gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ E.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TC_Read_E(module, channel, ret_type)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1…8) des Kanals.	LONG
ret_type	Typ des Rückgabewerts ret_val: 0: Thermospannung in µV. 1: Temperatur in °C. 2: Temperatur in °F.	LONG
ret_val	Messwert des Kanals, abhängig von ret_type: Thermospannung: -8825µV … 76373µV. Temperatur in °C: -200°C … 1000°C. Temperatur in °F: -328°F … 1832°F.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtastrate siehe **TC_Set_Rate**). Die Anweisung **TC_Read_E** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in °C und °F wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

TC_Read_B, TC_Read_J, TC_Read_K, TC_Read_N, TC_Read_R, TC_Read_S, TC_Read_T, TC_Set_Rate

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem Read temperature in °C from channel 5
  FPar_1 = TC_Read_E(1,5,1)
```

TC_Read_J

TC_Read_J gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ J.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TC_Read_J(module, channel, ret_type)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1…8) des Kanals.	LONG
ret_type	Typ des Rückgabewerts ret_val: 0: Thermospannung in µV. 1: Temperatur in °C. 2: Temperatur in °F.	LONG
ret_val	Messwert des Kanals, abhängig von ret_type: Thermospannung: -8095 µV … 69553 µV. Temperatur in °C: -210°C … 1200°C. Temperatur in °F: -346°F … 2192°F.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtastrate siehe **TC_Set_Rate**). Die Anweisung **TC_Read_J** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in °C und °F wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

TC_Read_B, TC_Read_E, TC_Read_K, TC_Read_N, TC_Read_R, TC_Read_S, TC_Read_T, TC_Set_Rate

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem Read temperature in °C from channel 5
  FPar_1 = TC_Read_J(1,5,1)
```

TC_Read_K

`TC_Read_K` gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ K.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TC_Read_K(module, channel, ret_type)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`channel`	Nummer (1…8) des Kanals.	LONG
`ret_type`	Typ des Rückgabewerts `ret_val`: 0: Thermospannung in µV. 1: Temperatur in °C. 2: Temperatur in °F.	LONG
`ret_val`	Messwert des Kanals, abhängig von `ret_type`: Thermospannung: -5891 µV … 54886 µV. Temperatur in °C: -200°C … 1372°C. Temperatur in °F: -328°F … 2501,6°F.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtastrate siehe `TC_Set_Rate`). Die Anweisung `TC_Read_K` gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in °C und °F wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

TC_Read_B, TC_Read_E, TC_Read_J, TC_Read_N, TC_Read_R, TC_Read_S, TC_Read_T, TC_Set_Rate

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem Read temperature in °C from channel 5
  FPar_1 = TC_Read_K(1,5,1)
```

TC_Read_N

TC_Read_N gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ N.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TC_Read_N(module, channel, ret_type)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1…8) des Kanals.	LONG
ret_type	Typ des Rückgabewerts ret_val: 0: Thermospannung in µV. 1: Temperatur in °C. 2: Temperatur in °F.	LONG
ret_val	Messwert des Kanals, abhängig von ret_type: Thermospannung: -3990 µV … 47513 µV. Temperatur in °C: -200°C … 1300°C. Temperatur in °F: -328°F … 2372°F.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtastrate siehe **TC_Set_Rate**). Die Anweisung **TC_Read_N** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in °C und °F wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

TC_Read_B, TC_Read_E, TC_Read_J, TC_Read_K, TC_Read_R, TC_Read_S, TC_Read_T, TC_Set_Rate

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem Read temperature in °C from channel 5
  FPar_1 = TC_Read_N(1,5,1)
```

TC_Read_R

`TC_Read_R` gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ R.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TC_Read_R(module, channel, ret_type)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`channel`	Nummer (1…8) des Kanals.	LONG
`ret_type`	Typ des Rückgabewerts `ret_val`: 0: Thermospannung in µV. 1: Temperatur in °C. 2: Temperatur in °F.	LONG
`ret_val`	Messwert des Kanals, abhängig von `ret_type`: Thermospannung: -226 µV … 21101 µV. Temperatur in °C: -50°C … 1768°C. Temperatur in °F: -58°F … 3214,4°F.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtastrate siehe `TC_Set_Rate`). Die Anweisung `TC_Read_R` gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in °C und °F wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

TC_Read_B, TC_Read_E, TC_Read_J, TC_Read_K, TC_Read_N, TC_Read_S, TC_Read_T, TC_Set_Rate

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem Read temperature in °C from channel 5
  FPar_1 = TC_Read_R(1,5,1)
```

TC_Read_S

TC_Read_S gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ S.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TC_Read_S(module, channel, ret_type)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1…8) des Kanals.	LONG
ret_type	Typ des Rückgabewerts ret_val: 0: Thermospannung in µV. 1: Temperatur in °C. 2: Temperatur in °F.	LONG
ret_val	Messwert des Kanals, abhängig von ret_type: Thermospannung: -236 µV … 18693 µV. Temperatur in °C: -50°C … 1768°C. Temperatur in °F: -58°F … 3214,4°F.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtastrate siehe **TC_Set_Rate**). Die Anweisung **TC_Read_S** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in °C und °F wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

TC_Read_B, TC_Read_E, TC_Read_J, TC_Read_K, TC_Read_N, TC_Read_R, TC_Read_T, TC_Set_Rate

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem Read temperature in °C from channel 5
  FPar_1 = TC_Read_S(1,5,1)
```

TC_Read_T

`TC_Read_T` gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ T.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = TC_Read_T(module, channel, ret_type)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`channel`	Nummer (1…8) des Kanals.	LONG
`ret_type`	Typ des Rückgabewerts `ret_val`: 0: Thermospannung in µV. 1: Temperatur in °C. 2: Temperatur in °F.	LONG
`ret_val`	Messwert des Kanals, abhängig von `ret_type`: Thermospannung: -5603µV … 20872µV. Temperatur in °C: -200°C … 400°C. Temperatur in °F: -454°F … 752°F.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtastrate siehe TC_Set_Rate). Die Anweisung `TC_Read_T` gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in °C und °F wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

TC_Read_B, TC_Read_E, TC_Read_J, TC_Read_K, TC_Read_N, TC_Read_R, TC_Read_S, TC_Set_Rate

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem Temperatur in °C am Kanal 5 abfragen
  FPar_1 = TC_Read_T(1,5,1)
```

TC_Set_Rate

TC_Set_Rate stellt die Abtastrate für das angegebene Modul ein.

Syntax

```
#Include ADwinPro_All.Inc
TC_Set_Rate(module, rate)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
rate	Kennzahl für die gewählte Abtastrate (siehe Tabelle); Voreinstellung: 15.	LONG

Kennzahl	Abtastrate [Hz]	ADC-Rauschen [nV]
1	3520	23000
2	1760	3500
3	880	2000
4	440	1400
5	220	1000
6	110	750
7	55	510
8	27,5	375
9	13,75	250
15	6,875	200

Bemerkungen

Die Abtastrate gilt für alle Kanäle gleichermaßen.

Mit steigender Abtastrate steigt das Rauschsignal, das am ADC eines Kanals entsteht und das eintreffende Signal überlagert (siehe Tabelle).

Siehe auch

TC_Read_B, TC_Read_E, TC_Read_J, TC_Read_K, TC_Read_N, TC_Read_R, TC_Read_S, TC_Read_T

Gültig für

TC-8-ISO Rev. A

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Rem Abtastrate auf 27,5 Hz einstellen
  TC_Set_Rate(1,8)
```

3.5.2 CAN-Bus

Dieser Abschnitt beschreibt folgende Befehle:

CAN_Msg

CAN_Msg ist ein eindimensionales Feld mit 9 Elementen, in dem die Message-Objekte gespeichert sind oder werden.

Syntax

```
#Include ADwinPro_All.Inc
CAN_Msg[n] = value
```

oder

```
value = CAN_Msg[n]
```

Parameter

n	Elementnummer im Feld **CAN_Msg** (1…9)	LONG
value	Wert (8 Bit), der in das Message-Objekt geschrieben oder daraus gelesen wird.	LONG
ret_val	Wert (0…256), der aus dem Message-Objekt gelesen wird.	LONG

Bemerkungen

Die Elemente des Felds **CAN_Msg**[] haben folgende Funktion:

Elementnr. in **CAN_Msg**	1…8	9
Inhalt	Message-Objekt(e) = Datenbyte(s)	Anzahl (0…8) belegter Datenbytes

Tragen Sie die zu übertragenden Datenbytes und ihre Anzahl in das Feld **CAN_Msg**[] ein, *bevor* Sie diese mit **Transmit** übertragen.

Siehe auch

En_Receive, En_Transmit, Read_Msg, Transmit

Gültig für

CAN-1, CAN-2

Beispiel

```
REM Sends a 32 Bit FLOAT-value (here: Pi) as sequence of
REM 4 bytes in a message object
REM (Receiving of a float value see example at Read_Msg)

#Include ADwinPro_All.Inc
#Define pi 3.14159265
Dim i As Long

Init:
  Init_CAN(1,1)              'Initialize CAN controller 1

  REM Enable message object 6 of controller 1
  REM for sending with the identifier 40 (11 bit)
  En_Transmit(1,1,6,40,0)

  REM Create bit pattern of Pi with data type Long
  Par_1 = Cast_FloatToLong(pi)

  REM divide bit pattern (32 Bit) into 4 bytes
  CAN_Msg[4] = Par_1 And 0FFh 'assign LSB
  For i = 1 To 3
    CAN_Msg[4-i] = Shift_Right(Par_1,8*i) And 0FFh
  Next i
  CAN_Msg[9] = 4             'message length in bytes

Event:
  Transmit(1,1,6)            'Send the message object 6
```

En_Interrupt

En_Interrupt konfiguriert ein Message-Objekt des angegebenen Moduls so, dass es bei Eintreffen einer Nachricht einen Event-Signal (Interrupt) erzeugt.

Syntax

```
#Include ADwinPro_All.Inc
En_Interrupt(module,channel,msg_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) der CAN-Schnittstelle.	LONG
msg_no	Nummer (1…15) des Message-Objektes.	LONG

Bemerkungen

Ein erzeugtes Event-Signal wird nur dann an das Prozessormodul geleitet, wenn das Event-Signal mit **EventEnable** freigegeben ist. Konfigurieren Sie zuerst alle gewünschten Message-Objekte und geben Sie erst anschließend das Event-Signal frei.

In einem System darf – zusätzlich zum Prozessor-Modul – nur ein einziger Event-Eingang aktiv sein, d.h. Sie müssen einen eventuell aktiven Event-Eingang sperren, bevor Sie den Event-Eingang an einem anderen Modul aktivieren.

Siehe auch

En_Receive, EventEnable, Get_CAN_Reg, Init_CAN

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  Rem CAN-Controller 1 auf dem CAN-Modul 1 initialisieren
  Init_CAN(1,1)
  Rem Message-Objekte 3 und 15 zum Lesen konfigurieren
  En_Receive(1,1,3,1,0)
  En_Receive(1,1,15,385,0)
  Rem Interrupt für Message-Objekte 3 und 15 konfigurieren
  En_Interrupt(1,1,3)
  En_Interrupt(1,1,15)
  Rem Event-Signal freigeben
  EventEnable(1,ext,1)
Event:
  Rem Interruptregister lesen und Wert in Objektnr. wandeln (s.u.)
  Par_13 = Get_CAN_Reg(1,1,5Fh)
  If (Par_13 = 2) Then
    Par_13 = 15
  Else
    Par_13 = Par_13 - 2
  EndIf
```

Der Wert im Interrupt-Register **5Fh** entspricht einem der Message-Objekte nach folgendem Schema:

Wert	2	3	4	…	16
Nummer Message-Objekt	15	1	2	…	14

En_Receive

En_Receive gibt ein Message-Objekt für den Empfang von Nachrichten auf dem angegebenen Modul frei.

Für das Message-Objekt werden der CAN-Kanal, die Länge des Nachrichten-Identifiers und der Identifier selbst festgelegt.

Syntax

```
#Include ADwinPro_All.Inc
En_Receive(module,channel,msg_no,id,id_extend)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
msg_no	Nummer (1…15) des Message-Objektes im CAN-Controller.	LONG
id	Identifier der Nachrichten, die in diesem Message-Objekt empfangen werden sollen ($0…2^{11}$ oder $0…2^{29}$).	LONG
id_extend	Merker für die Länge des Identifiers: 0: 11 Bit Identifier. 1: 29 Bit Identifier.	LONG

Bemerkungen

Ein Message-Objekt kann nur Nachrichten vom CAN-Bus empfangen, wenn Sie es zuvor mit **En_Receive** zum Empfang freigegeben haben.

Das Message-Objekt empfängt nur Nachrichten mit dem von Ihnen angegebenen Identifier.

Siehe auch

CAN_Msg, En_Transmit, Init_CAN, Read_Msg, Transmit

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Rem Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
  Init_CAN(1,1)
  Rem Message-Objekt 1 freigeben für den Empfang von
  Rem CAN-Nachrichten mit dem 11 Bit-Identifier 200
  En_Receive(1,1,1,200,0)
```

En_Transmit

En_Transmit gibt ein Message-Objekt für das Senden von Nachrichten auf dem angegebenen Modul frei.

Für das Message-Objekt werden der CAN-Kanal, die Länge des Nachrichten-Identifiers und der Identifier selbst festgelegt.

Syntax

```
#Include ADwinPro_All.Inc
En_Transmit(module,channel,msg_no,id,id_extend)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
msg_no	Nummer (1…14) des Message-Objektes im CAN-Controller.	LONG
id	Identifier der Nachrichten, die in diesem Message-Objekt gesendet werden sollen (0…2^{11} oder 0…2^{29}).	LONG
id_extend	Merker für die Länge des Identifiers: 0: 11 Bit Identifier. 1: 29 Bit Identifier.	LONG

Bemerkungen

Erst wenn ein Message-Objekt mit **En_Transmit** zum Senden freigegeben ist, kann das Objekt Nachrichten auf dem CAN-Bus senden.

