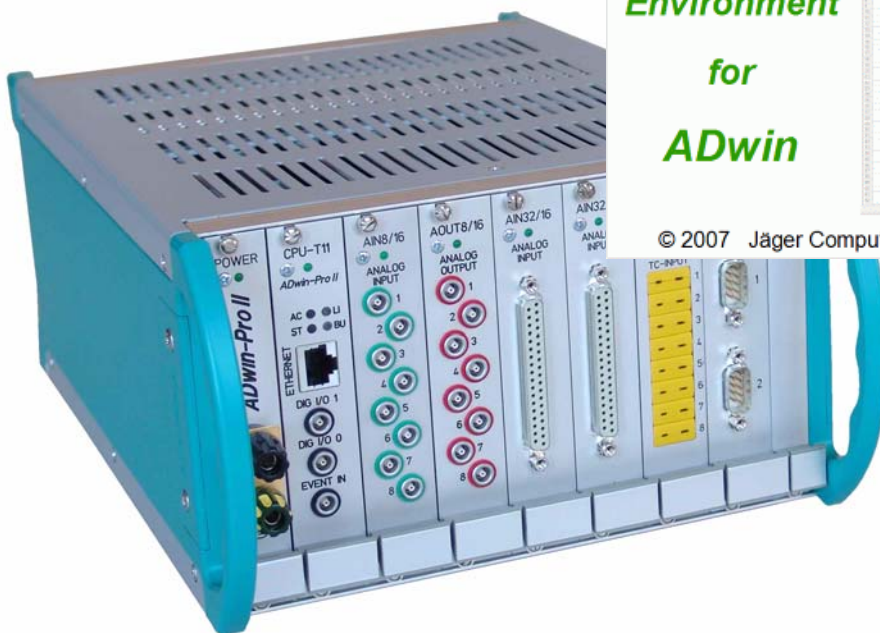
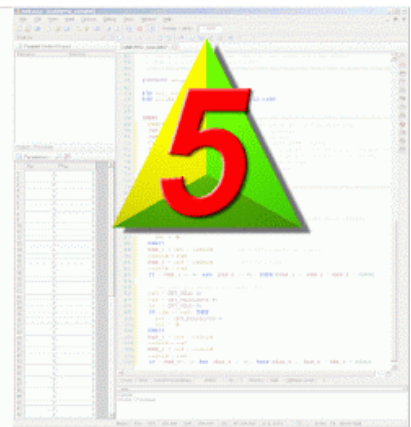


ADwin-Pro II

Systembeschreibung Programmierung in *ADbasic*



ADbasic
*Real-Time
Development
Environment*
for
ADwin



© 2007 Jäger Computergesteuerte Messtechnik GmbH

Hier finden Sie immer einen Ansprechpartner für Ihre Fragen:

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



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

Inhaltsverzeichnis

Inhaltsverzeichnis	III
Typografische Konventionen	IV
1 Einführung	1
2 Das Programm ADpro.exe	2
3 ADbasic -Befehle	3
3.1 Pro II: Allgemeine Befehle	3
3.2 Pro II: Multi-I/O	23
3.3 Pro II: Analoge Eingänge	32
3.4 Pro II: Analoge Ausgänge	115
3.5 Pro II: Digitale Ein-/Ausgänge	130
3.6 Pro II: Zähler	161
3.7 Pro II: PWM-Ausgänge	188
3.8 Pro II: Temperaturmess-Module	199
3.9 Pro II: CAN-Bus	217
3.10 Pro II: RSxxx	237
3.11 Pro II: LIN-Bus-Schnittstelle	248
3.12 Pro II: Profibus-Schnittstelle	261
3.13 Pro II: EtherCAT-Schnittstelle	266
3.14 Pro II: Flexray	273
4 Programmbeispiele	281
4.1 Kontinuierliche Messwertwandlung	281
Befehlsübersichten	A-1
A.1 Alphabetische Befehlsübersicht	A-1
A.2 Befehlsübersicht nach Modulen	A-4
A.3 Thematische Befehlsübersicht	A-13

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 angegeben.

`Programtext`

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 (wie im Editor der Entwicklungsumgebung *ADbasic*).

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 II* 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 II*-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 II*-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.

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 be-rechtigt 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).*

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.

Diese Dokumentation ist einschließlich aller Abbildungen urheberrechtlich geschützt. Reproduktion, Übersetzung sowie elektronische und fotografische Archivierung und Veränderung bedürfen der schriftlichen Genehmigung der Firma *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch.

Fremdprodukte werden ohne Vermerk auf mögliche Patentrechte genannt, deren Existenz nicht auszuschließen ist.

Hotline-Adresse siehe vordere Umschlagseite, innen.

Einschränkung der Anwendergruppe

Verfügbarkeit der Unterlagen



Rechtliche Grundlagen

Änderungen vorbehalten.

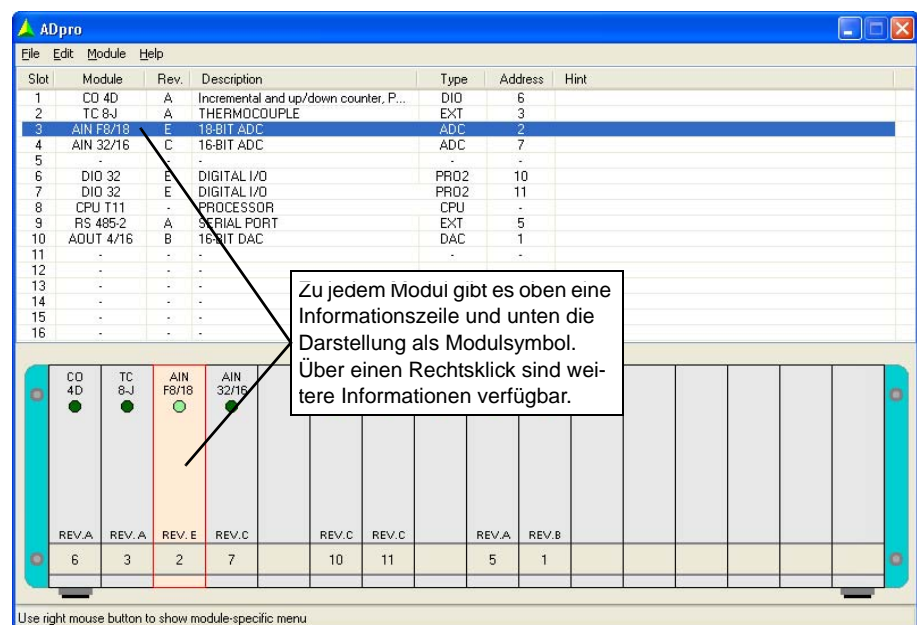
2 Das Programm ADpro.exe

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

- Bestückung eines ADwin-Pro II 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 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.

3 ADbasic-Befehle

Dieser Abschnitt beschreibt Befehle zum Ansprechen der *ADwin-Pro II*-Module.

Im Anhang finden Sie außerdem sortierte Befehlsübersichten:

- Alphabetische Befehlsübersicht (Anhang A.1)
- Befehlsübersicht nach Modulen (Anhang A.2)

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

- Thematische Befehlsübersicht (Anhang A.3)

Um einen Befehl 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.

Ein Pro II-Modul, das mit einem *ADbasic*-Befehl angesprochen wird, muss korrekt eingesteckt sein. Anderenfalls steigt die Prozessorauslastung, im Grenzfall kann sogar die Kommunikation zum PC abreißen.

Im Unterschied zu einem Pro I-Modul dauert ein Zugriffsversuch auf ein nicht ansprechbares Pro II-Modul länger als wenn das Modul ansprechbar ist. Dies kann beispielsweise geschehen, wenn Sie ein eingestecktes Pro II-Modul herausziehen. Die längere Zugriffszeit erhöht die Auslastung des CPU-Moduls und verändert das Zeitverhalten des Prozesses.

3.1 Pro II: Allgemeine Befehle

Dieser Abschnitt beschreibt Befehle, die für die meisten oder alle Pro II-Module gelten:

- [P2_Check_LED](#) (Seite 4)
- [P2_Set_LED](#) (Seite 5)
- [P2_Event_Enable](#) (Seite 6)
- [P2_Event_Config](#) (Seite 7)
- [P2_Event2_Config](#) (Seite 8)
- [P2_Event_Read](#) (Seite 10)



P2_Check_LED

P2_Check_LED gibt den Status der LED (oben auf der Frontplatte) auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_Check_LED(module)
```

Parameter

module	Moduladresse (0...15): 0: CPU-Modul. 1...15: Eingestellte Moduladresse.	LONG
ret_val	0: LED ist aus (Default). 1: LED ist an.	LONG

Siehe auch

[P2_Set_LED](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
  
Init:  
If (P2_Check_LED(1)=0) Then 'Falls LED aus ist ...  
    P2_Set_LED(1,1)          '... dann LED einschalten  
EndIf
```


P2_Set_LED schaltet die LED (oben auf der Frontplatte) auf dem angegebenen Modul ein oder aus.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_Set_LED(module, enable)
```

Parameter

module	Moduladresse (0...15): 0: CPU-Modul. 1...15: Eingestellte Moduladresse.	LONG
enable	Gewünschter Schaltzustand der LED. 0: ausschalten. 1: einschalten.	LONG

Bemerkungen

Manche Module besitzen zusätzliche LED. Der Status solcher LED wird mit separaten Befehlen eingestellt.

Siehe auch

[P2_Check_LED](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
  
Init:  
    P2_Set_LED(1,1)           'LED am Modul 1 einschalten  
  
Event:  
    Rem ...  
  
Finish:  
    P2_Set_LED(1,0)           'LED am Modul 1 ausschalten
```

P2_Set_LED

P2_Event_Enable

P2_Event_Enable 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  
  
P2_Event_Enable(module, enable)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
enable	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 (vgl. *ADbasic*-Handbuch).

Die meisten Module verfügen über einen Event-Eingang. Konfigurieren Sie den Event-Eingang zuerst mit **P2_Event_Config**. Sobald Sie mit **P2_Event_Enable** den Eingang aktiviert haben, wird ein anliegendes Signal an das Prozessormodul weitergeleitet. Das Prozessormodul erkennt den eingestellten Flankentyp (positiv oder negativ) als Event-Signal und der eingestellte Prozess reagiert.

Beachten Sie bei Modulen mit mehreren Event-Eingängen die Einstellung mit **P2_Event2_Config**. Die Konfiguration von **P2_Event_Config** bezieht sich dann auf das resultierende Event-Signal.

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.

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

[P2_Event_Config](#), [P2_Event2_Config](#), [P2_Event_Read](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
Init:  
  Rem Event-Eingang am Modul 1 konfigurieren für  
  Rem Mindestzeit 15 ns, neg. Flanken, 4 Flanken  
  P2_Event_Config(1,0,2,4)  
  Rem Externes Event-Signal am Modul 1 freigeben  
  P2_Event_Enable(1,1)
```

Ein Event-Eingang

Mehrere Event-Eingänge



P2_Event_Config konfiguriert den externen Event-Eingang des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc

P2_Event_Config(module,min_hold,edge,prescale)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
min_hold	Mindestzeit, die eine Flanke anliegen muss, damit sie akzeptiert wird: 0: 15ns (Default). 1: 50ns.	LONG
edge	Flankentyp, der akzeptiert wird: 1: positive Flanke (Default). 2: negative Flanke. 3: positive und negative Flanke.	LONG
prescale	Anzahl (1...255) an Flanken, nach der ein Event-Signal erzeugt wird (Default: 1).	LONG

Bemerkungen

Ein Event-Eingang muss mit der Anweisung **P2_Event_Enable** aktiviert werden, damit ein anliegendes Signal verarbeitet werden kann. Konfigurieren Sie den Event-Eingang zuerst mit **P2_Event_Config** und aktivieren den Eingang danach.

Beachten Sie bei Modulen mit mehreren Event-Eingängen die Einstellung mit **P2_Event2_Config**. Die Konfiguration von **P2_Event_Config** bezieht sich dann auf das resultierende Event-Signal. Die Einstellung **min_hold** gilt für alle Event-Eingänge



Siehe auch

[P2_Event_Enable](#), [P2_Event2_Config](#), [P2_Event_Read](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
Rem Event-Eingang am Modul 1 konfigurieren für
Rem Mindestzeit 15 ns, neg. Flanken, 4 Flanken
P2_Event_Config(1,0,2,4)
Rem Externes Event-Signal am Modul 1 freigeben
P2_Event_Enable(1,1)
```

P2_Event2_Config

P2_Event2_Config konfiguriert die Vorverarbeitung der Event-Signale auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc

P2_Event2_Config(module, mode, edge)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
mode	Modus der Event-Vorverarbeitung: 0: Keine Vorverarbeitung (Default). 1: Signal nach Freigabeimpuls am Eingang ENABLE . 2: Signal aus AB-Modus. 3: Signal aus AB-Modus nach Freigabeimpuls am Eingang ENABLE .	LONG
edge	Nur für mode =1 oder mode =3; Flankentyp, der am Eingang ENABLE akzeptiert wird: 1: positive Flanke (Default). 2: negative Flanke. 3: positive und negative Flanke.	LONG

Bemerkungen

Der Befehl hat nur eine Bedeutung für Module mit mehr als einem Event-Eingang. Das Modul verarbeitet die Signale der Event-Eingänge zum resultierenden Event-Signal, das modulinterne Vorgänge startet und den Event-Prozess steuert.

Das resultierende Event-Signal wird mit **P2_Event_Config** konfiguriert. **P2_Event_Enable** gibt das resultierende Event-Signal für die Steuerung des Event-Prozesses frei.

Je nach Modul stehen verschiedene Modi zur Verfügung:

Modul	Verfügbare Modi
Pro-Aln-F-8/14-D	0, 1, 2, 3
Pro-Aln-F-8/18-D	0, 1

Das Modul verwendet das Signal am Eingang **EVENT / A** als resultierendes Event-Signal. Der Parameter **edge** hat keine Bedeutung.

Sobald ein Impuls vom Typ **edge** am Eingang **ENABLE** anliegt, gibt das Modul den Eingang **EVENT / A** frei und verwendet dessen Signal als resultierendes Event-Signal.

Ein Aufruf von **P2_Event2_Config** mit **mode**=0 sperrt den Eingang **EVENT / A** wieder.

Im AB-Modus wertet das Modul zwei Rechteck-Signale an den Eingängen **EVENT / A** und **B** aus, die um 90 Grad gegeneinander versetzt sind (typisch für Inkrementalgeber): Wenn eine Flanke vom Typ **edge** an einem der Eingänge eintrifft, wird ein resultierendes Event-Signal ausgelöst.

Die maximal verwertbare Eingangsfrequenz beträgt 5MHz; gemeinsam mit den 4 Flanken je Signalzyklus ergibt sich eine maximale Frequenz von 20MHz für das resultierende Event-Signal.

Der Abstand zwischen einer Flanke an **EVENT / A** und einer Flanke an **B** darf 50ns nicht unterschreiten. Impulsbreiten oder Pausenzeiten kürzer als 100ns werden nicht verwertet.

Ohne Vorverarbeitung

Signal nach
Freigabeimpuls

Signal aus AB-Modus

Eine Änderung der Phasenverschiebung hat Einfluss auf die maximale Eingangsfrequenz wegen der Mindestabstände der Flanken. Wenn die Eingangssignale nicht um 90 Grad gegeneinander versetzt sind, sinkt die maximale Eingangsfrequenz von 5MHz beispielsweise bei 45 Grad auf 2,5MHz.

Sobald ein Impuls vom Typ **edge** am Eingang **ENABLE** anliegt, gibt das Modul das resultierende Event-Signal aus den Eingängen **EVENT / A** und **B** (siehe [Signal aus AB-Modus](#)) frei und verarbeitet die Rechtecksignale zum .

Mit Modus 0 werden die Eingänge **EVENT / A** und **B** gesperrt.

Siehe auch

[P2_Event_Enable](#), [P2_Event_Config](#), [P2_Event_Read](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
Rem Event-Eingang am Modul 1 konfigurieren für  
Rem Mindestzeit 15 ns, neg. Flanken, 4 Flanken  
P2_Event_Config(1,0,2,4)  
Rem Vorverarbeitung auf Freigabeimpuls und  
Rem neg. Flanke einstellen  
P2_Event2_Config(1,1,2)  
Rem Externes Event-Signal am Modul 1 freigeben  
P2_Event_Enable(1,1)
```

**Signal aus AB-Modus
nach Freigabeimpuls**

P2_Event_Read

P2_Event_Read gibt den aktuellen TTL-Pegel an den Event-Eingängen des angegebenen Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_Event_Read(module)
```

Parameter

module	Moduladresse (0...15): 1...15: Eingestellte Moduladresse.	LONG
ret_val	Bitmuster, das die anliegenden TTL-Pegel darstellt; Zuordnung der Bits zu den Event-Eingängen siehe Tabelle. Bit = 0: TTL-Pegel low. Bit = 1: TTL-Pegel high.	LONG

Bits in ret_val	31:03	02	01	00
Eingang	–	B	Enable	Event / A

Bemerkungen

Die Eingänge A, B und Enable sind nicht auf jedem Modul vorhanden.

Siehe auch

[P2_Event_Enable](#), [P2_Event_Config](#), [P2_Event2_Config](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc

Init:
    Rem Event-Eingang am Modul 1 (Aln-F-8/14) konfigurieren
    Rem für Mindestzeit 15 ns, neg. Flanken, 4 Flanken
    P2_Event_Config(1,0,2,4)
    Rem Externes Event-Signal am Modul 1 freigeben
    P2_Event_Enable(1,1)
- / -
Event:
    Par_1 = P2_Event_Read(1)
```

Nur Prozessor T11. **CPU_Digin** gibt zurück, ob seit dem letzten Befehlsaufruf eine Flanke an einem DIG I/O-Eingang des Prozessormoduls aufgetreten ist.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = CPU_Digin(channel)
```

Parameter

channel	Numer des DIG I/O-Eingangs am Prozessormodul: 0: DIG I/O 0. 1: DIG I/O 1.	LONG
ret_val	Statusmeldung, ob eine Flanke an dem gewählten DIG I/O-Eingang aufgetreten ist: 0: Flanke ist nicht aufgetreten. 1: Flanke ist ein- oder mehrfach aufgetreten.	LONG

Bemerkungen

Die Anweisung **CPU_Digin** hat nur eine Funktion, wenn der gewählte DIG I/O-Kanal mit **CPU_Dig_IO_Config** als Eingang konfiguriert ist.