Siehe auch

CAN_Msg, En_Receive, Init_CAN, Read_Msg, Transmit, Transmit_Status

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  Rem Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
  Init_CAN(1,1)
  Rem Message-Objekt 6 freigeben für das Senden von
  Rem CAN-Nachrichten mit dem 11 Bit-Identifier 40
  En_Transmit(1,1,6,40,0)
```

Get_CAN_Reg

Get_CAN_Reg gibt den Inhalt eines bestimmten Registers auf einem CAN-Controller des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Get_CAN_Reg(module,channel,regno)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
regno	Register-Nummer des CAN-Controllers (0…255).	LONG
ret_val	Inhalt des CAN-Controller-Registers.	LONG

Bemerkungen

Sie finden die Registernummern des CAN-Controllers AN82527 im Intel®-Datenblatt (Address map). Beispiele sind:

- Adresse **00h**: Kontroll-Register
- Adresse **01h**: Status-Register
- Adresse **5fh**: Interrupt-Register

Siehe auch

En_Interrupt, Init_CAN, Set_CAN_Baudrate, Set_CAN_Reg

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  Rem Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
  Init_CAN(1,1)
  Rem Das Kontroll-Register des CAN-Controller 1, Modul 1 auslesen
  Par_1 = Get_CAN_Reg(1,1,0)
```

Init_CAN

Init_CAN initialisiert einen der CAN-Controller auf dem angegebenen Modul und bringt ihn in einen definierten Anfangszustand.

Syntax

```
#Include ADwinPro_All.Inc
Init_CAN(module,channel)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG

Bemerkungen:

Die Anweisung führt folgende Aktionen aus:

- Reset (Hardware-Reset des CAN-Controllers)
- Alle Filter auf "must match" setzen.
- Clockout-Register auf 0 setzen (= externe Frequenz wird nicht geteilt).
- Register „Bus-Configuration“ auf 0 setzen.
- Übertragungsrate für den CAN-Bus auf 1 MBit/s setzen.
- Alle Message-Objekte sperren.

Sie müssen diese Anweisung ausführen, bevor Sie mit anderen Befehlen auf den CAN-Controller zugreifen. Wir empfehlen die Angabe im Prozessabschnitt **LowInit:** oder **Init:**.

Bei Low speed CAN beträgt die Übertragungsrate maximal 125 kBit/s und muss deswegen auf jeden Fall mit **Set_CAN_Baudrate** neu eingestellt werden.

Siehe auch

En_Receive, En_Transmit, Get_CAN_Reg, Set_CAN_Baudrate, Set_CAN_Reg

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  Rem Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
  Init_CAN(1,1)
```

Read_Msg

Read_Msg gibt zurück, ob eine neue Nachricht in einem Message-Objekt eines der CAN-Controller auf dem angegebenen Modul empfangen wurde.

Falls ja, wird der Nachrichteninhalt in das Feld **CAN_Msg** kopiert und der Identifier zurückgegeben.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Read_Msg(module,channel,msg_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
msg_no	Nummer (1… 15) des Message-Objektes im CAN-Controller.	LONG
ret_val	Nachrichtenstatus und -inhalt: ≥-1: Eine neue Nachricht ist eingegangen, der Wert ist der Identifier des Message-Objektes. -1: keine neue Nachricht vorhanden.	LONG

Bemerkungen

Um eine Nachricht zu empfangen, müssen Sie folgende Reihenfolge einhalten:

- Einmal: Geben Sie das Message-Objekt mit **En_Receive** zum Empfangen frei.
- Sooft erforderlich: Prüfen Sie auf eine neue Nachricht und – falls vorhanden – speichern die Nachricht in **CAN_MSG** mit **Read_Msg**.

Sie können eine empfangene Nachricht nur einmal auslesen.

Siehe auch

CAN_Msg, En_Receive, En_Transmit, Init_CAN, Transmit, Transmit_Status

Gültig für

CAN-1, CAN-2

Beispiel

```
Rem Wenn eine neue Nachricht mit dem passenden Identifier
Rem empfangen wurde, werden die Daten gelesen. Die
Rem ersten 4 Bytes der Nachricht werden zu einer Fließkomma-
Rem Zahl mit 32 Bit Länge zusammengesetzt (Senden einer
Rem Fließkomma-Zahl siehe Bsp. bei Transmit).
#Include ADwinPro_All.Inc
Dim n As Long

Init:
  Par_1 = 0
  Init_CAN(1,1)              'CAN-Controller 1 initialisieren
  En_Receive(1,1,8,40,0)     'Message-Objekt 8 initialisieren
                             'zum Empfangen von CAN-Nachrichten
                             'mit dem Identifier 40


Event:
  Rem Wenn das Message-Objekt geändert wurde, werden die
  Rem empfangenen Daten aus Objekt 8 gelesen und der
  Rem Identifier an Par_9 übergeben.
  Rem Die Daten stehen im Feld CAN_Msg[] bereit.
  Par_9 = Read_Msg(1,1,8)

  If (Par_9 = 40) Then
    Rem Für das Message-Objekt ist eine neue Nachricht mit dem
    Rem Identifier 40 eingetroffen
    Par_1 = CAN_Msg[1]       'High-Byte auslesen
    For n = 2 To 4           'Mit restlichen 3 Bytes zu 32 Bit-Zahl
     Par_1 = Shift_Left(Par_1,8) + CAN_Msg[n] 'zusammenfügen
    Next n
    Rem Das Bitmuster in Par_1 in den Datentyp Float wandeln und
    Rem der Variablen FPar_1 zuweisen.
    FPar_1 = Cast_LongToFloat(Par_1)
  EndIf
```

Read_Msg_Con

Falls ja, wird die Nachricht in **CAN_Msg** gespeichert und der Identifier der Nachricht zurückgegeben.

Read_Msg gibt zurück, ob eine neue Nachricht in einem Message-Objekt einer CAN-Schnittstelle empfangen wurde.

Syntax

```
#Include ADwinPro_All.Inc
```

module	Eingestellte Moduladresse (1…255).	LONG
can_no	Nummer (1, 2) der CAN-Schnittstelle	LONG

Bemerkungen

Im Unterschied zu **Read_Msg** stellt **Read_Msg_Con** sicher, dass die Nachricht konsistent ist: Wenn während des Auslesens eine neue Nachricht eintrifft, kann es nicht zu einer Mischung der alten und der neuen Nachricht kommen.

Um eine Nachricht zu empfangen, müssen Sie folgende Reihenfolge einhalten:

- Einmal: Geben Sie das Message-Objekt mit **En_Receive** zum Empfangen frei.
- Sooft erforderlich: Prüfen Sie auf eine neue Nachricht und – falls vorhanden – speichern die Nachricht in **CAN_MSG** mit **Read_Msg**.

Sie können eine empfangene Nachricht nur einmal auslesen.

Siehe auch

CAN_Msg, En_Interrupt, En_Receive, En_Transmit, Read_Msg

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Rem Wenn eine neue Nachricht mit dem passenden Identifier
Rem empfangen wurde, werden die Daten gelesen. Die
Rem ersten 4 Bytes der Nachricht werden zu einer Fließkomma-
Rem Zahl mit 32 Bit Länge zusammengesetzt.
Dim n As Long

INIT:
  PAR_1 = 0
  Init_CAN(module, 1)          'CAN-Controller 1 initialisieren
  Rem Message-Objekt 1 initialisieren zum Empfangen von
  Rem CAN-Nachrichten mit dem Identifier 40
  En_Receive(module,1,1,40,0)


EVENT:
  Rem Wenn das Message-Objekt geändert wurde, werden die
  Rem empfangenen Daten aus Objekt 1 gelesen und der
  Rem Identifier an PAR_9 übergeben.
  Rem Die Daten stehen im Feld CAN_Msg[] bereit.
  PAR_9 = Read_Msg_Con(module,1,1)

  If (PAR_9 = 40) Then
    Rem Für das Message-Objekt ist eine neue Nachricht mit dem
    Rem Identifier 40 eingetroffen
    PAR_1 = CAN_Msg[1]          'High-Byte auslesen
    FOR n = 2 TO 4 'Mit restlichen 3 Bytes zu 32 Bit-Zahl
      PAR_1 = Shift_Left(PAR_1,8) + CAN_Msg[n] 'zusammenfügen
    NEXT n
    Rem Das Bitmuster in PAR_1 in den Datentyp FLOAT wandeln und
    Rem der Variablen FPAR_1 zuweisen.
    FPAR_1 = Cast_LongToFloat(PAR_1)
  EndIf
```

Senden einer Fließkomma-Zahl siehe Bsp. bei Transmit.

Set_CAN_Baudrate

Set_CAN_Baudrate stellt die angegebene Baudrate auf einem der Controller auf dem angegebenen Modul ein und gibt zurück, ob dies erfolgreich geschehen ist.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Set_CAN_Baudrate(module,channel,rate)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
rate	Baudrate des CAN-Controllers: High speed CAN: 5000…1000000 Bit/s. Low speed CAN: 5000 … 125000 Bit/s. (Werte siehe Tabelle „Einstellbare Baudraten“).	LONG
ret_val	Status der Befehlsausführung: 0: Baudrate ist gesetzt. 1: Baudrate ist unzulässig und kann nicht gesetzt werden.	LONG

Bemerkungen

Die möglichen Baudraten (= Busfrequenzen) entnehmen Sie bitte der Tabelle „Einstellbare Baudraten“ im Anhang. Übernehmen Sie bitte die genaue Schreibweise, d.h. nicht ganzzahlige Baudraten mit 4 Nachkommastellen; Werte mit abweichender Schreibweise werden als nicht zulässig zurückgewiesen.

Die Anweisung führt folgende Aktionen aus:

- Prüfen, ob die übergebene Baudrate zulässig ist. Falls nicht, dann den Rückgabewert auf 1 setzen und die Bearbeitung beenden.
- Die Register des CAN-Controllers für die Baudrate setzen.
- Sampling Mode auf 0 setzen: Ein Sample pro Bit.
- Die Einstellungen so wählen, dass der Sample-Punkt immer zwischen 60% und 72% der Gesamt-Bitlänge liegt.
- Die Sprungweite zur Synchronisation auf 1 setzen.

In Sonderfällen kann es vorteilhaft sein, die Einstellungen anders zu wählen als oben beschrieben. Sie finden hierzu eine Erläuterung im Hardware-Handbuch.

Die Anweisung sollte in den Programm-Abschnitten **LOWINIT:** oder **INIT:** aufgerufen werden, und zwar erst nach der Anweisung Initialisierung, weil sonst die eingestellte Baudrate wieder mit der Standardeinstellung (1 MBit/s) überschrieben wird.

Siehe auch

Get_CAN_Reg, Init_CAN, Set_CAN_Reg

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.Inc
Dim status As Long
Init:
  Init_CAN(1,1)              'Initialisierung des CAN-Controllers
  status = Set_CAN_Baudrate(1,1,125000)'Baudrate = 125 kBit/s
```

Einstellbare Baudraten

Einstellbare Baudraten [Bit/s]				
1000000.0000	888888.8889	800000.0000	727272.7273	666666.6667
615384.6154	571428.5714	533333.3333	500000.0000	470588.2353
444444.4444	421052.6316	400000.0000	380952.3810	363636.3636
347826.0870	333333.3333	320000.0000	307692.3077	296296.2963
285714.2857	266666.6667	250000.0000	242424.2424	235294.1176
222222.2222	210526.3158	205128.2051	200000.0000	190476.1905
181818.1818	177777.7778	173913.0435	166666.6667	160000.0000
156862.7451	153846.1538	148148.1481	145454.5455	142857.1429
140350.8772	133333.3333	126984.1270	125000.0000	123076.9231
121212.1212	117647.0588	115942.0290	114285.7143	111111.1111
106666.6667	105263.1579	103896.1039	102564.1026	100000.0000
98765.4321	95238.0952	94117.6471	90909.0909	88888.8889
87912.0879	86956.5217	84210.5263	83333.3333	81632.6531
80808.0808	80000.0000	78431.3725	76923.0769	76190.4762
74074.0741	72727.2727	71428.5714	70175.4386	69565.2174
68376.0684	67226.8908	66666.6667	66115.7025	64000.0000
63492.0635	62500.0000	61538.4615	60606.0606	60150.3759
59259.2593	58823.5294	57971.0145	57142.8571	55944.0559
55555.5556	54421.7687	53333.3333	52631.5789	52287.5817
51948.0519	51282.0513	**50000.0000**	49689.4410	49382.7160
48484.8485	47619.0476	47337.2781	47058.8235	46783.6257
45714.2857	45454.5455	44444.4444	43956.0440	43478.2609
42780.7487	42328.0423	42105.2632	41666.6667	41025.6410
40816.3265	40404.0404	40000.0000	39215.6863	38647.3430
38461.5385	38277.5120	38095.2381	37037.0370	36363.6364
36199.0950	35714.2857	35555.5556	35087.7193	34782.6087
34632.0346	34482.7586	34188.0342	33613.4454	33333.3333
33057.8512	32921.8107	32388.6640	32258.0645	32000.0000
31746.0317	31620.5534	31372.5490	31250.0000	30769.2308
30651.3410	30303.0303	30075.1880	29629.6296	29411.7647
29304.0293	29090.9091	28985.5072	28673.8351	28571.4286
28070.1754	27972.0280	27777.7778	27681.6609	27586.2069
27210.8844	27027.0270	26936.0269	26755.8528	26666.6667
26315.7895	26143.7908	25974.0260	25806.4516	25641.0256
25396.8254	25078.3699	25000.0000	24844.7205	24767.8019
24691.3580	24615.3846	24390.2439	24242.4242	24024.0240
23809.5238	23668.6391	23529.4118	23460.4106	23391.8129
23255.8140	23188.4058	22988.5057	22857.1429	22792.0228
22727.2727	22408.9636	22222.2222	22160.6648	22038.5675
21978.0220	21739.1304	21680.2168	21621.6216	21505.3763
21390.3743	21333.3333	21276.5957	21220.1592	21164.0212
21052.6316	20833.3333	20779.2208	20671.8346	20512.8205
20460.3581	20408.1633	20202.0202	20050.1253	**20000.0000**
19851.1166	19753.0864	19704.4335	19656.0197	19607.8431
19512.1951	19323.6715	19230.7692	19138.7560	19047.6190