Mit **CPU_Dig_IO_Config** wird festgelegt, ob **CPU_Digin** auf steigende oder auf fallende Flanken reagiert. Nach einem Neustart sind die DIG I/O-Kanäle als Eingang und für fallende Flanken konfiguriert.

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

An den DIG I/O-Eingängen werden TTL-Signale erwartet.

Siehe auch

[CPU_Digout](#), [CPU_Dig_IO_Config](#), [CPU_Digin](#) (T9, T10)

Gültig für

CPU-T11

Beispiel

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

Init:

```
Rem Beide DIG I/O Kanäle als Eingang mit steigender Flanke
Rem einstellen
CPU_Dig_IO_Config(100010b)
Rem Statusmeldung an DIG I/O 1 lesen und dadurch zurücksetzen
dummy = CPU_Digin(1)
```

Event:

```
Rem ...
If(CPU_Digin(1) = 1) Then 'Bei steigender Flanke ...
    End '... das Programm beenden
EndIf
Rem ...
```

CPU_Digin

CPU_Digout

CPU_Digout setzt einen DIG I/O-Ausgang des Prozessormoduls auf den angegebenen TTL-Pegel.

Syntax

```
#Include ADwinPro_All.Inc  
CPU_Digout(channel, level)
```

Parameter

channel	Nummer (0, 1) des DIG I/O-Ausgangs am Prozessormodul.	LONG
level	TTL-Pegel des Ausgangs: 0: TTL-Pegel low. 1: TTL-Pegel high.	LONG

Bemerkungen

Die Anweisung **CPU_Digout** hat nur eine Funktion, wenn der gewählte DIG I/O-Kanal mit **CPU_Dig_IO_Config** als Ausgang konfiguriert ist.

Siehe auch

[CPU_Digin](#), [CPU_Dig_IO_Config](#)

Gültig für

CPU-T11

Beispiel

```
#Include ADwinPro_All.Inc
```

Event:

```
Rem ...  
CPU_Digout(1,0)           'DIG I/O 1 auf TTL-Pegel low setzen  
Rem ...
```


CPU_Dig_IO_Config konfiguriert alle DIG I/O-Kanäle des Prozessormoduls.

Syntax

```
#Include ADwinPro_All.Inc
CPU_Dig_IO_Config(pattern)
```

Parameter

pattern Bitmuster zur Einstellung von Kanal- und Flankentyp am Eingang DIG I/O n: LONG

Bit = 0: Kanal als Eingang oder Flankentyp fallend.
Bit = 1: Kanal als Ausgang oder Flankentyp steigend.

Bits in pattern		31:06	05	04	03:02	01	00
DIG I/O-0	Kanaltyp	–	–	–	–	–	x
	Flankentyp	–	–	–	–	x	–
DIG I/O-1	Kanaltyp	–	–	x	–	–	–
	Flankentyp	–	x	–	–	–	–

Bemerkungen

Der Flankentyp kann nur für Eingänge eingestellt werden.

Nach einem Neustart sind die DIG I/O-Kanäle als Eingang und für fallende Flanken konfiguriert.

Siehe auch

[CPU_Digin](#), [CPU_Digout](#), [CPU_Event_Config](#)

Gültig für

CPU-T11

Beispiel

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

Init:

```
Rem Beide DIG I/O Kanäle als Eingang mit steigender Flanke
Rem einstellen
CPU_Dig_IO_Config(100010b)
Rem Statusmeldung an DIG I/O 1 lesen und dadurch zurücksetzen
dummy = CPU_Digin(1)
Rem ...
```

CPU_Dig_IO_Config

CPU_Event_Config

CPU_Event_Config konfiguriert den EVENT IN-Kanal des Prozessormoduls.

Syntax

```
#Include ADwinPro_All.Inc  
  
CPU_Event_Config(min_hold, edge, prescale)
```

Parameter

min_hold	Mindestzeit, die eine Flanke anliegen muss, damit sie akzeptiert wird: 0: 15ns (Default). 1: 50ns.	LONG
edge	Flankentyp, der akzeptiert wird: 1: positive Flanke (Default). 2: negative Flanke. 3: positive und negative Flanke.	LONG
prescale	Anzahl (1...15) an Flanken, nach der ein Event-Signal erzeugt wird (Default:1).	LONG

Bemerkungen

Am Eingang **EVENT IN** werden TTL-Signale erwartet.

Wenn die Eingangssignale zu häufig Störimpulse enthalten – soweit die Störimpulse nicht vermeidbar sind –, kann man Folgendes tun:

- Parameter **min_hold** auf 1 setzen, um kurze Störimpulse auszufiltern.
- Das Eingangssignal vorher über einen Optokoppler leiten.
- Das Eingangssignal am Modul Pro-OPT-16 anlegen und mit **EventEnable** den externen Event-Eingang des Moduls aktivieren.

Siehe auch

[P2_Event_Config](#), [P2_Event_Enable](#), [EventEnable](#)

Gültig für

CPU-T11

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
Rem Eingang EVENT IN konfigurieren für  
Rem Mindestzeit 15 ns, neg. Flanken, 4 Flanken  
CPU_Event_Config(0,2,4)
```

Event:

```
Rem Event-gesteuerter Prozess startet jeweils, wenn 4 negative  
Rem Flanken am Eingang EVENT IN angelegen haben.  
Rem ...
```

P2_Sync_All startet auf den angegebenen Modulen synchron eine bestimmte Aktion.

Syntax

```
#Include ADwinPro_All.Inc

P2_Sync_All(module_pattern)
```

Parameter

module_ Bitmuster zum Ansprechen der Moduladressen, LONG
pattern die synchron starten sollen:
 Bit = 0: Modul ignorieren.
 Bit = 1: Modul synchron starten.

Bits in pattern	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen

Die Aktion, die auf einem der gewählten Module startet, ist abhängig vom jeweiligen Modultyp. Für die Aktion gelten die zuvor festgelegten Einstellungen, z. B. für Multiplexer, Ausgabewert oder Burst-Modus.

Modultyp	Aktion
Analog-Eingang	A/D-Wandlung auf allen freigegebenen ADC starten, siehe P2_Start_Conv / P2_Start_ConvF . Die Wandlung kann eine Einzelmessung oder Teil einer Burst-Messreihe sein.
Analog-Ausgang	D/A-Wandlung auf allen freigegebenen DAC starten mit dem Wert aus dem DAC-Register, siehe P2_Start_DAC
Digital-Eingang	Aktuellen Zustand der Eingänge in das Eingangs-Zwischenregister übertragen (von dort auslesen mit P2_Dig_Read_Latch).
Digital-Ausgang	Wert aus dem Ausgangs-Zwischenregister lesen und auf die digitalen Ausgänge schalten (siehe P2_Dig_Write_Latch).
Zähler	Aktuelle Zählerstände in die Zähler-Zwischenregister übertragen; von dort auslesen z. B. mit P2_Cnt_Read_Latch oder P2_Cnt_Get_PW .
Temperatur	Aktuelle Werte an den Kanälen in Zwischenregister übertragen.

Als Voreinstellung nehmen alle Ein- oder Ausgänge der gewählten Module an der Aktion teil. Mit **P2_Sync_Enable** können Sie ein oder mehrere Ein- oder Ausgänge eines Moduls für die Synchronaktion sperren oder freigeben.

Mit den folgenden Befehlen können ebenfalls Module synchron angesprochen werden:

- **P2_Sync_Mode**: Ein externes Event-Signal löst eine Wandlung auf mehreren Modulen und auf allen Kanälen aus. Die Wandlung kann eine Einzelmessung sein (**P2_ADCF_Mode**) oder Teil einer Burst-Messreihe (**P2_Burst_Init**).
- **P2_Burst_Start**: Per Software werden Burst-Messreihen auf mehreren Modulen gestartet.

P2_Sync_All

Siehe auch

[P2_Sync_Enable](#), [P2_Sync_Mode](#), [P2_Sync_Stat](#)

[P2_Start_Conv](#), [P2_Start_DAC](#)

[P2_Write_DAC](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E, TC-8-ISO Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_5[1000] As Long

Init:
    Rem Kanäle auf den Modulen 1, 2, 4 und 5 aktivieren
    P2_Sync_Enable(1,11b)
    P2_Sync_Enable(2,11b)
    P2_Sync_Enable(4,11b)
    P2_Sync_Enable(5,100b)
    P2_Write_DAC(5,1,0)      'Ausgabe initialisieren
    i=1                      'Index initialisieren

Event:
    Rem Wandlung für Module 1,2,4,5 synchron starten
    P2_Sync_All(11011b)
    P2_Wait_EOC(1)           'Auf des Ende der Wandlung warten
    Rem A/D Wandler 1 der Module 1,2,4 auslesen
    Data_1[i]=P2_Read_ADCF(1,1)
    Data_2[i]=P2_Read_ADCF(2,1)
    Data_3[i]=P2_Read_ADCF(4,1)

    Rem Wert in Ausgangsregister des D/A Moduls 5 schreiben
    P2_Write_DAC(5,1,Data_5[i])
    If (i=1000) Then End     'Ende nach 1000 Durchläufen
    Inc(i)                   'Index erhöhen
```

P2_Sync_Enable aktiviert oder deaktiviert die gewählten Eingänge, Ausgänge oder Funktionsgruppen auf dem angegebenen Modul für die Synchron-Option.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_Sync_Enable(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Bitmuster zur Festlegung der Eingänge, Ausgänge oder Funktionsgruppen, die aktiviert oder deaktiviert werden: Bit = 0: deaktivieren. Bit = 1: aktivieren	LONG

Bits in channel	31:08	07	06	05	04	03	02	01	00
Kanalnr. in Analog-Eingangsmodulen Aln-F-x/x Rev. E	–	8	7	6	5	4	3	2	1
Kanalnr. in Analog-Ausgangsmodulen AOut-x/16 Rev. E	–	8	7	6	5	4	3	2	1
Bits in channel	31:04	03	02		01		00		
Funktionsgruppe in Multi-I/O-Modulen	–	Zähler	analoge Eingänge		analoge Ausgänge		digitale Kanäle		

Bemerkungen

Nach dem Einschalten des Geräts ist für alle Module der Wert **0FFFFh** voreingestellt, d.h. alle Eingänge, Ausgänge oder Funktionsgruppen sind aktiviert.

Der Befehl beeinflusst immer alle Kanäle oder Funktionsgruppen des Moduls. Um den Zustand eines einzigen Kanals zu ändern, müssen Sie daher den Zustand der übrigen Kanäle unverändert mit angeben.

Das Synchron-Signal wird mit **P2_Sync_All** ausgelöst.

Siehe auch

[P2_Sync_All](#), [P2_Sync_Mode](#), [P2_Sync_Stat](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

P2_Sync_Enable

Beispiel

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long
Dim Data_5[1000] As Long
- / -
Init:
    Rem Kanäle auf den Modulen 1, 2, 4 und 5 aktivieren
    P2_Sync_Enable(1,11b)
    P2_Sync_Enable(2,11b)
    P2_Sync_Enable(4,11b)
    P2_Sync_Enable(5,100b)
    P2_Write_DAC(5,1,0)      'Ausgabe initialisieren
    i=1                      'Index initialisieren

Event:
    Rem Wandlung für Module 1,2,4,5 synchron starten
    P2_Sync_All(11011b)
    P2_Wait_EOC(1)           'Auf des Ende der Wandlung warten
    Rem A/D Wandler 1 der Module 1,2,4 auslesen
    Data_1[i]=P2_Read_ADCF(1,1)
    Data_2[i]=P2_Read_ADCF(2,1)
    Data_3[i]=P2_Read_ADCF(4,1)
    Rem Wert in Ausgangsregister des D/A Moduls 5 schreiben
    P2_Write_DAC(5,1,Data_5[i])
    If (i=1000) Then End     'Ende nach 1000 Durchläufen
    Inc(i)                   'Index erhöhen
```

P2_Sync_Mode aktiviert oder deaktiviert die Synchronisation (von Messwert-Wandlungen) mit anderen Modulen als Master oder Slave.

Syntax

```
#INCLUDE ADwinPro_All.Inc
```

```
P2_Sync_Mode ( module , mode )
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
mode	Synchronisationsmodus (0...2): 0: keine Synchronisation (voreingestellt). 1: Synchronisation als Master-Modul. 2: Synchronisation als Slave-Modul.	LONG

Bemerkungen

Die Synchronisation erlaubt, dass ein Signal am Event-Eingang des Master-Moduls gleichzeitig auch Wandlungen auf allen Slave-Modulen startet. Hierzu leitet das Master-Modul das Event-Signal an die Slave-Module weiter.

Die Einstellungen für die Verarbeitung des Event-Signals (**P2_Event_Config**, **P2_Event2_Config**) können für jedes Modul unterschiedlich sein.

Sie dürfen nur ein einziges Master-Modul einstellen.

Sobald als Slave synchronisierte Module (Modus 2) das weitergeleitete Event-Signal empfangen, starten sie gleichzeitig eine Wandlung auf allen Kanälen (wie mit **P2_Start_ConvF**). Die Wandlung kann Teil einer Einzelmessung oder einer Burst-Messreihe sein.

Wenn Sie Burst-Messreihen auf mehreren Modulen synchronisieren, sollten Sie mit **P2_Burst_Init** auf jedem Modul die gleiche Zahl an Messungen einstellen. Insbesondere muss die Zahl der Messungen auf dem Master-Modul 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

[P2_Event_Enable](#), [P2_Event_Config](#), [P2_Event2_Config](#), [P2_Sync_All](#), [P2_Sync_Enable](#), [P2_Sync_Stat](#)

Gültig für Module

Aln-F-4/14 Rev. E, Aln-F-8/14 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/18 Rev. E

P2_Sync_Mode

Beispiel

```
#Include ADwinPro_All.Inc
#Define count 10000
#Define module 1

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:
P2_Sync_Mode(module,1) 'master module
P2_Sync_Mode(module+1,2) 'slave module 1
P2_Sync_Mode(module+2,2) 'slave module 2
Rem initialize and start burst measurement for 4 channels
P2_Burst_Init(module,15,0,count,1,100b)
P2_Burst_Init(module+1,15,0,count,1,100b)
P2_Burst_Init(module+2,15,0,count,1,100b)
P2_Burst_Start(111b) 'start burst measurement on module 1-3
Processdelay=800 'get trigger point with 50 kHz

Event:
Par_1=P2_Burst_Status(module)'number of remaining measurements
If (Par_1=0) Then End 'burst sequence finished, go to FINISH

Finish:
Rem copy the last converted data of all 4 channels
P2_Burst_Read_Unpacked4(module,count,0,
    Data_1,Data_2,Data_3,Data_4,1,3)
P2_Burst_Read_Unpacked4(module+1,count,0,
    Data_5,Data_6,Data_7,Data_8,1,3)
P2_Burst_Read_Unpacked4(module+2,count,0,
    Data_10,Data_10,Data_11,Data_12,1,3)
```


P2_Sync_Stat gibt die Einstellung der Synchron-Option auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_Sync_Stat(module)
```

Parameter

module Eingestellte Moduladresse (1...15). LONG

ret_val Einstellung der Synchron-Option für die LONG
Eingänge, Ausgänge oder Funktionsgruppen:
Bit = 0: deaktiviert.
Bit = 1: aktiviert.

Bits in channel	31:08	07	06	05	04	03	02	01	00
Kanalnr. in Analog-Eingangsmodulen	–	8	7	6	5	4	3	2	1
Pro-Aln-F-x/x Rev. E									
Kanalnr. in Analog-Ausgangsmodulen	–	8	7	6	5	4	3	2	1
Pro-AOut-x/16 Rev. E									
Bits in channel	31:05	04	03	02	01	00			
Funktionsgruppe in Multi-I/O-Modulen	–	Zähler	digitale Ausgänge	digitale Eingänge	analoge Ausgänge	analoge Eingänge			

Bemerkungen

Sie stellen die Synchronoption der Kanäle für Funktionsgruppen mit **P2_Sync_Enable** ein.

Siehe auch

[P2_Sync_All](#), [P2_Sync_Enable](#), [P2_Sync_Mode](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

P2_Sync_Stat

Beispiel

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

Init:
    Rem Ist Kanal 1 auf Modul 1 noch deaktiviert?
    If (P2_Sync_Stat(1) And 1 = 0) Then
        Rem Kanal 1 auf den D/A-Modulen 1+2 aktivieren,
        Rem alle anderen Kanäle deaktivieren
        P2_Sync_Enable(1,1)
        P2_Sync_Enable(2,1)
    EndIf
    i=1                                'Index initialisieren

Event:
    Rem Werte in Ausgangsregister schreiben
    P2_Write_DAC(1,1,Data_1[i])
    P2_Write_DAC(2,1,Data_2[i])
    Rem Ausgabe auf Modulen 1+2 synchron starten
    P2_Sync_All(11b)
    If (i=1000) Then End                'Ende nach 1000 Durchläufen
    Inc(i)                              'Index erhöhen
```

3.2 Pro II: Multi-I/O

Dieser Abschnitt beschreibt Befehle, die für Pro II Multi-I/O-Module gelten:

- [P2_MIO_Dig_Latch](#) (Seite 24)
- [P2_MIO_Dig_Read_Latch](#) (Seite 25)
- [P2_MIO_Dig_Write_Latch](#) (Seite 26)
- [P2_MIO_Digin_Long](#) (Seite 27)
- [P2_MIO_Digout](#) (Seite 28)
- [P2_MIO_Digout_Long](#) (Seite 29)
- [P2_MIO_DigProg](#) (Seite 30)
- [P2_MIO_Get_Digout_Long](#) (Seite 31)

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 MIO-Modul die Adresse 1 eingestellt ist.