Einstellbare Baudraten [Bit/s]				
18912.5296	18867.9245	18823.5294	18648.0186	18604.6512
18518.5185	18433.1797	18390.8046	18306.6362	18181.8182
18140.5896	18099.5475	18018.0180	17857.1429	17777.7778
17738.3592	17582.4176	17543.8596	17429.1939	17391.3043
17316.0173	17241.3793	17204.3011	17094.0171	17021.2766
16949.1525	16913.3192	16842.1053	16806.7227	16771.4885
16666.6667	16632.0166	16563.1470	16528.9256	16460.9053
16393.4426	16326.5306	16260.1626	16227.1805	16194.3320
16161.6162	16129.0323	16000.0000	15873.0159	15810.2767
15779.0927	15686.2745	15625.0000	15594.5419	15503.8760
15473.8878	15444.0154	15384.6154	15325.6705	15238.0952
15180.2657	15151.5152	15122.8733	15094.3396	15065.9134
15037.5940	15009.3809	14842.3006	14814.8148	14705.8824
14652.0147	14571.9490	14545.4545	14519.0563	14492.7536
14414.4144	14336.9176	14311.2701	14285.7143	14260.2496
14184.3972	14109.3474	**14035.0877**	13986.0140	13937.2822
13913.0435	13888.8889	13840.8304	13793.1034	13722.1269
13675.2137	13605.4422	13582.3430	13559.3220	13513.5135
13468.0135	13445.3782	13377.9264	13333.3333	13289.0365
13223.1405	13157.8947	13136.2890	13114.7541	13093.2897
13071.8954	13008.1301	12987.0130	12903.2258	12882.4477
12820.5128	12800.0000	12759.1707	12718.6010	12698.4127
12578.6164	12558.8697	12539.1850	12500.0000	12422.3602
12403.1008	12383.9009	12345.6790	12326.6564	12307.6923
12288.7865	12195.1220	12158.0547	12121.2121	12066.3650
12030.0752	12012.0120	11994.0030	11922.5037	11904.7619
11851.8519	11834.3195	11764.7059	11730.2053	11695.9064
11661.8076	11627.9070	11611.0305	11594.2029	11544.0115
11494.2529	11477.7618	11428.5714	11396.0114	11379.8009
11363.6364	11347.5177	11299.4350	11220.1964	11204.4818
11188.8112	11111.1111	11080.3324	11034.4828	11019.2837
10989.0110	10943.9124	10928.9617	10884.3537	10869.5652
10840.1084	10810.8108	10796.2213	10781.6712	10752.6882
10695.1872	10666.6667	10638.2979	10610.0796	10582.0106
10540.1845	10526.3158	10457.5163	10430.2477	10416.6667
10389.6104	10335.9173	10322.5806	10296.0103	10269.5764
10256.4103	10230.1790	10204.0816	10101.0101	10088.2724
10062.8931	10025.0627	10012.5156	**10000.0000**	9937.8882
9925.5583	9876.5432	9852.2167	9828.0098	9803.9216
9791.9217	9768.0098	9756.0976	9696.9697	9685.2300
9661.8357	9615.3846	9603.8415	9569.3780	9523.8095
9456.2648	9433.9623	9411.7647	9400.7051	9367.6815
9356.7251	9324.0093	9302.3256	9291.5215	9259.2593
9227.2203	9216.5899	9195.4023	9153.3181	9142.8571
9090.9091	9070.2948	9049.7738	9039.5480	9009.0090
8958.5666	8928.5714	8918.6176	8888.8889	8879.0233

Einstellbare Baudraten [Bit/s]				
8869.1796	8859.3577	8771.9298	8743.1694	8714.5969
8695.6522	8658.0087	8648.6486	8620.6897	8602.1505
8592.9108	8556.1497	8547.0085	8510.6383	8483.5631
8474.5763	8465.6085	8456.6596	8421.0526	8403.3613
8385.7442	8333.3333	8281.5735	8264.4628	8255.9340
8230.4527	8205.1282	8196.7213	8163.2653	8130.0813
8113.5903	8105.3698	8097.1660	8088.9788	8080.8081
8064.5161	8000.0000	7976.0718	7944.3893	7936.5079
7905.1383	7843.1373	7812.5000	7804.8780	7797.2710
7774.5384	7751.9380	7736.9439	7729.4686	7714.5612
7692.3077	7662.8352	7655.5024	7619.0476	7590.1328
7575.7576	7561.4367	7547.1698	7532.9567	**7518.7970**
7469.6545	7441.8605	7421.1503	7407.4074	7400.5550
7386.8883	7352.9412	7326.0073	7285.9745	7272.7273
7259.5281	7246.3768	7187.7808	7168.4588	7142.8571
7136.4853	7130.1248	7111.1111	7098.4916	7092.1986
7054.6737	7017.5439	6993.0070	6956.5217	6944.4444
6926.4069	6902.5022	6896.5517	6861.0635	6820.1194
6808.5106	6802.7211	6791.1715	6779.6610	6734.0067
6688.9632	6683.3751	6666.6667	6611.5702	6578.9474
6568.1445	6562.7564	6557.3770	6535.9477	6530.6122
6493.5065	6456.8200	6451.6129	6441.2238	6410.2564
6400.0000	6379.5853	6349.2063	6324.1107	6289.3082
6274.5098	6269.5925	6250.0000	6245.1210	6211.1801
6172.8395	6163.3282	6153.8462	6144.3932	6102.2121
6060.6061	6046.8632	6037.7358	5997.0015	5961.2519
5952.3810	5925.9259	5895.3574	5865.1026	5847.9532
5818.1818	5797.1014	5772.0058	5747.1264	5714.2857
5702.0670	5681.8182	5649.7175	5614.0351	5610.0982
5555.5556	5521.0490	5517.2414	5464.4809	5434.7826
5423.7288	5376.3441	5333.3333	5291.0053	5245.9016
5208.3333	5161.2903	5079.3651	**5000.0000**	

Set_CAN_Reg

Set_CAN_Reg schreibt einen Wert in ein Register des gewählten CAN-Controllers auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
Set_CAN_Reg(module,channel,regno,value)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
regno	Register-Nummer (0…255) des CAN-Controllers.	LONG
value	Wert (0…255), der in das Controller-Register geschrieben werden soll.	LONG

Bemerkungen

Sie finden die Registernummern des CAN-Controllers AN82527 im Intel®-Datenblatt.

Siehe auch

Init_CAN, Set_CAN_Baudrate, Get_CAN_Reg

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  Init_CAN(1,1)              'Initialisierung des CAN-Controllers
  Set_CAN_Reg(1,1,0,1)       'Setze Control-Register auf den Wert 1
```

Transmit

Transmit sendet die Nachricht in **CAN_Msg** über ein bestimmtes Message-Objekt.

Syntax

```
#Include ADwinPro_All.Inc
Transmit(module,channel,msg_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
msg_no	Nummer (1… 14) des Message-Objektes im CAN-Controller.	LONG

Bemerkungen

Um eine Nachricht zu senden, müssen Sie folgende Reihenfolge einhalten:

- Einmal: Geben Sie das Message-Objekt mit **En_Transmit** zum Senden frei.
- Sooft erforderlich: Geben Sie die Nachricht in das Feld **CAN_MSG** ein: Die Datenbytes und die Anzahl der Datenbytes.
- Vor dem Senden: Fragen Sie mit **Transmit_Status** ab, ob das Message-Objekt zum Senden bereit ist.
- Senden Sie die Nachricht mit **Transmit**.

Die CAN-Schnittstelle sendet die Nachricht, sobald das Message-Objekt Zugriffsrecht auf den CAN-Bus hat.

Siehe auch

CAN_Msg, En_Receive, En_Transmit, Read_Msg, Transmit_Status

Gültig für

CAN-1, CAN-2

Beispiel

```
Rem Sendet eine 32 Bit-Float-Zahl (hier: Pi) als Folge von
Rem 4 Bytes in einem Message-Objekt
Rem (Empfangen einer Fließkomma-Zahl siehe Bsp. bei Read_Msg)

#Include ADwinPro_All.Inc
#Define pi 3.14159265
Dim i As Long

Init:
  Init_CAN(1,1)              'CAN-Controller initialisieren

  Rem Initialisiere das Message-Objekt 6
  Rem zum Senden von CAN-Nachrichten mit dem Identifier 40
  En_Transmit(1,1,6,40,0)

  Rem Bitmuster von Pi mit Datenformat Long erzeugen
  Par_1 = Cast_FloatToLong(pi)

  Rem Bitmuster (32 Bit) in 4 Bytes aufteilen
  CAN_Msg[4] = Par_1 And 0FFh 'LSB zuweisen
  For i = 1 To 3
    CAN_Msg[4-i] = Shift_Right(Par_1,8*i) And 0FFh
  Next i
  CAN_Msg[9] = 4             'Länge der Nachricht in Bytes

Event:
  Transmit(1,1,6)            'Message-Objekt 6 senden
```

Transmit_Status

Transmit_Status gibt zurück, ob ein Message-Objekt bereit ist zum Senden.

Syntax

```
#Include ADwinPro_All.inc
ret_val = Transmit_Status(module, channel, msg_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
msg_no	Nummer (1… 14) des Message-Objektes im CAN-Controller	LONG
ret_val	Status des Message-Objekts. 0: Bereit zum Senden. 1: Nicht bereit zum Senden.	LONG

Bemerkungen

Der Rückgabewert ist nur für solche Message-Objekte sinnvoll, die zum Senden konfiguriert sind.

Ein Message-Objekt, das nicht bereit zum Senden ist, enthält noch eine Nachricht, die gesendet werden soll oder gerade gesendet wird.

Die CAN-Schnittstelle sendet die Nachricht, sobald das Message-Objekt Zugriffsrecht auf den CAN-Bus hat.

Sie können Nachrichten auch verschicken, ohne vorher den Status des Message-Objekts abzufragen. Wenn Sie jedoch Nachrichten schneller bereitstellen als der CAN-Controller sie verschicken kann, gehen einzelne Nachrichten verloren.

Siehe auch

CAN_Msg, En_Receive, En_Transmit, Read_Msg, Transmit

Gültig für

CAN-1, CAN-2

Beispiel

```
#Include ADwinPro_All.inc

Init:
  Init_CAN(1,1)              'CAN-Controller initialisieren
  En_Transmit(1,1,6,40,0)    'Message-Objekt 6 initialisieren
  Par_1 = 0
  CAN_Msg[1] = Par_1         'Wert setzen
  CAN_Msg[9] = 1             'Länge der Nachricht in Bytes

Event:
  Inc(Par_1)
  CAN_Msg[1] = Par_1         'Wert setzen
  If (Transmit_Status(1,1,6) = 0) Then 'bereit zum Senden?
    Transmit(1,1,6)          'Message-Objekt 6 senden
  EndIf
  If (Par_1 = 255) Then Par_1 = 0
```

3.5.3 Feldbus

Dieser Abschnitt beschreibt folgende Befehle:

Changed_Data

Changed_Data überprüft, ob sich auf dem angegebenen Modul seit dem letzten Zugriff der Applikation auf das DP-RAM die Daten im Ausgangsbereich geändert haben.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Changed_Data(module,outsize)
```

Parameter

module	Eingestellte Moduladresse (1…4).	LONG
outsize	Größe des zu prüfenden Ausgangsbereichs in Byte.	LONG
ret_val	Merker für Datenänderungen im Ausgangsbereich: 0: Daten sind unverändert. 0: neue Daten vorhanden.	LONG

Bemerkungen

Diese Anweisung kann verwendet werden um Rechenzeit zu sparen, indem man nur dann Daten ausliest, wenn sie sich geändert haben.

Der Merker, der dieser Anweisung zu Grunde liegt, wird zurückgesetzt, wenn das Zugriffsrecht auf den Ausgangsbereich von der Applikation zur Feldbusseite wechselt. Dies ist unabhängig davon, ob die Anweisung **Changed_Data** aufgerufen wurde oder nicht.

Die Anweisung kann nur genutzt werden, wenn sie während der Initialisierung freigegeben wurde (siehe auch **Init_Slave** / **Init_Slave_Profibus**).

Gültig für

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Check_Access, Get_Pro_Byte, Get_Read_Buffer, Init_Slave, Request_Access, Request_Release_Access, Set_Pro_Byte, Set_Write_Buffer

Beispiel

```
#Include ADwinPro_All.Inc
Rem Das Beispiel setzt eine Initialisierung voraus

Event:
  Request_Access(1,2)       'Zugriff auf DP-RAM beantragen
                            '(Ausgangsbereich)
  Do
    Par_1 = Check_Access(1) 'Status der Zugriffsrechte lesen
  Until (Par_1 <> -1)
  If (Par_1 = 2) Then       'Wenn Zugriffsrecht erteilt ist …
    If (Changed_Data(1,10)<>0) Then 'und neue Daten vorhanden …
      Par_2 = Get_Pro_Byte(1,10) 'ein Byte lesen
    EndIf
  EndIf
  Request_Release_Access(1,2) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „Datenaustausch mit dem Feldbus (Pro I)“ auf Seite 220.

Check_Access

`Check_Access` gibt zurück, auf welche Bereiche des DP-RAM im angegebenen Modul die Applikation das Zugriffsrecht hat.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Check_Access(module)
```

Parameter

module	Eingestellte Moduladresse (1…4).	LONG
ret_val	Status des Zugriffsrechts: -1: Zugriffsanfrage ist noch nicht bearbeitet. ≥0: Bitmuster für den Zugriffsstatus der Bereiche. Bit = 0: Applikation hat kein Zugriffsrecht. Bit = 1: Applikation hat Zugriffsrecht.	LONG

Bitnr.	31:3	2	1	0
Datenbereich	–	Eingang	Ausgang	Kontroll-Register

Bemerkungen

Wenn das Zugriffsrecht für einen Datenbereich des DP-RAM nicht bei der Applikation liegt, kann auf diesen Bereich nicht zugegriffen werden. Dies ist erforderlich, weil sonst die Daten inkonsistent werden und das System in einen undefinierten Zustand geraten kann.

Das Zugriffsrecht kann erst wieder beantragt werden, wenn eine vorherige Zugriffsanfrage abgearbeitet ist.

Gültig für

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Changed_Data, Get_Pro_Byte, Get_Read_Buffer, Init_Slave, Request_Access, Request_Release_Access, Set_Pro_Byte, Set_Write_Buffer

Beispiel

```
#Include ADwinPro_All.Inc
Rem Das Beispiel setzt eine Initialisierung voraus

Event:
  Request_Access(1,1)       'Zugriff auf DP-RAM beantragen
                            '(Kontrollregister)
  Do
    Par_1 = Check_Access(1) 'Status der Zugriffsrechte lesen
  Until (Par_1 <> -1)
  If (Par_1 = 1) Then       'Leserecht erteilt?
    Par_2 = Get_Pro_Byte(1,7F6h)'ein Byte lesen
  EndIf
  Request_Release_Access(1,1)'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „Datenaustausch mit dem Feldbus (Pro I)“ auf Seite 220.

Get_Pro_Byte

Get_Pro_Byte gibt ein Byte aus einer bestimmten Speicheradresse des DP-RAM des Feldbus-Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Get_Pro_Byte(module,byteno)
```

Parameter

module	Eingestellte Moduladresse (1…4).	LONG
byteno	Speicheradresse im DP-RAM.	LONG
ret_val	Inhalt der Speicheradresse aus dem DP-RAM (0…255).	LONG

Bemerkungen

Die Anweisung kann auf jede Speicherstelle zugreifen, unabhängig davon, ob die Applikation Zugriffsrecht hat oder nicht. Wenn die Applikation Daten ohne Zugriffsrecht ausliest, können fehlerhafte Daten übertragen oder ein Busfehler erzeugt werden.

Achten Sie daher darauf, die Anweisung nicht auf einen unerlaubten Bereich oder zu einem unerlaubten Zeitpunkt verwenden.

Gültig für

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Changed_Data, Check_Access, Get_Read_Buffer, Init_Slave, Request_Access, Request_Release_Access, Set_Pro_Byte, Set_Write_Buffer

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Rem Inhalt von Byte-Nummer 7CDh (Feldbustyp) lesen
  Par_1 = Get_Pro_Byte(1,7CDh)
```

Get_Read_Buffer

`Get_Read_Buffer` kopiert einen definierten Datenblock aus dem Speicherbereich des DP-RAM in das angegebene Zielfeld.

Syntax

```
#Include ADwinPro_All.Inc
Get_Read_Buffer(module,array[],start,count)
```

Parameter

`module`	Eingestellte Moduladresse (1…4).	LONG
`array[]`	Name des Zielfelds.	LONG
		LONG
`start`	Index (0…244), der sowohl die erste gelesene Speicherstelle im DP-RAM bestimmt als auch das erste beschriebene Element im Zielfeld.	LONG
`count`	Anzahl der aus dem DP-RAM gelesenen Werte.	LONG

Bemerkungen

Der Befehl darf nur genutzt werden, wenn das Zugriffsrecht für den entsprechenden Teil des DP-RAM bei der Applikation liegt.

Das Zielfeld muss bereits deklariert sein und zwar mindestens mit `start` + `count` Feldelementen. Der erste Byte wird aus der Adresse `start` im DP-RAM gelesen, während das erste beschriebene Feldelement `array[start+1]` ist.

Das Datenbyte wird jeweils in die Bits 0…7 eines Feldelements (das unterste Byte) geschrieben, die übrigen Bits sind 0.

Gültig für

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Changed_Data, Check_Access, Get_Pro_Byte, Init_Slave, Request_Access, Request_Release_Access, Set_Pro_Byte, Set_Write_Buffer

Beispiel

```
#Include ADwinPro_All.Inc
Dim Data_1[100] As Long
Rem Das Beispiel setzt eine Initialisierung voraus

Event:
  Request_Access(1,2)       'Zugriff auf DP-RAM beantragen
                            '(Ausgangsbereich)
  Do
    Par_1 = Check_Access(1) 'Status der Zugriffsrechte lesen
  Until (Par_1 <> -1)
  If (Par_1 = 2) Then       'Wenn Zugriffrecht erteilt ist...
    If (Changed_Data(1,10) <> 0) Then'und neue Daten vorhanden…
      Get_Read_Buffer(1,Data_1,200h,10) '10 Bytes lesen
    EndIf
  EndIf
  Request_Release_Access(1,2) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „Datenaustausch mit dem Feldbus (Pro I)“ auf Seite 220.

Init_Slave

Init_Slave initialisiert den Feldbus-Slave und ist nur nach einem Einschalten (power up) möglich.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Init_Slave(module, IO_in, Par_in, io_out,
    Par_out, in_hdl, out_hdl, reserved)
```

Parameter

`module` — Eingestellte Moduladresse (1…4). LONG

`IO_in` — Größe des Eingangs-Datenbereichs für zyklische Daten. LONG

Inter-SL: 1…10 Worte (1 Wort = 2 Byte).
andere Module: 0…244 Byte
(`IO_in` + `IO_out`: 1…416 Byte).

`Par_in` — Länge des Eingangs-Datenbereichs für azyklische Daten. LONG
Inter-SL (Angabe in Worten):
0: Modul arbeitet als digitaler Slave.
1…200; Standard 32. Modul arbeitet als PCP-Teilnehmer.

andere Module: 0 (azyklische Daten nicht unterstützt).

`IO_out` — Länge des Ausgangs-Datenbereichs für zyklische Daten. LONG

Inter-SL: 1…10 Worte (1 Wort = 2 Byte).
andere Module: 0…244 Byte
(`IO_in` + `IO_out`: 1…416 Byte).

`Par_out` — Länge des Ausgangs-Datenbereichs für azyklische Daten. LONG
Inter-SL (Angabe in Worten):
0: Modul arbeitet als digitaler Slave.
1…200; Standard 32. Modul arbeitet als PCP-Teilnehmer.

andere Module: 0 (azyklische Daten nicht unterstützt).

`in_hdl` — Bitmuster zur Handhabung des Eingangs-Datenbereichs: LONG

Bitnr.	31:2	1	0
Bit = 0	–	Merker deaktivieren für **Changed_Data**.	Eingangsbereich löschen, sobald die Applikation stoppt.
Bit = 1	–	Merker aktivieren für **Changed_Data**.	Eingangsbereich einfrieren, sobald die Applikation stoppt.

out_hdl — Bitmuster zur Handhabung des Ausgangs-Datenbereichs: LONG

Bitnr.	31:2	1	0
Bit = 0	–	Ausgangsbereich löschen, sobald der Feldbus stoppt.	Feldbus Offline starten.
Bit = 1	–	Ausgangsbereich einfrieren, sobald der Feldbus stoppt.	Feldbus Online starten.

reserved — Immer 0 angeben. Der Parameter ist nur aus Kompatibilitätsgründen erforderlich. LONG

ret_val — Bitmuster als Status der Initialisierung: LONG
Bit = 0: kein Fehler.

Bit = 1: Fehler aufgetreten (Bedeutung s.u.).

Bitnr.	31:16	15	14	13:8	7	6	5:3	2	1	0
Fehler	–	A_7	A_6	–	A_5	A_4	–	A_3	A_2	A_1

A_1: Fehler bei Hardware-Überprüfung.

A_2: Fehler beim Start der Initialisierung.

A_3: Fehler bei der Initialisierung.

A_4: Fehler im DP-RAM.

A5: Fehler beim Abschluss der Initialisierung.

A_6: Keine Nachricht empfangen.

A_7: Keine Nachricht gesendet.

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem Slave-Modul ausgeführt werden.

Eine zweite Initialisierung nach dem Einschalten ist nicht möglich.

Während der Initialisierung mit **Init_Slave** wird die Größe des Ein- und Ausgangsbereichs festgelegt (getrennt für zyklische und azyklische Daten). Die Größe muss mit der projektierten Größe im zugehörigen Master übereinstimmen. Die maximale Größe der einzelnen Bereiche ist je nach Feldbus unterschiedlich; Angaben finden Sie im Hardware-Handbuch.

Nach der erfolgreichen Initialisierung ist der Slave bereit, am Datenverkehr teilzunehmen und es können Parameter aus dem DP-RAM ausgelesen werden. Die Applikation darf nur Informationen ins DP-RAM schreiben, wenn sie die Zugriffsrechte dazu besitzt!

Wurden bei der ersten Initialisierung die falschen Daten übergeben, muss das System ausgeschaltet werden. Erst nach einem erneuten Einschalten kann das Modul neu initialisiert werden.

Die Anweisung löst eine Ablaufkette aus, die etwa 2-3 Sekunden dauert. Wenn die Anweisung aus einem (nicht unterbrechbaren) hochprioren Prozess aufgerufen wird, kann in dieser Zeit keine Kommunikation zwischen PC und *ADwin*-System stattfinden. Es ist deshalb empfehlenswert, die Initialisierung aus einem niedrigprioren Prozess oder einem Prozessabschnitt mit niedriger Priorität (z.B. **LowInit:**) heraus aufzurufen.**Gültig für**

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Changed_Data, Check_Access, Get_Pro_Byte, Get_Read_Buffer, Request_Access, Request_Release_Access, Set_Pro_Byte, Set_Write_Buffer

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  Rem Slave initialisieren:
  Rem Nur zyklische Daten (10 IO_in / 10 IO_out),
  Rem CHANGE_DATA-Funktion ein, Ausgänge werden gelöscht, wenn
  Rem Bus OFF-Line.
  Par_1 = Init_Slave(1,10,0,10,0,2,0,0)
```

Request_Access

Request_Access beantragt das Zugriffsrecht auf das DP-RAM des Slaves für die Applikation.

Syntax

```
#Include ADwinPro_All.Inc
Request_Access(module,area)
```

Parameter

module	Eingestellte Moduladresse (1…4).	LONG
area	Bitmuster für den Bereich, für den der Zugriffsstatus geändert wird: Bit = 0: Zugriffsrecht bleibt unverändert. Bit = 1: Zugriffsrecht wird beantragt.	LONG

Bitnr.	31:3	2	1	0
Bereich	–	Eingang	Ausgang	Kontroll-Register

Bemerkungen

Wenn das Zugriffsrecht für mehrere Bereiche gleichzeitig beantragt wird, kann es sein, dass die Feldbusseite den Zugriff auf einen Teilbereich verweigert. Dann kann auf die anderen Bereiche trotzdem zugegriffen werden.

Liegt das Zugriffsrecht für das DP-RAM nicht bei der Applikation, darf nicht auf den Bereich zugegriffen werden, da die Daten inkonsistent werden und das System in einen undefinierten Zustand kommen kann.

Nach dem Datenaustausch mit dem DP-RAM sollte das Zugriffsrecht mit **Request_Release_Access** wieder an die Feldbusseite zurückgegeben werden, damit die Daten an den Master weitergeleitet und die Daten vom Master in das DP-RAM geschrieben werden können. Gibt die Applikation das Zugriffsrecht nicht aktiv an die Busseite zurück, fällt es nach 1 Sekunde automatisch zurück.

Das Zugriffsrecht kann erst wieder beantragt werden, wenn eine vorherige Zugriffsanfrage abgearbeitet ist.

Gültig für

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Changed_Data, Check_Access, Get_Pro_Byte, Set_Pro_Byte, Get_Read_Buffer, Init_Slave, Set_Write_Buffer, Request_Release_Access

Beispiel

```
#Include ADwinPro_All.Inc
Rem Das Beispiel setzt eine Initialisierung voraus

Event:
  Request_Access(1,1)       'Zugriff auf DP-RAM beantragen
                            '(Kontroll-Register)
  Do
    Par_1 = Check_Access(1) 'Status der Zugriffsrechte lesen
  Until (Par_1 <> -1)
  If (Par_1 = 1) Then       'Wenn Zugriffsrecht erteilt ist,
    Par_2 = Get_Pro_Byte(1,7F6h)'ein Byte lesen
  EndIf
  Request_Release_Access(1,1) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „Datenaustausch mit dem Feldbus (Pro I)“ auf Seite 220.

Request_Release_Access

Request_Release_Access beantragt, das Zugriffsrecht auf das DP-RAM des Slaves an den Feldbus zurückzugeben.

Syntax

```
#Include ADwinPro_All.Inc
Request_Release_Access(module,area)
```

Parameter

module	Eingestellte Moduladresse (1…4).	LONG
area	Bitmuster für den Bereich, für den der Zugriffsstatus geändert wird: Bit = 0: Zugriffsrecht bleibt unverändert. Bit = 1: Zugriffsrecht wird zurückgegeben.	LONG

Bitnr.	31:3	2	1	0
Daten-bereich	–	Eingang	Ausgang	Kontroll-Register

Bemerkungen

Das Zugriffsrecht kann für mehrere Bereiche gleichzeitig zurückgegeben werden.

Nach dem Datenaustausch mit dem DP-RAM sollte das Zugriffsrecht wieder an die Feldbusseite zurückgegeben werden, damit die Daten an den Master weitergeleitet und die Daten vom Master in das DP-RAM geschrieben werden können. Gibt die Applikation das Zugriffsrecht nicht aktiv an die Busseite zurück, fällt es nach 1 Sekunde automatisch zurück.

Das Zugriffsrecht kann erst wieder beantragt werden, wenn eine vorherige Zugriffsanfrage abgearbeitet ist.

Gültig für

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Changed_Data, Check_Access, Get_Pro_Byte, Get_Read_Buffer, Init_Slave, Request_Access, Set_Pro_Byte, Set_Write_Buffer

Beispiel

```
#Include ADwinPro_All.Inc
Rem Das Beispiel setzt eine Initialisierung voraus

Event:
  Request_Access(1,1)       'Zugriff auf DP-RAM beantragen
                            '(Kontroll-Register).
  Do
    Par_1 = Check_Access(1) 'Status der Zugriffsrechte lesen
  Until (Par_1 <> -1)
  If (Par_1 = 1) Then       'Wenn Zugriffsrecht erteilt ist,
    Par_2 = Get_Pro_Byte(1,7F6h) 'ein Byte lesen
  EndIf
  Request_Release_Access(1,1) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „Datenaustausch mit dem Feldbus (Pro I)“ auf Seite 220.