P2_MIO_Dig_Latch

P2_MIO_Dig_Latch überträgt auf dem angegebenen Modul Digital-Informationen von den Eingängen in die Eingangs-Latches und von den Ausgangs-Latches zu den Ausgängen.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_MIO_Dig_Latch(module)
```

Parameter

module Eingestellte Moduladresse (1...15).

LONG

Bemerkungen

; davon ausgenommen sind Bei digitalen Eingängen überträgt die Anweisung die Eingangs-Signale an die Eingangs-Latches. Bei digitalen Ausgängen überträgt die Anweisung die Werte der Ausgangs-Latches an die Ausgänge.

Wenn das angegebene Modul durch **P2_Sync_Enable** zur Synchronisation freigeschaltet ist, hat der Befehl **P2_Sync_All** die gleiche Funktion wie **P2_MIO_Dig_Latch**.

Siehe auch

[P2_MIO_Dig_Read_Latch](#), [P2_MIO_Dig_Write_Latch](#), [P2_MIO_Dig_Prog](#), [P2_MIO_Digin_Long](#), [P2_MIO_Digout](#), [P2_MIO_Digout_Long](#), [P2_MIO_Get_Digout_Long](#), [P2_Sync_All](#)

Gültig für

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Kanäle 0...3 als Ausgang setzen, 4...7 als Eingang  
P2_MIO_Digprog(1,0011b)  
P2_MIO_Dig_Write_Latch(1,0)'Alle Ausgangs-Bits auf 0 setzen
```

Event:

```
Rem Eingänge latchen, Inhalt des Ausgangs-Latches ausgeben  
P2_MIO_Dig_Latch(1)  
Par_1 = P2_MIO_Dig_Read_Latch(1)'Eingangsbits einlesen und ...  
P2_MIO_Dig_Write_Latch(1,Par_1)'beim nächsten Event ausgeben
```

P2_MIO_Dig_Read_Latch liefert die Bitwerte aus dem Latch-Register für digitale Eingänge auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_MIO_Dig_Read_Latch(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Bitwerte des Latch-Registers. Jedes Bit entspricht einem digitalen Eingang (siehe Tabelle).	LONG

Bitnr.	31...20	19 ... 16	15...8	7 6 ... 1 0
Eingang	–	OPT4 ... OPT1	–	7 6 ... 1 0

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **P2_MIO_DigProg** als Eingänge zu programmieren; davon ausgenommen sind OPT-Kanäle.

Sie können den aktuellen Zustand der digitalen Eingänge mit folgenden Anweisungen in das Zwischenregister übernehmen:

- **P2_MIO_Dig_Latch**
- **P2_Sync_All** (falls für das Modul aktiviert).

Siehe auch

[P2_MIO_Dig_Latch](#), [P2_MIO_Dig_Write_Latch](#), [P2_MIO_DigProg](#), [P2_MIO_Digin_Long](#), [P2_Sync_All](#)

Gültig für

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

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

Init:

```
Rem Kanäle 07:00 der Module 1+2 als Eingänge setzen
P2_MIO_Digprog(1,00b)
P2_MIO_Digprog(2,00b)
```

Event:

```
Rem Pegel an den digitalen Eingängen von beiden Modulen synchron
Rem in die Zwischenregister übernehmen
P2_Sync_All(11b)
Par_1 = P2_MIO_Dig_Read_Latch(1)'Latch von Modul 1 lesen
Par_2 = P2_MIO_Dig_Read_Latch(2)'Latch von Modul 2 lesen
```

P2_MIO_Dig_Read_Latch

P2_MIO_Dig_Write_Latch

P2_MIO_Dig_Write_Latch schreibt einen 32 Bit-Wert in das Latch-Register für digitale Ausgänge auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

P2_MIO_Dig_Write_Latch(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster. Jedes Bit entspricht einem digitalen Ausgang (siehe Tabelle).	LONG

Bitnr.	31...28	27 ... 24	23...8	7 6 ... 1 0
Ausgang	–	TRA4 ... TRA1	–	7 6 ... 1 0

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **P2_MIO_DigProg** als Ausgänge programmiert werden; davon ausgenommen sind .

Sie können den Wert des Zwischenregisters für die digitalen Ausgänge auch mit der Anweisung **P2_MIO_Digout** setzen.

Siehe auch

[P2_MIO_Dig_Latch](#), [P2_MIO_Dig_Read_Latch](#), [P2_MIO_DigProg](#), [P2_MIO_Get_Digout_Long](#)

Gültig für

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    P2_MIO_Digprog(1,11b)    'Kanäle 07:00 des Moduls als Ausgang

Event:
    Rem Informationen des Ausgangs-Latches ausgeben
    P2_MIO_Dig_Latch(1)
    Rem Long-Word ins Ausgangs-Latch schreiben
    P2_MIO_Dig_Write_Latch(1,Par_1)
```

P2_MIO_Digin_Long gibt den Zustand der Eingänge (Bits 7...0) des angegebenen Moduls als Bitmuster zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_MIO_Digin_Long(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Bitmuster. Jedes Bit entspricht dem Eingangszustand eines digitalen Eingangs (siehe Tabelle). Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an.	LONG

Bitnr.	31...20	19 ... 16	15...8	7 6 ... 1 0
Eingang	–	OPT4 ... OPT1	–	7 6 ... 1 0

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **P2_MIO_DigProg** als Eingänge zu programmieren; davon ausgenommen sind .

Siehe auch

[P2_MIO_Dig_Latch](#), [P2_MIO_DigProg](#), [P2_MIO_Digout_Long](#)

Gültig für

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    P2_MIO_Digprog(1,00b)    'Kanäle 07:00 als Eingang

Event:
    Par_1 = P2_MIO_Digin_Long(1)'Alle Eingänge einlesen
```

P2_MIO_Digin_Long

P2_MIO_Digout

P2_MIO_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  
  
P2_MIO_Digout(module,output,value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
output	Nummer des Ausgangs, der angesprochen werden soll. 0...7: DIO-Kanäle. 24...27: TRA-Ausgänge.	LONG
value	Neuer Zustand für den gewählten Ausgang: 0: Pegel Low. 1: Pegel High.	LONG

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **P2_MIO_DigProg** als Ausgänge programmiert werden; davon ausgenommen sind

Mit **P2_MIO_Digout** kann ein beliebiger Ausgang gelöscht oder gesetzt werden, ohne den Zustand der anderen Ausgänge zu ändern.

Siehe auch

[P2_MIO_Digout_Long](#) [P2_MIO_DigProg](#)

Gültig für

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Kanäle 0...3 als Eingang, 4...7 als Ausgang  
P2_MIO_Digprog(1,10b)
```

Event:

```
Rem Eingangsbits einlesen und prüfen, ob Kanal 3 gesetzt ist  
If (P2_MIO_Digin_Long(1) And 100b = 1) Then  
    P2_MIO_Digout(1,5,0)    'Kanal 3 gesetzt: Bit 5 löschen  
Else  
    P2_MIO_Digout(1,5,1)    'Kanal 3 gelöscht: Bit 5 setzen  
EndIf
```


P2_MIO_Digout_Long setzt oder löscht alle Ausgänge des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.inc

P2_MIO_Digout_Long(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...15). LONG

pattern Bitmuster, nach dem die digitalen Ausgänge gesetzt werden: LONG

Bit = 0: Ausgang auf Pegel Low setzen.
Bit = 1: Ausgang auf Pegel High setzen.

Bitnr.	31...28	27 ... 24	23...8	7 6 ... 1 0
Ausgang	–	TRA4 ... TRA1	–	7 6 ... 1 0

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **P2_MIO_DigProg** als Ausgänge programmiert werden; davon ausgenommen sind

Siehe auch

[P2_MIO_Digout](#), [P2_MIO_DigProg](#)

Gültig für

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    P2_MIO_Digprog(1, 11b)    'Kanäle 7:0 als Ausgang

Event:
    P2_MIO_Digout_Long(1,128)'Den Wert 128 als Binärwert
                               'auf die Digitalkanäle ausgeben
```

P2_MIO_Digout_Long

P2_MIO_DigProg

P2_MIO_DigProg programmiert die digitalen Kanäle 0...7 des angegebenen Moduls in Gruppen zu je 4 als Ein- oder Ausgang.

Syntax

```
#Include ADwinPro_All.inc

P2_MIO_Digprog(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

Bitnr.	31...2	1	0
Kanalnr.	–	07:04	03:00

Bemerkungen

Nach dem Einschalten des Systems sind alle Kanäle als Eingänge konfiguriert, sind als Ausgänge konfiguriert.

Die Kanäle können in Gruppen zu je 4 als Ein- oder Ausgang gesetzt werden (nur 2 relevante Bits, die anderen Bits werden ignoriert). OPT- und TRA-Kanäle können nicht anders konfiguriert werden.

Siehe auch

[P2_MIO_Dig_Latch](#), [P2_MIO_Dig_Read_Latch](#), [P2_MIO_Dig_Write_Latch](#)
[P2_MIO_Digin_Long](#), [P2_MIO_Digout](#), [P2_MIO_Digout_Long](#), [P2_MIO_Get_Digout_Long](#)

Gültig für

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Kanäle 0...3 des Moduls Nr. 1 als Eingang konfigurieren
Rem und Kanäle 4...7 als Ausgang
P2_MIO_Digprog(1, 10b)
```

P2_MIO_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 = P2_MIO_Get_Digout_Long(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Inhalt des Ausgangs-Latches. Jedes Bit entspricht dem Zustand eines digitalen Ausgangs (siehe Tabelle): Bit = 0: Ausgang auf Pegel Low. Bit = 1: Ausgang auf Pegel High.	LONG

Bitnr.	31...28	27 ... 24	23...8	7 6 ... 1 0
Ausgang	–	TRA4 ... TRA1	–	7 6 ... 1 0

Bemerkungen

Ein Rücklesen der aktuellen Zustände der Ausgänge anstatt des Ausgangs-Latches ist technisch nicht möglich.

Siehe auch

[P2_MIO_Dig_Latch](#), [P2_MIO_Dig_Read_Latch](#), [P2_MIO_Dig_Write_Latch](#), [P2_MIO_DigProg](#), [P2_MIO_Digin_Long](#), [P2_MIO_Digout](#), [P2_MIO_Digout_Long](#)

Gültig für

MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Event:

```
Rem Bits 31:00 aus dem Latch zurücklesen
Par_1 = P2_MIO_Get_Digout_Long(1)
```

P2_MIO_Get_Digout_Long



3.3 Pro II: Analoge Eingänge

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit analogen Eingängen gelten.

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

Analoge Eingänge mit Multiplexer

- [P2_ADC \(Seite 34\)](#)
- [P2_ADC24 \(Seite 35\)](#)
- [P2_ADC_Read_Limit \(Seite 36\)](#)
- [P2_ADC_Set_Limit \(Seite 38\)](#)
- [P2_Read_ADC \(Seite 39\)](#)
- [P2_Read_ADC24 \(Seite 40\)](#)
- [P2_Read_ADC_SConv \(Seite 41\)](#)
- [P2_Read_ADC_SConv24 \(Seite 42\)](#)
- [P2_SE_Diff \(Seite 43\)](#)
- [P2_Seq_Init \(Seite 44\)](#)
- [P2_Seq_Read \(Seite 47\)](#)
- [P2_Seq_Read24 \(Seite 48\)](#)
- [P2_Seq_Read_Packed \(Seite 50\)](#)
- [P2_Seq_Start \(Seite 51\)](#)
- [P2_Seq_Wait \(Seite 52\)](#)
- [P2_Set_Mux \(Seite 53\)](#)
- [P2_Start_Conv \(Seite 54\)](#)
- [P2_Wait_EOC \(Seite 55\)](#)
- [P2_Wait_Mux \(Seite 56\)](#)

Analoge Eingänge mit Fast-ADC

- [P2_Burst_CRead_Unpacked1](#) (Seite 57)
- [P2_Burst_CRead_Unpacked2](#) (Seite 59)
- [P2_Burst_CRead_Unpacked4](#) (Seite 61)
- [P2_Burst_CRead_Unpacked8](#) (Seite 63)
- [P2_Burst_Init](#) (Seite 65)
- [P2_Burst_Read_Index](#) (Seite 68)
- [P2_Burst_Read](#) (Seite 70)
- [P2_Burst_Read_Unpacked1](#) (Seite 72)
- [P2_Burst_Read_Unpacked2](#) (Seite 74)
- [P2_Burst_Read_Unpacked4](#) (Seite 76)
- [P2_Burst_Read_Unpacked8](#) (Seite 78)
- [P2_Burst_Reset](#) (Seite 80)
- [P2_Burst_Start](#) (Seite 82)
- [P2_Burst_Status](#) (Seite 83)
- [P2_Burst_Stop](#) (Seite 85)
- [P2_Set_Average_Filter](#) (Seite 87)
- [P2_ADCF](#) (Seite 88)
- [P2_ADCF24](#) (Seite 89)
- [P2_ADCF_Mode](#) (Seite 90)
- [P2_ADCF_Read_Limit](#) (Seite 93)
- [P2_ADCF_Set_Limit](#) (Seite 94)
- [P2_ADCF_Reset_Min_Max](#) (Seite 95)
- [P2_ADCF_Read_Min_Max4](#) (Seite 96)
- [P2_ADCF_Read_Min_Max8](#) (Seite 98)
- [P2_Read_ADCF](#) (Seite 100)
- [P2_Read_ADCF24](#) (Seite 101)
- [P2_Read_ADCF4](#) (Seite 102)
- [P2_Read_ADCF4_24B](#) (Seite 103)
- [P2_Read_ADCF8](#) (Seite 104)
- [P2_Read_ADCF8_24B](#) (Seite 105)
- [P2_Read_ADCF4_Packed](#) (Seite 106)
- [P2_Read_ADCF8_Packed](#) (Seite 107)
- [P2_Read_ADCF32](#) (Seite 108)
- [P2_Read_ADCF_SConv](#) (Seite 109)
- [P2_Read_ADCF_SConv24](#) (Seite 110)
- [P2_Read_ADCF_SConv32](#) (Seite 111)
- [P2_Set_Gain](#) (Seite 112)
- [P2_Start_ConvF](#) (Seite 113)
- [P2_Wait_EOCF](#) (Seite 114)

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

P2_ADC

P2_ADC führt eine komplette Messung auf einem ADC des angegebenen Moduls durch. Der Rückgabewert hat 16 Bit Auflösung.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_ADC(module, input_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
input_no	Nummer (1...8 oder 1...32) des analogen Eingangs.	LONG
ret_val	Wandlungsergebnis (0...65535).	LONG

Bemerkungen

P2_ADC ist eine Zusammenstellung aufeinander folgender Funktionen:

P2_Set_Mux	→	...	→	P2_Start_Conv	→	P2_Wait_EOC	→	P2_Read_ADC
Multiplexer auf einen Eingangskanal setzen		Einschwingen des Multiplexers abwarten		A/D-Wandlung starten		Ende der Wandlung abwarten		Gewandelten Wert auslesen

Wenn der Multiplexer auf den gleichen Kanal eingestellt ist wie bei der vorherigen Messung, entfällt die Wartezeit automatisch.

Wenn Sie die Verstärkung einstellen möchten, verwenden Sie dazu den Befehl **P2_Set_Mux**.

Siehe auch

[P2_ADC24](#), [P2_ADC_Read_Limit](#), [P2_ADC_Set_Limit](#), [P2_Read_ADC](#), [P2_Set_Mux](#), [P2_Start_Conv](#), [P2_Wait_EOC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Event:

```
Rem 16Bit-Wert am analogen Eingang 4 messen
value = P2_ADC(1, 4)
```

P2_ADC24 führt eine komplette Messung auf einem ADC des angegebenen Moduls durch. Der Rückgabewert ist (linksbündig) auf 24 Bit formatiert.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_ADC24(module, input_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
input_no	Nummer (1...8 oder 1...32) des analogen Eingangs.	LONG
ret_val	Wandlungsergebnis (0...16777215 = $2^{24}-1$).	LONG

Bemerkungen

Die Anweisung **P2_ADC24** ist eine Zusammenstellung von aufeinander folgenden Funktionen:

P2_Set_Mux	...	P2_Start_Conv	P2_Wait_EOC	P2_Read_ADC24
Multiplexer auf einen Eingangskanal setzen	Einschwingen des Multiplexers abwarten	A/D-Wandlung starten	Ende der Wandlung abwarten	Gewandelten Wert auslesen

Wenn der Multiplexer auf den gleichen Kanal eingestellt ist wie bei der vorherigen Messung, entfällt die Wartezeit automatisch.

Wenn Sie die Verstärkung einstellen möchten, verwenden Sie dazu den Befehl **P2_Set_Mux**.

Wenn ein Messwert eine geringere Auslösung als 24 Bit hat, werden im Rückgabewert die „fehlenden“ Bits rechts mit Nullen aufgefüllt. Beispielsweise steht der Messwert eines 18 Bit-ADC in den Bits 6...23 des Rückgabewerts; hier ist der Messwert um 6 Bits nach links verschoben und die Bits 0...5 sind Null.

Bit-Nr.	31...24	23...6	05...00
Inhalt	0	18-Bit Messwert	0

Siehe auch

[P2_ADC](#), [P2_ADC_Read_Limit](#), [P2_ADC_Set_Limit](#), [P2_Read_ADC24](#), [P2_Set_Mux](#), [P2_Start_Conv](#), [P2_Wait_EOC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Event:

```
Rem 24Bit-Wert am analogen Eingang 4 messen
value = P2_ADC24(1, 4)
```

P2_ADC24

P2_ADC_Read_Limit

P2_ADC_Read_Limit liest die Flags für Grenzwertüber- und unterschreitungen auf 16 Eingangskanälen des angegebenen Moduls aus.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_ADC_Read_Limit(module, ch_group)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ch_group	Kanalgruppe zu je 16 Kanälen: 1: Kanäle 1...16 2: Kanäle 17...32	
ret_val	Bitmuster aus den Flags für Grenzwertüber- und unterschreitungen:	LONG

Überschreitung der Obergrenze								
ch_group	Bit Nr.	31	30	29	...	18	17	16
1	Kanalnr.	16	15	14	...	3	2	1
2	Kanalnr.	32	31	30	...	19	18	17
Unterschreitung der Untergrenze								
ch_group	Bit Nr.	15	14	13	...	2	1	0
1	Kanalnr.	16	15	14	...	3	2	1
2	Kanalnr.	32	31	30	...	19	18	17

Bemerkungen

Sie stellen die Grenzwerte mit **P2_ADC_Set_Limit** ein.