Set_Pro_Byte

`Set_Pro_Byte` schreibt einen Wert in eine Speicherstelle des DP-RAM des Feldbus-Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Set_Pro_Byte(module,byteno,value)
```

Parameter

module	Eingestellte Moduladresse (1…4).	LONG
byteno	Speicheradresse im DP-RAM.	LONG
value	Wert, der in die Speicheradresse geschrieben werden soll (0…255).	LONG

Bemerkungen

Die Anweisung kann auf jede Speicherstelle zugreifen, unabhängig davon, ob die Applikation Zugriffsrecht hat oder nicht. Wenn die Applikation Daten ohne Zugriffsrecht ausliest, können fehlerhafte Daten übertragen oder ein Busfehler erzeugt werden.

Achten Sie daher darauf, die Anweisung nicht auf einen unerlaubten Bereich oder zu einem unerlaubten Zeitpunkt verwenden.

Gültig für

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Changed_Data, Check_Access, Get_Pro_Byte, Get_Read_Buffer, Init_Slave, Request_Access, Request_Release_Access, Set_Write_Buffer

Beispiel

```
#Include ADwinPro_All.Inc
Init:
  Rem Byte 0h mit dem Wert 2 beschreiben (zyklische Eingangsdaten)
  Set_Pro_Byte(1,0h,2)
```

Set_Write_Buffer

Set_Write_Buffer kopiert die Daten eines Felds in einen definierten Speicherbereich des DP-RAM.

Syntax

```
#Include ADwinPro_All.Inc
Set_Write_Buffer(module,array[],start,count)
```

Parameter

module	Eingestellte Moduladresse (1…4).	LONG
array[]	Name des Quellfelds.	LONG
		LONG
start	Index (0…244), der sowohl das erste gelesene Element im Quellfeld bestimmt als auch die erste Speicherstelle im DP-RAM.	LONG
count	Anzahl der ins DP-RAM übertragenen Werte.	LONG

Bemerkungen

Der Befehl darf nur genutzt werden, wenn das Zugriffsrecht für den entsprechenden Teil des DP-RAM bei der Applikation liegt.

Das Quellfeld muss bereits deklariert sein und zwar mindestens mit `start` + `count` Elementen. Der erste Wert wird aus `array[start+1]` gelesen, während die erst beschriebene Speicherzelle im DP-RAM die Adresse `start` hat.

Übertragen werden jeweils die Bits 0…7 eines Feldelements (das unterste Byte).

Gültig für

Inter-SL, Profi-DP-SL, Profi-IRT-CU, Profi-IRT-FO

Siehe auch

Changed_Data, Check_Access, Get_Pro_Byte, Get_Read_Buffer, Init_Slave, Request_Access, Request_Release_Access, Set_Pro_Byte

Beispiel

```
#Include ADwinPro_All.Inc
Rem Das Beispiel setzt eine Initialisierung voraus

Dim Data_1[100] As Long

Event:
  Request_Access(1,4)       'Zugriff auf DP-RAM beantragen
                            '(Eingangsbereich)
  Do
    Par_1 = Check_Access(1) 'Status der Zugriffsrechte lesen
  Until (Par_1 <> -1)
  If (Par_1 = 2) Then       'Wenn Zugriffrecht erteilt ist …
    If (Changed_Data(1,10) <> 0) Then'und neue Daten vorhanden:
      Rem 10 Byte schreiben
      Set_Write_Buffer(1,Data_1,200h,10)
    EndIf
  EndIf
  Request_Release_Access(1,4) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „Datenaustausch mit dem Feldbus (Pro I)“ auf Seite 220.

3.5.4 RSxxx

Dieser Abschnitt beschreibt folgende Befehle:

Check_Shift_Reg

Check_Shift_Reg gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO der RSxxx-Schnittstelle geschrieben wurden.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Check_Shift_Reg(module,channel)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer (1, 2 oder 1…4) der Schnittstelle, deren Sende-Status geprüft wird.	LONG
ret_val	Sende-Status: 0: Daten sind gesendet (= keine Daten im Sende-FIFO vorhanden). 1: Noch nicht alle Daten gesendet (= im Sende-FIFO sind noch Daten vorhanden).	LONG

Bemerkungen

Bei dem Rückgabewert 0 ist sowohl das Sende-FIFO als auch das Ausgangs-Shiftregister leer. Bei dem Rückgabewert 1 ist mindestens ein Bit noch nicht gesendet.

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

Get_RS, Read_FIFO, RS_Init, RS_Reset, RS485_Send, Set_RS

Gültig für

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#Include ADwinPro_All.Inc

Event:
  Rem …
  Par_1 = Check_Shift_Reg(1,1)'Prüft, ob Schnittstelle 1 noch
                              'Daten zu senden hat
  Rem …
```

Get_RS

`Get_RS` liest den Inhalt eines bestimmten Controller-Registers auf dem angegebenen Modul aus.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Get_RS(module,reg_no)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
reg_no	Adresse des zu lesenden Controller-Registers.	LONG
ret_val	Inhalt des Controller-Registers.	LONG

Bemerkungen

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

Check_Shift_Reg, Read_FIFO, RS_Init, RS_Reset, RS485_Send, Set_RS

Gültig für

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

- / -

Read_FIFO

`Read_FIFO` liest einen Wert aus dem Eingangs-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Read_FIFO(module,channel)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Nummer des auszulesenden Kanals (1, 2 oder 1…4).	LONG
ret_val	Inhalt des Eingangs-FIFO: -1: FIFO ist leer. ≥0:Übertragener Datenwert.	LONG

Bemerkungen

- / -

Siehe auch

Check_Shift_Reg, Get_RS, RS_Init, RS_Reset, RS485_Send, Set_RS

Gültig für

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  RS_Reset(1)
  RS_Init(1,1,9600,0,8,0,1) 'Initialisierung von Kanal 1 auf Modul
                            '1 mit 9600 Baud, ohne Parität,
                            '8 Datenbits, 1 Stoppbit und
                            'Hardwarehandshake (nur RS232).

Event:
  Par_1 = Read_FIFO(1,1)    'Einen Wert aus dem FIFO holen. Wenn
                            'der FIFO leer ist, wird -1
                            'zurückgeliefert.
```

Siehe auch weitere Beispiele für RS232 und RS485 (Pro I) ab Seite 222.

RS_Init

`RS_Init` initialisiert einen bestimmten Kanal auf dem angegebenen Modul.

Die folgenden Parameter werden gesetzt:

- Übertragungsgeschwindigkeit in Baud
- Anwendung von Prüf-Bits
- Datenlänge
- Anzahl der Stopp-Bits
- Übertragungs-Protokoll (Handshake)

Syntax

```
#Include ADwinPro_All.Inc
RS_Init(module,channel,baud,parity,bits,stop,
        handshake)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`channel`	Kanal, der initialisiert werden soll (1, 2 oder 1…4).	LONG
`baud`	Übertragungsgeschwindigkeit in Baud: RS232: 35 … 115200 Baud. RS422, RS485: 35…2304000 Baud.	LONG
`parity`	Anwendung von Prüf-Bits: 0: ohne Paritäts-Bit. 1: gerade Parität (even). 2: ungerade Parität (odd).	LONG
`bits`	Anzahl der Daten-Bits (5, 6, 7 oder 8).	LONG
`stop`	Anzahl der Stopp-Bits. 0: 1 Stopp-Bit. 1: 1½ Stopp-Bits bei 5 Daten-Bits; 2 Stopp-Bits bei 6, 7 oder 8 Daten-Bits.	LONG
`handshake`	Übertragungs-Protokoll: 0: kein Handshake. 1: Hardware-Handshake (RTS/CTS), nur RS232. 2: Software-Handshake (Xon/Xoff). 3: RS485.	LONG

Bemerkungen

Diese Anweisung ist vor dem ersten Arbeiten mit dem gewählten Kanal notwendig, um die Schnittstellen-Parameter einzustellen. Sie müssen für eine korrekte Übertragung mit der Gegenstelle identisch sein.

Die Initialisierung ist auch dann erforderlich, nachdem Sie auf dem Modul mit `RS_Reset` einen Hardware-Reset ausgeführt haben.

Die Baudrate wird vom Grundtakt (2304000Hz) des moduleigenen Taktgebers abgeleitet. Es ist jede Baudrate einstellbar, die sich durch ganzzahlige Division des Grundtakts ergibt. Der Teiler kann Werte im Bereich 1…0FFFFh annehmen.

Die Baudrate ergibt sich aus der grundlegenden Taktrate von 2304MHz der Schnittstelle, dividiert durch einen ganzzahligen Divisor. Der Wertebereich des Divisors von 1 … 0FFFFh ergibt eine Bandbreite von 35 … 2304 000 Bit/s. Entsprechend der Spezifikation ist die RS232-Schnittstelle auf 115200 Bit/s beschränkt. Die folgende Liste zeigt einige übliche Baudraten.

Übliche Baudraten [Bit/s]		
2304000	57600	2400
1152000	38400	1200
460800	19200	600

Übliche Baudraten [Bit/s]		
230400	9600	300
115200	4800	

Siehe auch

Check_Shift_Reg, Get_RS, Read_FIFO, RS_Reset, RS485_Send, Set_RS

Gültig für

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  RS_Reset(1)                 'RS-Modul zurücksetzen
  RS_Init(1,1,9600,0,8,0,1)'Initialisierung von Kanal 1 auf
                              'Modul 1 mit 9600 Baud, ohne Parität,
                              '8 Datenbits, 1 Stoppbit und
                              'Hardware-Handshake (nur RS232).
```

Siehe auch weitere Beispiele für RS232 und RS485 (Pro I) ab Seite 222.

RS_Reset

`RS_Reset` führt auf dem angegebenen Modul einen Hardware-Reset durch und löscht dabei die Einstellungen für alle Kanäle.

Syntax

```
#Include ADwinPro_All.Inc
RS_Reset(module)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG

Bemerkungen

Die Anweisung sendet einen Reset-Impuls auf den entsprechenden Eingang des Controllers TL16C754. Sie können dem Datenblatt des Controllers 16C754 von Texas Instruments entnehmen, auf welche Werte die Register durch den Hardware-Reset gesetzt werden.

Nach einem Hardware-Reset muss eine Initialisierung mit `RS_Init` folgen, um den Controller zu initialisieren und die gewünschten Schnittstellen-Parameter einzustellen.

Siehe auch

Check_Shift_Reg, Get_RS, Read_FIFO, RS_Init, RS485_Send, Set_RS

Gültig für

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#Include ADwinPro_All.Inc

Init:
  RS_Reset(1)                 'RS-Modul zurücksetzen
  RS_Init(1,1,9600,0,8,0,1)'Initialisierung von Kanal 1 auf
                              'Modul 1 mit 9600 Baud, ohne Parität,
                              '8 Datenbits, 1 Stoppbit und
                              'Hardware-Handshake (nur RS232).
```

Siehe auch weitere Beispiele für RS232 und RS485 (Pro I) ab Seite 222.

RS485_Send

RS485_Send legt die Übertragungsrichtung für einen bestimmten Kanal des angegebenen Moduls fest.

Syntax

```
#Include ADwinPro_All.Inc
RS485_Send(module,channel,dir)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Einzustellender Kanal (1, 2 oder 1…4).	LONG
dir	Übertragungsrichtung des Kanals: 0: Kanal als Empfänger einstellen. 1: Kanal als Sender einstellen. 2: Kanal als Sender einstellen, der gleichzeitig die gesendeten Daten empfängt.	LONG

Bemerkungen

Die Einstellung der Übertragungsrichtung bedeutet:

- Empfänger: Der Kanal kann Daten auf dem Bus ausschließlich lesen, auch wenn Daten im Ausgangs-FIFO des Controllers für diesen Kanal liegen.
- Sender: Der Kanal kann Daten auf den Bus legen, die von anderen Teilnehmern gelesen werden können.
- Sender/Empfänger: Der Kanal kann Daten auf den Bus legen und gleichzeitig zurücklesen. Dadurch ist eine Überprüfung der ausgegebenen Daten möglich.

Siehe auch

Check_Shift_Reg, Get_RS, Read_FIFO, RS_Init, RS_Reset, Set_RS

Gültig für

RS485-2, RS485-4

Beispiel

Siehe Beispiel RS485: Empfangen und senden auf Seite 226.

Set_RS

`Set_RS` schreibt einen Wert in ein bestimmtes Register des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
Set_RS(module,reg_no,value)
```

Parameter

`module`	Eingestellte Moduladresse (1…255).	LONG
`reg_no`	Nummer des zu beschreibenden Registers.	LONG
`value`	Wert, der in das Register geschrieben werden soll.	LONG

Bemerkungen

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers: TL16C754 von Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

Check_Shift_Reg, Get_RS, Read_FIFO, RS_Init, RS_Reset, RS485_Send

Gültig für

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

- / -

Write_FIFO

`Write_FIFO` schreibt einen Wert in den Sende-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Write_FIFO(module,channel,value)
```

Parameter

module	Eingestellte Moduladresse (1…255).	LONG
channel	Kanalnummer, dessen Sende-FIFO beschrieben wird (1, 2 oder 1…4).	LONG
value	Wert der ins Sende-FIFO geschrieben werden soll.	LONG
ret_val	Statusmeldung: 0: Daten wurden erfolgreich geschrieben. 1: Daten konnten nicht geschrieben werden, Sende-FIFO ist voll.	LONG

Bemerkungen

Die Anweisung prüft zuerst, ob noch mindestens ein Speicherplatz im Sende-FIFO frei ist. Ist dies der Fall, wird der übergebene Wert ins FIFO geschrieben (Rückgabewert 0); anderenfalls wird eine 1 zurückgeliefert, die angibt, dass das FIFO voll ist und ein Schreiben nicht möglich war.

Der zu übertragende Wert `value` kann auch ein einzelnes ASCII-Zeichen oder ein ASCII-Befehl sein (Zeichen werden intern mit dem Datentyp Long gleich gesetzt). Die Hardware-Dokumentation enthält ein Beispiel für das Senden einer Zeichenfolge.

Siehe auch

Check_Shift_Reg, Get_RS, Read_FIFO, RS_Init, RS_Reset, RS485_Send, Set_RS

Gültig für

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#Include ADwinPro_All.Inc
Dim val As Long

Init:
  RS_Reset(1)
  RS_Init(1,1,9600,0,8,0,1)'Initialisierung von Kanal 1 auf
                           'Modul 1 mit 9600 Baud, keine Parität,
                           '8 Datenbits, 1 Stoppbit und
                           'Hardware-Handshake (nur RS232).

Event:
  Par_1 = Write_FIFO(1,1,val)
  Rem Wenn das FIFO-Feld nicht voll ist, wird val ins FIFO-Feld
  Rem geschrieben. Anderenfalls enthält Par_1 den Wert 1 und zeigt
  Rem damit an, dass das FIFO-Feld nicht beschrieben werden konnte
  Rem (FIFO voll).
```

Siehe auch weitere Beispiele für RS232 und RS485 (Pro I) ab Seite 222.

4 Programmbeispiele

Folgende Beispiele stehen zur Verfügung:

Die meisten Beispiele sind auch als kompilierte Programmdateien im Verzeichnis <C:\ADwin\ADbasic> unter samples_ADwin_Pro\ bzw. samples_ADwin_ProII\ abgelegt.