Das Lesen der Flags setzt alle Flags der Kanalgruppe auf Null zurück.

Wir empfehlen, im Abschnitt **Init**: die Flags einmal zu lesen, damit eventuelle vorherige Grenzwertüber- und unterschreitungen gelöscht sind. Dies ist bei einem extern gesteuerten Prozess besonders wichtig.

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_ADC_Set_Limit](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Init:
  P2_SE_Diff(module,1)      'Differentielle Eingänge
  P2_ADC_Set_Limit(module, 2, 42768, 256) 'Grenzwerte Kanal 2
  P2_Seq_Init(module, 3, 0, 10b, 0) 'continuous max mode, Kanal 2
  P2_Seq_Start(Shift_Left(1, module-1)) 'Messreihe starten
  P2_Seq_Wait(module)
  Rem Flags durch Lesen rücksetzen
  flags = P2_ADC_Read_Limit(module, 1)

Event:
  flags = P2_ADC_Read_Limit(module,1) 'Flags 1...16 lesen
  If ((flags And 10b) = 10b) Then
    Rem Untergrenze auf Kanal 2 ist unterschritten
    Inc Par_1
  EndIf
  If ((flags And 20000h) = 20000h) Then
    Rem Obergrenze auf Kanal 2 ist überschritten
    Inc Par_2
  EndIf
```

P2_ADC_Set_Limit

P2_ADC_Set_Limit setzt den oberen und unteren Grenzwert für einen analogen Eingang des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_ADC_Set_Limit(module, input_no, high, low)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
input_no	Nummer (1...8 oder 1...32) des analogen Eingangs.	LONG
high	Oberer Grenzwert (0...65535) des Kanals. Voreinstellung: 65535.	LONG
low	Unterer Grenzwert (0...65535) des Kanals. Voreinstellung: 0.	LONG

Bemerkungen

Wenn ein Messwert den oberen Grenzwert überschreitet, wird für diesen Kanal ein Flag gesetzt, das mit **P2_ADC_Read_Limit** gelesen und zurückgesetzt wird.

In gleicher Weise wird ein Flag für den Kanal gesetzt, wenn ein Messwert den unteren Grenzwert unterschreitet.

Grenzwertübertretungen können keine Event-Signale auslösen.

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_ADC_Read_Limit](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
#Define module 1  
Dim flags As Long  
  
Init:  
P2_SE_Diff(module,1) 'Differentielle Eingänge  
P2_ADC_Set_Limit(module, 2, 42768, 256) 'Grenzwerte Kanal 2  
P2_Seq_Init(module, 3, 0, 10b, 0) 'continuous max mode, Kanal 2  
P2_Seq_Start(Shift_Left(1, module-1)) 'Messreihe starten  
P2_Seq_Wait(module)  
Rem Flags durch Lesen rücksetzen  
flags = P2_ADC_Read_Limit(module, 1)  
  
Event:  
flags = P2_ADC_Read_Limit(module,1) 'Flags 1...16 lesen  
If ((flags And 10b) = 10b) Then  
Rem Untergrenze auf Kanal 2 ist unterschritten  
Inc Par_1  
EndIf  
If ((flags And 20000h) = 20000h) Then  
Rem Obergrenze auf Kanal 2 ist überschritten  
Inc Par_2  
EndIf
```

P2_Read_ADC liest das Wandlungsergebnis des angegebenen Moduls aus. Das Ergebnis hat 16 Bit Auflösung.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_Read_ADC(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Im ADC-Register enthaltener Messwert (0...65535).	LONG

Bemerkungen

- / -

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_Start_Conv](#), [P2_Wait_EOC](#), [P2_ADC_Read_Limit](#), [P2_ADC_Set_Limit](#), [P2_Read_ADC_SConv](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Init:
    P2_Set_Mux(1,0100000010b) 'MUX auf Eing. 3, Verstärkung 2 setzen

Event:
    P2_Start_Conv(1) 'Start AD-Wandlung
    P2_Wait_EOC(1) 'Warten auf Wandlung-Ende
    value1 = P2_Read_ADC(1) 'Wert vom ADC einlesen
```

P2_Read_ADC

P2_Read_ADC24

P2_Read_ADC24 liest das Wandlungsergebnis des angegebenen Moduls aus. Der Rückgabewert ist (linksbündig) auf 24 Bit formatiert.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_Read_ADC24(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Im ADC-Register enthaltener Messwert (0...16777215 = $2^{24}-1$).	LONG

Bemerkungen

Wenn ein Messwert eine geringere Auslösung als 24 Bit hat, werden im Rückgabewert die „fehlenden“ Bits rechts mit Nullen aufgefüllt. Beispielsweise steht der Messwert eines 18 Bit-ADC in den Bits 6...23 des Rückgabewerts; hier ist der Messwert um 6 Bits nach links verschoben und die Bits 0...5 sind Null.

Bit-Nr.	31...24	23...6	05...00
Inhalt	0	18-Bit Messwert	0

Siehe auch

[P2_ADC24](#), [P2_Start_Conv](#), [P2_Wait_EOC](#), [P2_ADC_Read_Limit](#), [P2_ADC_Set_Limit](#), [P2_Read_ADC_SConv24](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Init:
    P2_Set_Mux(1,0100000010b)'MUX auf Eing. 3, Verstärkung 2 setzen

Event:
    P2_Start_Conv(1) 'Start AD-Wandlung
    P2_Wait_EOC(1) 'Warten auf Wandlung-Ende
    value1 = P2_Read_ADC24(1)'24Bit-Wert vom ADC einlesen
```

P2_Read_ADC_SConv liest das Wandlungsergebnis des angegebenen Moduls aus und startet sofort eine neue Konvertierung.
Der Rückgabewert hat eine Auflösung von 16 Bit.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_Read_ADC_SConv(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Im ADC-Register enthaltener Messwert (0...65535).	LONG

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_Read_ADC](#), [P2_Read_ADC_SConv24](#), [P2_Start_Conv](#), [P2_Wait_EOC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Init:
  i = 1
  P2_Set_Mux(1,0100000010b)'MUX auf Eing. 3, Verstärkung 2 setzen
  Rem Einschwingen des Multiplexers abwarten, hier 4 µs
  P2_Sleep(400)
  P2_Start_Conv(1)          'A/D-Wandler starten

Event:
  P2_Wait_EOC(1)
  Data_1[i] = P2_Read_ADC_SConv(1)'A/D-Wandler auslesen+starten
  Inc(i)          'Index erhöhen
  If (i=1001) Then End      'Nach 1000 Messwerten Prozess beenden
```

P2_Read_ADC_SConv

P2_Read_ADC_SConv24

P2_Read_ADC_SConv24 liest das Wandlungsergebnis des angegebenen Moduls aus und startet sofort eine neue Konvertierung.

Der Rückgabewert hat eine Auflösung von 24 Bit.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_Read_ADC_SConv24(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Im ADC-Register enthaltener Messwert (0...16777215 = $2^{24}-1$).	LONG

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_Read_ADC](#), [P2_Read_ADC_SConv](#), [P2_Start_Conv](#), [P2_Wait_EOC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
Dim i As Long  
Dim Data_1[1000] As Long 'Deklaration  
  
Init:  
i=1  
P2_Set_Mux(1,0100000010b)'MUX auf Eing. 3, Verstärkung 2 setzen  
Rem Einschwingen des Multiplexers abwarten, hier 4 µs  
P2_Sleep(400)  
P2_Start_Conv(1) 'A/D-Wandler starten  
  
Event:  
P2_Wait_EOC(1)  
Data_1[i] = P2_Read_ADC_SConv24(1)'A/D-Wandler 24 Bit  
'auslesen + starten  
Inc(i) 'Index erhöhen  
If (i=1001) Then End 'Nach 1000 Messwerten Prozess beenden
```

P2_SE_Diff stellt die Betriebsart single ended oder differentiell für alle Analog-Eingänge auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.Inc

P2_SE_Diff(module, mode)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
mode	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.