4.1 Online-Auswertung von Messwerten (Pro I)

Das Programm <PRO_DMO1.BAS> sucht den Maximal- und Minimalwert aus 1000 Messungen von ADC1 und schreibt das Ergebnis in die Variablen **Par_1** und **Par_2**.

Benötigt wird ein 16-Bit-A/D-Modul mit Moduladresse 1 (Gruppe AD-Module) und ein Signal am Eingang 1 des Moduls.

```
#Include ADwinPro_All.Inc  'Include file
#Define limit 65535        'max. 16 bit ADC-value
Dim i1, iw, max, min As Long

Init:
  i1 = 1                   'reset sample counter
  max = 0                  'initial maximum value
  min = limit              'initial minimum value
  Par_10 = 0               'init End-Flag
  Processdelay = 40000     'cycle-time of 1ms (T9)

Event:
  iw = ADC16(1,1)          'get sample
  If (iw > max) Then max = iw 'new maximum sample?
  If (iw < min) Then min = iw 'new minimum sample?
  Inc i1                   'increment index
  If (i1 > 1000) Then      '1000 samples done?
    i1 = 1                 'reset index
    Par_1 = min            'write minimum value
    Par_2 = max            'write maximum value
    max = 0                'reset minimum value
    min = 65535            'reset maximum value
    Par_10 = 1             'set End-Flag
  EndIf
```

4.2 Digitaler Proportional-Regler (Pro I)

Das Programm <PRO_DMO2.BAS> ist ein digitaler P-Regler. Der Sollwert wird mit Par_1 vorgegeben, die Verstärkung mit Par_2.

Benötigt werden:

– 16-Bit-A/D-Modul mit Moduladresse 1 (Gruppe AD-Module).

– D/A-Modul mit Moduladresse 1 (Gruppe DA-Module).

– Eine Regelstrecke, die das Signal des D/A-Moduls aufnimmt und ein Signal an das A/D-Modul zurückgibt.

```
#Include ADwinPro_All.Inc  'Include file
#Define offset 32768       '0V for 16 bit ADC/DAC-systems

Rem cd: control deviation; av: actuating value
Dim cd, av As Long

Init:
  Par_1 = offset           'initial setpoint
  Par_2 = 10               'initial gain
  Processdelay = 40000     'cycle-time of 1ms (T9)

Event:
  cd = Par_1 - ADC16(1, 1)'compute control deviation (cd)
  av = cd * Par_2 + offset'compute actuating value (av)
  DAC(1, 1, av)            'output actuating value on DAC-#1
```

4.3 Datenaustausch mit DATA-Feldern (Pro I)

Das Programm <PRO_DMO3.BAS> misst den analogen Eingang 1 des A/D-Moduls mit Adresse 1 und setzt nach 1000 Messungen eine Ende-Markierung, damit der PC erkennt, wann er die Messwerte abholen kann. Die Daten werden mit Hilfe des Felds Data_1 übertragen.

Benötigt wird ein 16-Bit-A/D-Modul mit Moduladresse 1 (Gruppe AD-Module).

```
#Include ADwinPro_All.Inc  'Include file
Dim Data_1[1000] As Long
Dim index As Long

Init:
 Par_10 = 0
 index = 0                 'reset array pointer
 Processdelay = 40000      'cycle-time of 1ms (T9)

Event:
 index = index + 1         'increment array pointer
 If (index > 1000) Then    '1000 samples done?
  Par_10 = 1               'set End-Flag
  End                      'terminate process
 EndIf
 Data_1[index] = ADC16(1, 1)  'acquire sample and save in array
```

4.4 Digitaler PID-Regler (Pro I)

Das Programm <PRO_DMO6.BAS> ist ein digitaler PID-Regler.

Vor dem Starten des Reglers müssen die Reglereinstellungen in den globalen Variablen stehen.

Benötigt werden:

- A/D-Modul mit Moduladresse 1 (Gruppe AD-Module).
- D/A-Modul mit Moduladresse 1 (Gruppe DA-Module).
- Eine Regelstrecke, die das Signal des D/A-Moduls aufnimmt und ein Signal an das A/D-Modul zurückgibt.

Berechnungen im PC:

Die Reglerkoeffizienten, der Sollwert und die Abtastrate werden auf dem PC berechnet und in globalen Variablen an den Prozessor des *ADwin-Pro*-Systems übergeben. Auf dem gleichen Weg werden Informationen aus dem Programm an den PC zurück gegeben:

Einstellparameter des Reglers	
FPar_2	Verstärkung des Reglers
FPar_3	Integrationszeit des Reglers
FPar_4	Differentationszeit des Reglers
Par_1	Sollwert in Digits
Par_6	Abstastrate des Reglers in Einheiten zu 25ns
Informationen aus dem Programm	
Par_5	Feld-Index (zu **Data_1**) der Regelabweichung
Par_9	Mittlere Regelabweichung
Par_10	Flag: Alle Messungen sind durchgeführt
Data_1[]	Feld, das die Regelabweichungen enthält

ADbasic-Programm:

Sowohl die Adresse des analogen Eingangsmoduls als auch die Adresse des analogen Ausgangsmoduls sind im Beispiel auf Adresse 1 eingestellt.

Beachten Sie, dass im Programm eine Zeitersparnis dadurch erreicht wird, dass während den erforderlichen Wartezeiten beim Einlesen der Regelabweichung (nach **Set_Mux** und **Start_Conv**) die Berechnung und Ausgabe des Stellwerts durchgeführt wird.

Als Folge ergibt sich, dass jeweils der ausgegebene Stellwert aus der Regelabweichung des vorigen Prozessaufrufs berechnet wird.

```
#Include ADwinPro_All.Inc  'Include file

#Define offset 32768       '0V output

Dim Data_1[4000] As Long
Dim av, cd, cdo, sum As Long
Dim diff As Float

Init:
  sum = 0                  'initial value of integral part
  cd = ADC(1)              'initial value of control deviation
                           '(cd) & MUX to Ch-#1
  Par_5 = 1                'set array index
  If (FPar_3 < 1E3) Then FPar_3 = 1E3'check min. of integration
                           'time
  If (Par_6 < 4E4) Then Par_6 = 4E4'allow cycle-times >= 1ms
  Processdelay = Par_6     'set cycle-time

Event:
  Rem compute actuating value
  av = FPar_2 * (cd + sum / FPar_3 + diff * FPar_4)
  Start_Conv(1)            'start conversion ADC-#1
  Rem while conversion is running …
  DAC(1, 1, av + offset)   'output actuating value at DAC-#1
  cdo = cd                 'keep control deviation in mind
  Wait_EOC(1)              'wait until End-of-conversion of ADC
  cd = Par_1 - ReadADC(1)  'compute control deviation
  FPar_9 = FPar_9 * 0.99 + cd * 0.01'mean value of control
                           'deviation
  sum = sum + cd           'calculate integral
  If (sum > 2E6) Then sum = 2E6 'positive limit of integral
  If (sum < -2E6) Then sum = -2E6 'negative limit of integral
  diff = (cd - cdo)        'calculate deviation difference
  Data_1[Par_5] = cd       'write control deviation in a buffer
  Inc Par_5                'increment buffer index
  If (Par_5 >= 4000) Then  '4000 samples done?
    Par_10 = 1             'set End-flag
    Par_5 = 1              'reset array index
  EndIf

Finish:
  DAC(1, 1, offset)        'analog output #1 to 0V
```

4.5 Datenaustausch mit dem Feldbus (Pro I)

Das folgende Programm tauscht Daten mit dem Feldbus aus.

Das Programm fordert zyklisch (zeitgesteuert) den Zugriff auf das DP-RAM an, überprüft das Zugriffsrecht, tauscht die Daten aus und gibt den Zugriff wieder an die Busseite zurück. Vor dem eigentlichen Austausch der Daten überprüft das Programm, ob sich die Daten geändert haben und nur in diesem Fall werden sie ausgelesen und neu geschrieben.

Benötigt werden:

– Feldbusmodul mit Moduladresse 1 (Gruppe EXT-Module).
– Ein Gerät oder Programm am Feldbus, das Daten empfangen und senden kann.

```
#Include ADwinPro_All.inc
#Define num_data 10          'Anzahl Eingangs- und Ausgangsdaten
#Define module 1             'Moduladresse
Dim Data_1[num_data] As Long 'Feld für Ausgangsdaten
Dim Data_2[num_data] As Long 'Feld für Eingangsdaten
Dim temparr[num_data] As Long 'Feld für temporäre Eingangsdaten
Dim i, seq_step As Long
Dim range, ret_val As Long

Init:
  Rem Die Taktrate des Prozesses muss mindestens 10Hz betragen
  Processdelay = 100000      '400 Hz für Prozessor T9
  range = 6                  'Speicherbereich, auf den der Zugriff
                             'erfolgen soll (Ein-/Ausgangsdaten)
  seq_step = 0               'aktueller Abschnitt der
                             'Ablaufsteuerung: neu starten
  Rem initialisieren, für Profinet 2. Zeile verwenden
  ret_val = Init_Slave(module, num_data, 0, num_data, 0, 2, 2, 0)
  Rem ret_val = Init_Slave_Profinet(module, num_data, 0,
    num_data, 0, 2, 2, 0)
  If (ret_val <> 0) Then Exit 'Initialisierungsfehler
  For i = 1 To num_data      'Sendedaten initialisieren
    Data_1[i] = i
  Next i

Event:
  SelectCase seq_step
  Case 0                     'neu starten
    Request_Access(module, range)'Zugriffsrecht beantragen
    seq_step = 1
  Case 1                     'Zugriffsrecht erteilt?
    ret_val = Check_Access(module)
    If (ret_val <> -1) Then 'wurde Anfrage bereits bearbeitet?
      If (ret_val = range) Then 'Wenn Zugriffsrecht erteilt...
        seq_step = 2         '...Daten austauschen
      Else
        seq_step = 0         '...sonst abbrechen und neu starten
      EndIf
    EndIf
  Case 2                     'Daten mit Feldbus austauschen
    Set_Write_Buffer(module, Data_1, 1, num_data) 'Daten senden
    Get_Read_Buffer(module, temparr, 200h, num_data) 'Daten lesen
    ret_val = Check_Access(module)
    Rem Wenn Zugriffsrecht besteht, sind gelesene Daten gültig
    If (ret_val = range) Then
      For i = 1 To num_data  'Empfangsdaten umkopieren
        Data_2[i] = temparr[i]
      Next i
      Request_Release_Access(module,range) 'Zugriffsrecht zurück
      seq_step = 3
    Else 'sonst abbrechen und neu starten, Daten verwerfen
      seq_step = 0
    EndIf
  Case 3                     'Zugriffsrecht freigegeben?
    ret_val = Check_Access(module)
    If(ret_val <> -1)Then    'wurde Anfrage bereits bearbeitet?
      If (ret_val = 0) Then  'Wenn Zugriffsrecht freigegeben...
        seq_step = 0         '...neu starten
      Else                   '...sonst nochmals zurückgeben
```

```
    Request_Release_Access(module, range)
   EndIf
  EndIf
 EndSelect
```

4.6 Beispiele für RS232 und RS485 (Pro I)

Die folgenden Beispiele sind vollständige (Pro I-) Programme für das Senden und Empfangen von Daten und Strings mit RS232 oder RS485.

RS232: Empfangen und senden

Benötigt wird ein RS232-Modul mit Moduladresse 1 (Gruppe EXT-Module).

Das Programm zeigt die Initialisierung der seriellen RS232-Schnittstelle im Abschnitt **Init:** und das zyklische Lesen und Schreiben von Daten im Abschnitt **Event:**. Der Prozess ist zeitgesteuert.

```
Rem Das Programm initialisiert die seriellen Schnittstellen im
Rem Abschnitt Init:
Rem Im Abschnitt Event: werden Daten zwischen den Schnittstellen
Rem 1 & 2 des RS-Moduls ausgetauscht.
Rem Mit Hilfe dieses Programms können die Schnittstellen
Rem untereinander getestet werden. Dazu müssen Sie die
Rem Schnittstellen vor dem Programmstart miteinander verbinden.

#Include ADwinPro_All.Inc
#Define num_data 1000      'Anzahl Sende- und Empfangsdaten
#Define module 1           'Moduladresse
Dim Data_1[num_data] As Long'Sendedaten
Dim Data_2[num_data] As Long'Empfangsdaten
Dim lauf As Long           'Laufvariable

Init:
 For lauf = 1 To num_data 'Initialisierung der Sendedaten
  Data_1[lauf] = lauf And 0FFh
 Next lauf
 RS_Init(module,1,9600,0,8,1,0) 'Schnittst. 1 initialisieren:
                           '9600 Baud;
                           'Kein Paritätsbit;
                           '8 Datenbits;
                           '2 Stoppbits;
                           'kein Handshake

 RS_Init(module,2,9600,0,8,1,0) 'Schnittst. 2 initialisieren:
                           'wie Schnittstelle 1
 Par_1 = 1
 Par_4 = 1

Event:
 Rem Einen Datensatz lesen und schreiben
 If (Par_1 <= num_data) Then'Daten senden
  Par_2 = Write_FIFO(module,1,Data_1[Par_1])
  If (Par_2 = 0) Then Inc Par_1
 EndIf

 Par_3 = Read_FIFO(module,2)'Daten lesen
 If (Par_3 <> -1) Then
  Data_2[Par_4] = Par_3
  Inc Par_4
 EndIf
 If (Par_4 > num_data) Then End 'Alle Daten sind übertragen
```

RS232:
String-Befehl senden

Benötigt wird ein RS232-Modul mit Moduladresse 1 (Gruppe EXT-Module).

Viele Geräte mit RS232-Schnittstelle können mit String-Befehlen gesteuert werden. Die beiden folgenden Programme zeigen, wie man mit einem Prozess eine Zeichenfolge sendet und mit einem anderen Prozess die Zeichenfolge empfängt. Die Programme sind auf der *ADwin*-CD verfügbar.

Die Programme können auf dem gleichen Modul, jedoch mit verschiedenen Schnittstellen eingesetzt werden. Beachten Sie bitte die Hinweise im Programmkommentar.

Das Programm RS232_send_string.BAS initialisiert zuerst die Schnittstelle 1. Im Abschnitt **EVENT** sendet die Schnittstelle 1 des RS-Moduls eine Zeichenfolge. Im Abschnitt **Finish** wird das Zeichen „#“ als Ende-Markierung gesendet. Es kann durch ein beliebiges anderes Zeichen ersetzt werden.

```
' Process for RS232-communication: sending a string
' ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
' The program may run together with RS232_receive_string.BAS
' on the same module.  If so, please follow these instructions:
' - connect the interfaces with each other
' - compile and start RS232_receive_string.BAS
' - compile and start RS232_send_string.BAS

#Include ADwinPro_All.Inc

Rem import string library
#If Processor = T9 Then
  Import string.li9
#Else
  #If Processor = T10 Then
    Import string.lia
  #Else
    Import string.lib
  #EndIf
#EndIf

#Define rs_adr 1            'module address
#Define rs_no 1             'interface number
#Define s_endchar "#"       'end marker "#"
#Define s_send Data_1
#Define str_len 50          'length of send string

Dim s_send[str_len] As String 'send string
Dim s_temp[1] As String     'single char
Dim sp As Long              'send pointer

Init:
  Rem 0.25 s
  #If Processor = T9 Then
    Processdelay = 10000000
  #Else
    #If Processor = T10 Then
      Processdelay = 10000000
    #Else
      Processdelay = 75000000
    #EndIf
  #EndIf
  Rem A reset is allowed only once on a module!
  'RS_RESET(rs_adr)         'reset RS module
  RS_Init(rs_adr,rs_no,9600,0,8,0,0) 'init RS interface
  sp=1                      'initialize pointer
  s_send = „This is a TESTSTRING" 'send string

Event:
  StrMid(s_send, sp, 1, s_temp) 'read next char of string
  Par_11 = Asc(s_temp)      'get ascii code of char
  If (Par_11 = 0) Then End 'quit when all chars are sent
  Par_12 = Write_FIFO(rs_adr, rs_no, Par_11) 'send code
  Rem increase pointer, else send again
  If (Par_12 = 0) Then Inc sp
  Rem quit when all chars are sent
  If (sp > str_len) Then End

Finish:
  Do                        'send End marker "#"
    Par_11 = Asc(s_endchar) 'get ascii code
    Par_12 = Write_FIFO(rs_adr, rs_no, Par_11) 'send code
  Until (Par_12 = 0)
```

**RS232:
String-Befehl empfangen**

Benötigt wird ein RS232-Modul mit Moduladresse 1 (Gruppe EXT-Module).

Das Programm RS232_receive_string.BAS initialisiert zuerst das Modul und die Schnittstelle 2. Im Abschnitt **EVENT** wird eine Zeichenfolge über die Schnittstelle 2 empfangen, bis die Ende-Markierung empfangen wird (oder der Empfangs-String voll ist).

```
' Process for RS232-communication: Receiving a string.
' ++++++++++++++++++++++++++++++++++++++++++++++++++++
' The program may run together with RS232_send_string.BAS
' on the same module. If so, please follow these instructions:
' - connect the interfaces with each other
' - compile and start RS232_receive_string.BAS
' - compile and start RS232_send_string.BAS
#Include ADwinPro_All.Inc

#If Processor = T9 Then    ' import string library
 Import string.li9
#Else
 #If Processor = T10 Then
  Import string.lia
 #Else
  Import string.lib
 #EndIf
#EndIf

#Define rs_adr 1           'module address
#Define rs_no 2            'interface number
#Define s_receive Data_2
#Define str_len 50         'max. length of received string

Dim s_receive[str_len] As String 'received string
Dim s_temp[1] As String    'single char
Dim s_endchar[1] As String 'end marker
Dim endflag As Long
Dim rp As Long             'receive pointer

Init:
 Rem 0.25 s
 #If Processor = T9 Then
  Processdelay = 10000000
 #Else
  #If Processor = T10 Then
   Processdelay = 10000000
  #Else
   Processdelay = 75000000
  #EndIf
 #EndIf

 RS_Reset(rs_adr)          'reset RS module
 RS_Init(rs_adr,rs_no,9600,0,8,0,0) 'init RS interface
 rp = 0                    'initialize receive pointer
 s_receive = ""            'initialize receive string
 s_endchar = "#"           'end marker

Event:
 Par_21 = Read_FIFO(rs_adr, rs_no) 'receive status / char
 If (Par_21 <> -1) Then
  Chr(Par_21,s_temp)       'get char from ascii value
  Inc rp                   'increase receive pointer
  Rem End marker received or string full?
  endflag = StrComp(s_temp, s_endchar)
  If ((endflag=0) Or (rp>str_len)) Then End
  s_receive = s_receive + s_temp 'save char to string
 EndIf
```

**RS485:
Empfangen und senden**

Benötigt wird ein RS485-Modul mit Moduladresse 1 (Gruppe EXT-Module).

In diesem Beispiel wird eine RS485-Schnittstelle als passiver Teilnehmer verwendet, der alle Daten liest, die an seinem Eingang anliegen. Wenn ein bestimmter Wert (55) empfangen wird, wird die Schnittstelle aktiv und sendet dann ihrerseits fortlaufend den Wert 44.

```
Rem Dieses Programm setzt eine RS485-Schnittstelle mit der
Rem Adresse 1 voraus.

#Include ADwinPro_All.Inc

#Define rs_adr 1

Dim ret_val As Long
Dim val As Long

Init:
  RS_Reset(rs_adr)
  RS_Init(rs_adr,2,38400,0,8,0,3)
  RS485_Send(rs_adr,2,0) 'Kanal 2 empfangen

Event:
   val = Read_FIFO(rs_adr,2) 'Daten lesen

   If (val = 55) then
    RS485_Send(rs_adr,2,1) 'Kanal 2 senden
    ret_val = Write_FIFO(rs_adr,2,44) 'Daten schreiben
   EndIf
```

Befehlsübersichten

A.1 Alphabetische Befehlsübersicht

A.2 Befehlsübersicht nach Modulen

Sie finden die Befehlsübersichten der Module auf den folgenden Seiten:

AIn-16/14-C Rev. A

AIn-32/12 Rev. A

AIn-32/12 Rev. B

AIn-32/14 Rev. A

AIn-32/16 Rev. B

AIn-32/16 Rev. C

AIn-8/12 Rev. A

AIn-8/12 Rev. B

AIn-8/14 Rev. A

AIn-8/16 Rev. A

AIn-8/16 Rev. B

AIn-8/16 Rev. C

AIn-F-4/12 Rev. A

AIn-F-4/14 Rev. B

AIn-F-4/16 Rev. A

AIn-F-4/16 Rev. B

AIn-F-8/12 Rev. A

AIn-F-8/14 Rev. B

AIn-F-8/16 Rev. A

AIn-F-8/16 Rev. B

AOut-16/8-12

AOut-4/16 Rev. A

AOut-4/16 Rev. B

AOut-4/16 Rev. C

AOut-8/16 Rev. A

AOut-8/16 Rev. B

AOut-8/16 Rev. C

Profi-IRT-CU

Profi-IRT-FO

PT100-4, PT100-8

PWM-4(-I)

REL-16 Rev. A

REL-16 Rev. B

RS232-2, RS232-4

RS422-2, RS422-4

RS485-2, RS485-4

Storage Rev. A

TC-16

TC-4

TC-8

TC-8-ISO Rev. A

TRA-16 Rev. A

TRA-16 Rev. B

(LP)SH-8(-FI)

A.3 Thematische Befehlsübersicht

Die Befehle sind in die folgenden Themengruppen aufgeteilt. Innerhalb der Themengruppen sind die Befehle alphabetisch sortiert.

Analoge Ausgänge

DAC	gibt auf einem Kanal des angegebenen Moduls eine (analoge) Spannung aus, die dem angegebenen Digitalwert entspricht.
FG_Control	startet oder stoppt den Funktionsgenerator (d.h. die Werteausgabe) auf den gewählten Ausgabekanälen des angegebenen Moduls.
FG_Def	definiert für einen Ausgabekanal auf dem angegebenen Modul die Startadresse und die Größe des internen Zwischenspeichers für den Funktionsgenerator-Modus.
FG_Delay	stellt auf dem angegebenen Modul die Ausgaberate des Funktionsgenerators ein.
FG_Mode	aktiviert oder deaktiviert auf dem angegebenen Modul den Funktionsgenerator-Modus.
FG_Read_Index	gibt auf dem angegebenen Modul den Positionszeiger eines bestimmten Funktionsgenerators zurück.
FG_Status	gibt den Status aller Funktionsgeneratoren auf dem angegebenen Modul zurück.
FG_Write	überträgt eine gerade Zahl von Daten eines Felds an eine bestimmte Adresse im internen Zwischenspeicher des angegebenen Moduls.
Start_DAC	startet die Wandlung bzw. Ausgabe aller DAC des angegebenen D/A-Moduls.
WriteDAC	schreibt einen Digitalwert in das Ausgaberegister eines bestimmten DAC des angegebenen Moduls.
WriteDAC32	WriteDAC schreibt einen Digitalwert in das Ausgaberegister eines bestimmten DAC des angegebenen Moduls.

Analoge Eingänge

ADC	führt eine komplette Messung auf dem 12 Bit-, 14 Bit- oder 16 Bit-ADC des angegeben Moduls durch.
ADC16	führt eine komplette Messung auf einem 16 Bit ADC durch. Die Angaben gelten ausschließlich für das Modul Pro-AIn-8/16 Rev. A.
ADCF	führt eine komplette Messung auf einem Fast-ADC durch.
Burst_Abort	bricht eine laufende Burst-Messreihe auf dem angegebenen Modul ab und gibt die Anzahl der bereits durchgeführten Messungen oder der gespeicherten Messwerte zurück.
Burst_CRead	kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines bestimmten Kanals in ein Feld. Die Anzahl der zu kopierenden Messwerte muss angegeben werden.
Burst_CStart	startet eine kontinuierliche Burst-Messreihe (Modus „Continuous“) auf dem angegebenen Modul.
Burst_Init	legt die Kennwerte für eine Burst-Messreihe auf dem angegebenen Modul fest.
Burst_Read	kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines Kanals in ein bestimmtes Feld.
Burst_Read_Packed	kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines Kanals in ein be-

	stimmtes Feld, jedoch komprimiert und schnell.
Burst_Start	startet (unabhängig vom Prozessor) eine Burst-Messreihe auf dem angegebenen Modul.
Burst_Status	ermittelt die Anzahl der noch durchzuführenden Burst-Messungen auf dem angegebenen Modul.
ReadADC	liest das Ergebnis einer Wandlung aus dem ADC-Register des angegebenen Moduls aus.
ReadADCF	liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus.
ReadADCF_32	liest die Wandlungsergebnisse aus 2 aufeinander folgenden F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.
ReadADCF_SConv	liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung.
ReadADCF_SConv_32	liest die Wandlungsergebnisse aus 2 F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.
ReadADC_SConv	liest das Wandlungsergebnis aus dem ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung.
Read_ADCF4	liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus.
Read_ADCF4_Packed	liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus. Je 2 Messwerte werden gemeinsam in einem 32 Bit-Wert zurückgegeben.
Read_ADCF8	liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus.
Read_ADCF8_Packed	liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus. Je 2 Messwerte werden gemeinsam in einem 32 Bit-Wert zurückgegeben.
Seq_Mode	initialisiert das angegebene Modul für den Betrieb mit Ablaufsteuerung.
Seq_Read	kopiert eine bestimmte Anzahl an Messwerten (16 Bit) von dem angegebenen Modul in ein Ziel-Feld.
Seq_Read32	kopiert alle 32 Messwerte (je 16 Bit) von dem angegebenen Modul in ein Ziel-Feld.
Seq_Read_One	liest einen bestimmten Messwert (16 Bit) einer Messgruppe auf dem angegebenen Modul aus.
Seq_Read_Packed	kopiert eine gerade Anzahl an Messwerten (16 Bit) paarweise von dem angegebenen Modul in ein Ziel-Feld.
Seq_Read_Two	liest auf einmal 2 aufeinanderfolgende Messwerte (je 16 Bit) einer Messgruppe auf dem angegebenen Modul aus und gibt diese in einem 32 Bit-Wert zurück.
Seq_Select	legt fest, welche Kanäle zu der Messgruppe gehören, die die Ablaufsteuerung auf dem angegebenen Modul wandeln soll.
Seq_Set_Delay	legt die Einschwingzeit der Ablaufsteuerung auf dem angegebenen Modul fest.
Seq_Status	ermittelt, wieviele Kanäle der Messgruppe die Ablaufsteuerung auf dem angegebenen Modul bereits gewandelt und gespeichert hat.
Set_Gain	setzt für einen Kanal des angegebenen Moduls die Betriebsart fest und damit auch den Verstärkungsfaktor und den Messbereich.
Set_Mux	stellt den Multiplexer des angegebenen Moduls auf einen bestimmten Eingang und eine bestimmte Verstärkung ein.
SE_Diff	stellt die Betriebsart single ended oder differentiell für alle Analog-Eingänge auf dem angegebenen Modul ein.
SH_SetMode	stellt den Modus der Sample- und Hold-Stufen auf dem angegebenen Modul ein.
Start_Conv	startet die A/D-Wandlung auf dem angegebenen Modul.
Start_ConvF	startet die Wandlung auf einem oder mehreren F-ADC des angegebenen Moduls.
Sync_Mode	bestimmt auf dem angegebenen Modul die Art der Synchronisation mit anderen Modulen, insbesondere für Burst-Messreihen.
Wait_EOC	wartet, bis die zuletzt gestartete A/D-Wandlung abgeschlossen ist.
Wait_EOCF	wartet, bis die Wandlung auf allen angegebenen F-ADC des Moduls abgeschlossen ist.

Bustyp: CAN

Transmit_Status	gibt zurück, ob ein Message-Objekt bereit ist zum Senden.

CAN-Bus

CAN_Msg	ist ein eindimensionales Feld mit 9 Elementen, in dem die Message-Objekte gespeichert sind

	oder werden.
En_Interrupt	konfiguriert ein Message-Objekt des angegebenen Moduls so, dass es bei Eintreffen einer Nachricht einen Event-Signal (Interrupt) erzeugt.
En_Receive	gibt ein Message-Objekt für den Empfang von Nachrichten auf dem angegebenen Modul frei.
En_Transmit	gibt ein Message-Objekt für das Senden von Nachrichten auf dem angegebenen Modul frei.
Get_CAN_Reg	gibt den Inhalt eines bestimmten Registers auf einem CAN-Controller des angegebenen Moduls zurück.
Init_CAN	initialisiert einen der CAN-Controller auf dem angegebenen Modul und bringt ihn in einen definierten Anfangszustand.
Read_Msg	gibt zurück, ob eine neue Nachricht in einem Message-Objekt eines der CAN-Controller auf dem angegebenen Modul empfangen wurde.
Read_Msg_Con	prüft, ob eine vollständige neue Nachricht in einem bestimmten Message-Objekt in einer CAN-Schnittstelle empfangen wurde.
Set_CAN_Baudrate	stellt die angegebene Baudrate auf einem der Controller auf dem angegebenen Modul ein und gibt zurück, ob dies erfolgreich geschehen ist.
Set_CAN_Reg	schreibt einen Wert in ein Register des gewählten CAN-Controllers auf dem angegebenen Modul.
Transmit	sendet die Nachricht in CAN_Msg über ein bestimmtes Message-Objekt.

Digitale Ein-/Ausgänge

Digin_Long_F	gibt den Zustand der Eingänge (Bits 31…00) des angegebenen Moduls als Bitmuster zurück.
Digin_Word1	gibt den Zustand der Eingänge 0…15 des angegebenen Moduls als Bitmuster zurück.
Digin_Word2	gibt den Zustand der Eingänge 16…31 des angegebenen Moduls als Bitmuster zurück.
Digout	setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.
Digout_Bits_F	setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel High oder den Pegel Low.
Digout_F	setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.
Digout_Long_F	setzt oder löscht durch den übergebenen 32 Bit-Wert alle Ausgänge des angegebenen Moduls.
Digout_Word1	setzt gleichzeitig die digitalen Ausgänge 0…15 des angegebenen Moduls auf einen bestimmten Pegel.
Digout_Word2	setzt gleichzeitig die digitalen Ausgänge 16…31 des angegeben Moduls auf einen bestimmten Pegel.
DigProg1	programmiert die digitalen Kanäle 0…15 des angegebenen Moduls als Ein- oder Ausgang.
DigProg2	programmiert die Kanäle 16…31 des angegebenen Moduls als Ein- oder Ausgang.
Dig_Latch	überträgt auf dem angegebenen Modul Digital-Informationen von den Eingängen in die Eingangs-Latches und/oder von den Ausgangs-Latches zu den Ausgängen.
Dig_ReadLatch1	Dig_ReadLatch2 liefert die oberen 16 Bit (Bit 16…Bit 31) aus dem Zwischenregister für die digitalen Eingänge des angegebenen Moduls.
Dig_ReadLatch2	liefert die oberen 16 Bit (Bit 16…Bit 31) aus dem Zwischenregister für die digitalen Eingänge des angegebenen Moduls.
Dig_WriteLatch1	schreibt einen Wert in die unteren 16 Bit (Bit 0…15) des Zwischenregisters für die digitalen Ausgänge des angegebenen Moduls.
Dig_WriteLatch2	schreibt einen Wert in die oberen 16 Bit (Bit 16…31) des Zwischenregisters für die digitalen Ausgänge des angegebenen Moduls.
Dig_WriteLatch32	schreibt einen 32 Bit-Wert in das Langwort (Bits 31…00) des Latches auf dem angegebenen Modul.
Get_Digout_Long	gibt den Inhalt des Ausgangs-Latches (Register für digitale Ausgänge) auf dem angegebenen Modul zurück.
Get_Digout_Word1	gibt das untere Wort (Bit 0…15) des Ausgangs-Latches (Register für digitale Ausgänge) des angegebenen Moduls zurück.
Get_Digout_Word2	gibt das obere Wort (Bit 16…31) des Ausgangs-Latches (Register für digitale Ausgänge) des angegebenen Moduls zurück..
PWM_Enable	gibt alle internen Zähler frei oder stoppt sie. Die Zähler werden über die Nummer des zuge-

	hörigen PWM-Ausgangs gewählt.
PWM_Out	setzt einen bestimmten PWM-Ausgabekanal des angegebenen Moduls auf den Pegel High oder Low.
PWM_Set	setzt die Voreinstellungen eines bestimmten PWM-Ausgabekanals auf dem angegebenen Modul.

Kommunikationsschnittstelle

Feldbus

Changed_Data	überprüft, ob sich auf dem angegebenen Modul seit dem letzten Zugriff der Applikation auf das DP-RAM die Daten im Ausgangsbereich geändert haben.
Check_Access	gibt zurück, auf welche Bereiche des DP-RAM im angegebenen Modul die Applikation das Zugriffsrecht hat.
Get_Pro_Byte	gibt ein Byte aus einer bestimmten Speicheradresse des DP-RAM des Feldbus-Moduls zurück.
Get_Read_Buffer	kopiert einen definierten Datenblock aus dem Speicherbereich des DP-RAM in das angegebene Zielfeld.
Init_Slave	initialisiert den Feldbus-Slave und ist nur nach einem Einschalten (power up) möglich.
Request_Access	beantragt das Zugriffsrecht auf das DP-RAM des Slaves für die Applikation.
Request_Release_Access	Request_Release_Access beantragt, das Zugriffsrecht auf das DP-RAM des Slaves an den Feldbus zurückzugeben.
Set_Pro_Byte	schreibt einen Wert in eine Speicherstelle des DP-RAM des Feldbus-Moduls.
Set_Write_Buffer	kopiert die Daten eines Felds in einen definierten Speicherbereich des DP-RAM.

RSxxx

Check_Shift_Reg	gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO der RSxxx-Schnittstelle geschrieben wurden.
Get_RS	liest den Inhalt eines bestimmten Controller-Registers auf dem angegebenen Modul aus.
Read_FIFO	liest einen Wert aus dem Eingangs-FIFO eines bestimmten Kanals auf dem angegebenen Modul.
RS485_Send	legt die Übertragungsrichtung für einen bestimmten Kanal des angegebenen Moduls fest.
RS_Init	initialisiert einen bestimmten Kanal auf dem angegebenen Modul.
RS_Reset	führt auf dem angegebenen Modul einen Hardware-Reset durch und löscht dabei die Einstellungen für alle Kanäle.
Set_RS	schreibt einen Wert in ein bestimmtes Register des angegebenen Moduls.
Write_Fifo	Write_FIFO schreibt einen Wert in den Sende-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

Komparator

Comp_Digin_Word	gibt den aktuellen Status des Schwellenwert-Vergleichs für alle Kanäle auf dem angegebenen Modul zurück.
Comp_Digin_Word_Diff	gibt den aktuellen Status des Schwellenwert-Vergleichs auf dem angegebenen Modul zurück.
Comp_Fifo_Read	liest die letzten 2 x 1024 Messwerte eines Kanalpaars aus dem internen FIFO-Speicher des angegebenen Moduls und überträgt sie in 2 Felder.
Comp_Fifo_Select	legt das Kanalpaar fest, dessen Daten im internen FIFO-Speicher des angegebenen Moduls gespeichert werden.
Comp_Read	gibt den aktuellen, minimalen oder maximalen Messwert eines bestimmten Kanals auf dem angegebenen Modul zurück.
Comp_Reset	setzt auf dem angegebenen Modul die Messung des Minimal- und Maximalwerts für die ausgewählten Kanäle gleichzeitig zurück.
Comp_Set	setzt den unteren und den oberen Schwellenwert für einen bestimmten Kanal des angegebenen Moduls fest.

Speichermedium

Media_Rd_Blk_F	Media_RD_Blk_F kopiert eine Anzahl an Datenblöcken mit Float-Werten aus einer der zehn Dateien auf dem Datenträger des angegebenen Moduls in ein Feld.
Media_Rd_Blk_L	Media_RD_Blk_L kopiert eine Anzahl an Datenblöcken mit Long-Werten aus einer der zehn

	Dateien auf dem Datenträger des angegebenen Moduls in ein Feld.
Media_Rd_Fileinfo	Media_RD_FileInfo initialisiert die Mediumverwaltung (Glue-Logik) auf dem angegebenen Modul und gibt die Datei-Informationen (Start- und Endsektor) in einem Feld zurück.
Media_Wr_Blk_F	Media_WR_Blk_F kopiert eine Anzahl an Float-Datenblöcken aus einem Feld in eine der zehn Dateien auf dem Datenträger des angegebenen Moduls.
Media_Wr_Blk_L	Media_WR_Blk_L kopiert eine Anzahl an Long-Datenblöcken aus einem Feld in eine der zehn Dateien auf dem Datenträger des angegebenen Moduls.
RTC_Get	gibt das Datum und die Zeit von der Echtzeituhr des angegebenen Moduls zurück.
RTC_Set	setzt das Datum und die Zeit auf der Echtzeituhr des angegebenen Moduls. Ungültige Werte werden nicht akzeptiert.

System

CheckLED	gibt den Status der LED (oben auf der Frontplatte) auf dem angegebenen Modul zurück.
CPU_Digin (T9, T10)	Nur Prozessoren T9, T10. CPU_Digin gibt zurück, ob seit dem letzten Aufruf der Anweisung eine fallende Flanke am Eingang Digin 0 des Prozessormoduls aufgetreten ist.
EVENTENABLE	EventEnable sperrt oder aktiviert den externen Event-Eingang auf dem angegebenen Modul.
ResetWatchdogTimer	setzt den Watchdog-Zähler des CPU-Moduls zurück auf den Startwert. Der Zähler bleibt aktiv.
SetLED	schaltet die LED (oben auf der Frontplatte) auf dem angegebenen Modul ein oder aus.
StartWatchdog	aktiviert den Watchdog-Zähler des CPU-Moduls und setzt ihn auf den Startwert.
StopWatchdog	deaktiviert den Watchdog-Zähler des CPU-Moduls.
SyncAll	startet eine bestimmte Aktion synchron auf allen Modulen, die mit SyncEnable aktiviert wurden.
SyncEnable	aktiviert oder deaktiviert auf dem angegebenen Modul die Synchron-Option.
SyncStat	gibt die Einstellung der Synchron-Option auf dem angegebenen Modul zurück.

Temperatur-Eingänge

PT100_Dig_To_R	berechnet aus dem Digitalwert eines PT100-Fühlers den zugehörigen Widerstand in Ohm.
PT100_Dig_To_Temp	berechnet aus dem Digitalwert eines PT100-Fühlers die zugehörige Temperatur in Celsius oder Fahrenheit.
TCJ_Dig_To_Temp	berechnet aus einem Digitalwert eines Thermoelements vom Typ J die zugehörige Temperatur in Celsius oder Fahrenheit.
TCK_Dig_To_Temp	berechnet aus einem Digitalwert eines Thermoelements vom Typ K die zugehörige Temperatur in Celsius oder Fahrenheit.
TC_Read_B	gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.
TC_Read_E	gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.
TC_Read_J	gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.
TC_Read_K	gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.
TC_Read_N	gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.
TC_Read_R	gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.
TC_Read_S	gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.
TC_Read_T	gibt die Thermospannung (µV) oder die Temperatur (°C / °F) eines bestimmten Kanals auf dem Modul zurück.
TC_Select	schaltet den angegebenen Thermoelement-Kanal über Multiplexer auf den analogen Ausgang des Moduls.
TC_Set_Rate	stellt die Abtastrate für das angegebene Modul ein.

Zähler

Cnt_Clear	setzt einen oder mehrere Zähler auf Null, gemäß dem Bitmuster in pattern.
Cnt_Enable	aktiviert oder deaktiviert einen oder mehrere Zähler auf dem angegebenen Modul.
Cnt_Latch	übernimmt den aktuellen Zählerstand eines oder mehrerer Zähler auf dem angegebenen Modul in die jeweiligen Zwischenregister (= latchen).
Cnt_Read16	gibt den aktuellen Zählerstand eines 16 Bit-Zählers auf dem angegebenen Modul zurück.
Cnt_Read32	gibt den aktuellen Zählerstand eines 32 Bit-Zählers auf dem angegebenen Modul zurück.
Cnt_ReadLatch16	gibt den Wert aus dem Zwischenregister eines 16 Bit-Zählers auf dem angegebenen Modul zurück.
Cnt_ReadLatch32	gibt den Wert aus dem Zwischenregister eines 32 Bit-Zählers auf dem angegebenen Modul zurück.
Cnt_SetMode	stellt die Betriebsart aller Zähler auf dem angegebenen Modul ein, Vierfach-Flankenauswertung oder Takt- und Richtungseingang.
CO4_ClearEnable	schaltet den externen Eingang CLR eines oder mehrerer Zähler auf dem angegebenen Modul frei.
CO4_GetStatus	gibt den Status der Eingangssignale eines Zählers auf dem angegebenen Modul in einem Bitmuster zurück.
CO4_LatchEnable	schaltet den externen Eingang LATCH eines oder mehrerer Zähler auf dem angegebenen Modul frei.
CO4_Read	gibt den aktuellen Zählerstand eines Zählers des angegebenen Moduls zurück.
CO4_ReadLatch	gibt den Wert aus dem Zwischenregister (Latch) eines Zählers des angegebenen Moduls zurück.
CO4_ResetStatus	löscht das Statusregister eines oder mehrerer Zähler auf dem angegebenen Modul.
CO4_SetMode	stellt den Zählmodus eines Zählers auf dem angegebenen Modul ein.
CO4_Set_LatchMode	bestimmt den Modus der Latch-Eingänge für alle Zähler des angegebenen Moduls.
ExtLch_Enable	gibt alle Latch-Eingänge auf dem angegebenen Modul entweder frei oder sperrt sie. Die Latch-Eingänge werden über die Nummer des zugehörigen Zählers ausgewählt.
SSI_Mode	stellt auf dem angegebenen Modul den Modus aller SSI-Decoder ein, entweder „single shot“ (einzelne lesen) und „continuous“ (kontinuierlich lesen).
SSI_Read	gibt den zuletzt gespeicherten Zählerstand eines bestimmten SSI-Zählers auf dem angegebenen Modul zurück.
SSI_Set_Bits	stellt für einen bestimmten SSI-Zähler auf dem angegebenen Modul die Anzahl der zu Bits ein, die einen vollständigen Encoder-Wert bilden.
SSI_Set_Clock	stellt die Taktrate (ca. 40 kHz bis 1 MHz) auf dem angegebenen Modul ein, mit der der Encoder getaktet wird.
SSI_Start	startet auf dem angegebenen Modul das Auslesen eines oder beider SSI-Encoder (nur im Modus single shot).
SSI_Status	liefert für einen bestimmten Decoder den aktuellen Lese-Status auf dem angegebenen Modul zurück.