Siehe auch

[P2_ADC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_SE_Diff(1,0)      'Modul mit der Adresse 1 wird
                     'auf SE gesetzt
P2_SE_Diff(2,1)      'Modul mit der Adresse 2 wird
                     'auf DIFF gesetzt
```

P2_SE_Diff

P2_Seq_Init

P2_Seq_Init initialisiert das angegebene Modul für den Betrieb mit Ablaufsteuerung.

Eingestellt werden der Arbeitsmodus, der Verstärkungsfaktor, die Kanäle und die Einschwingzeit des Multiplexers (für alle Kanäle gleich).

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_Seq_Init(module, mode, gain, channels, mux_time)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
mode	Arbeitsmodus der Ablaufsteuerung: 0: Einzelmessung (Default), ohne Ablaufsteuerung. 1: Modus „single shot“, einfacher Messzyklus. 2: Modus „continuous“, regelmäßiger Messzyklus. 3: Modus „continuous max“ Messzyklus mit maximaler Geschwindigkeit.	LONG
gain	Verstärkungsfaktor (nur für die Modi 1...3): 0 Faktor = 1, Spannungsbereich -10V...+10V. 1 Faktor = 2, Spannungsbereich -5V...+5V. 2 Faktor = 4, Spannungsbereich -2,5V...+2,5V. 3 Faktor = 8, Spannungsbereich -1,25V...+1,25V.	LONG
channels	Bitmuster, das die zu wandelnden Kanäle, die Messgruppe, bestimmt. Bit = 0: Kanal nicht wandeln. Bit = 1: Kanal wandeln.	LONG

Bitnr.	31	...	7	...	2	1	0
Kanal-Nr.	32	...	8	...	3	2	1

muxtime	Anzahl der Zeiteinheiten, aus der sich die Einschwingzeit der Ablaufsteuerung ergibt: 0: Werkseinstellung (125 = 2,5µs). 125...2 ³¹ : Einschwingzeit in Einheiten von 20ns.	LONG
----------------	--	------

Bemerkungen

Nach dem Einschalten ist der Modus 0 aktiv.

Die Modi 1...3 aktivieren die Ablaufsteuerung des Moduls, Einzelmessungen mit **P2_ADC** sind dann nicht möglich. Die Ablaufsteuerung führt an mehreren Kanälen nacheinander eine Wandlung durch. Die Steuerung bezieht sich immer nur auf die mit **channels** definierte Auswahl an Kanälen.

Die Modi unterscheiden sich wie folgt:

Modus	Art der Messung
0 Normal:	Standard: Einzelne Messung an einem Kanal ohne Ablaufsteuerung, siehe P2_ADC .

- 1 single shot: Die Ablaufsteuerung wird mit **P2_Seq_Start** gestartet; die Ablaufsteuerung endet, sobald die gewählten Kanäle je einmal gewandelt sind.
Das Ende der Ablaufsteuerung wird mit **P2_Seq_Wait** abgefragt und die Messwerte mit **P2_Seq_Read** eingelesen.
- 2 continuous: Die Ablaufsteuerung wandelt für jeden Prozesszyklus einen Satz neuer Messwerte.
Die Wandlung wird im Abschnitt **Init:** gestartet, mit **P2_Seq_Start** als letztem Befehl. Das Wandlungsende (für alle Kanäle) wird automatisch mit dem Beginn des nächsten Prozesszyklus synchronisiert. Daher können – und sollten auch – alle Messwerte bereits zu Beginn des Prozesszyklus mit **P2_Seq_Read** gelesen werden.
- 3 continuous max: Die Ablaufsteuerung wandelt ununterbrochen neue Messwerte an den gewählten Kanälen, d.h. Wandlung und Prozesszyklus laufen asynchron.
Die Wandlung wird mit **P2_Seq_Start** gestartet. Im Prozesszyklus wird mit **P2_Seq_Read** der jeweils neueste Messwert gelesen.

Beachten Sie beim Modus 2 (continuous): Die Synchronisierung geschieht nur einmal und gilt nur für die gerade eingestellte Zykluszeit (**Processdelay**). Wenn sich das Zeitverhalten ändert, z.B. durch Ändern der Zykluszeit, geht die Synchronisation verloren. Als Folge werden entweder Messwerte zu früh und damit mehrfach gelesen, oder es gehen Messwerte verloren, weil sie beim Auslesen bereits von neueren Werten überschrieben sind.

Sie können in der Messgruppe eine beliebige Auswahl aus den 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.

Der Status- und die Lese-Anweisungen beziehen sich immer und ausschließlich auf die Gruppe der hier ausgewählten Kanäle.

Bei den 32kanaligen Modulen müssen die Eingänge mit **P2_SE_Diff** als single ended oder differentiell eingestellt werden.

Wenn der Innenwiderstand der Spannungsquelle des Messsignals zu groß ist, genügt die voreingestellte Einschwingzeit des Multiplexers nicht mehr aus für eine genaue Messung. Sie können die Einschwingzeit des Multiplexers mit dem Parameter **mux_time** verändern.

Die Einstellung der Einschwingzeit beeinflusst die Genauigkeit der Messergebnisse in starkem Maß. Tendenziell ergeben kürzere Einschwingzeiten ungenauere und längere Einschwingzeiten genauere Messergebnisse.

Die Einschwingzeit berechnet sich nach folgender Formel:

$$\text{Einschwingzeit} = \text{muxtime} \cdot 20\text{ns} + \text{Wandlerzeit}$$

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

Siehe auch

[P2_ADC](#), [P2_Seq_Read](#), [P2_Seq_Read24](#), [P2_Seq_Read_Packed](#),
[P2_Seq_Start](#), [P2_Seq_Wait](#)



Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Define module 1
#include ADwinPro_All.inc

Dim Data_1[16] As Long At DM_Local

Init:
    P2_SE_Diff(module,0)      'Eingänge auf single ended stellen
    Rem Ablaufsteuerung: Continuous Mode, Verstärkungsfaktor 1
    Rem ungeradzahlige Kanäle des Moduls AIN-32
    Rem Standard-Einschwingzeit
    P2_Seq_Init(module,3,0,55555555h,0)
    Rem Messsequenzen auf dem Modul starten
    P2_Seq_Start(Shift_Left(1,module-1))
    P2_Seq_Wait(module)      'Warten, bis alle angegebenen Kanäle
                             'einmal gemessen sind

Event:
    Rem Messwerte lesen und in Data_1 kopieren
    P2_Seq_Read(module,16,Data_1,1)
```

P2_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

P2_Seq_Read(module, count, array[], array_idx)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Gerade Anzahl (2...32) der zu lesenden Messwerte. Eine ungerade Anzahl ist nicht erlaubt.	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden.	ARRAY LONG FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1...n).	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **P2_Seq_Init** die Ablaufsteuerung des Moduls aktiviert wurde.

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

Siehe auch

[P2_Seq_Init](#), [P2_Seq_Read](#), [P2_Seq_Read24](#), [P2_Seq_Start](#), [P2_Seq_Wait](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim Data_1[16] As Long At DM_Local

Init:
    P2_SE_Diff(module, 0)          'Single-Ended Eingänge
    Rem Ablaufsteuerung: Continuous Mode, Verstärkungsfaktor 1
    Rem ungeradzahlige Kanäle, Standard-Einschwingzeit
    P2_Seq_Init(module, 3, 0, 55555555h, 0)
    P2_Seq_Start(Shift_Left(1, module-1)) 'Messesequenzen starten
    P2_Seq_Wait(module)             'Warten, bis einmal alle angegebenen
                                    'Kanäle gemessen wurden

Event:
    Rem Aktuelle Messwerte von dem Modul in Data_1 umkopieren
    P2_Seq_Read(module, 16, Data_1, 1)
```

P2_Seq_Read

P2_Seq_Read24

P2_Seq_Read24 kopiert eine bestimmte Anzahl an Messwerten (24 Bit) von dem angegebenen Modul in ein Ziel-Feld.

In jedes Feldelement wird jeweils 1 Messwert kopiert.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_Seq_Read24(module, count, array[ ], array_idx)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>count</code>	Anzahl (1...32) der zu lesenden Messwerte.	LONG
<code>array[]</code>	Ziel-Feld, in das die Messwerte übertragen werden.	ARRAY LONG FLOAT
<code>array_idx</code>	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1...n).	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **P2_Seq_Init** die Ablaufsteuerung des Moduls aktiviert wurde.

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

Wenn ein Messwert eine geringere Auslösung als 24 Bit hat, werden im Rückgabewert die „fehlenden“ Bits rechts mit Nullen aufgefüllt.

Beispielsweise steht der Messwert eines 18 Bit-ADC in den Bits 6...23 des Rückgabewerts; hier ist der Messwert um 6 Bits nach links verschoben und die Bits 0...5 sind Null.

Bit-Nr.	31...24	23...6	05...00
Inhalt	0	18-Bit Messwert	0

Siehe auch

[P2_Seq_Init](#), [P2_Seq_Read](#), [P2_Seq_Read_Packed](#), [P2_Seq_Start](#), [P2_Seq_Wait](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 1
Dim Data_1[16] As Long At DM_Local

Init:
    P2_SE_Diff(module,0)      'Single-Ended Eingänge
    Rem Ablaufsteuerung: Continuous Mode, Verstärkungsfaktor 1
    Rem ungeradzahlige Kanäle, Standard-Einschwingzeit
    P2_Seq_Init(module,3,0,55555555h,0)
    P2_Seq_Start(Shift_Left(1, module-1))'Messsequenzen starten
    P2_Seq_Wait(module)       'Warten, bis einmal alle angegebenen
                                'Kanäle gemessen wurden

Event:
    Rem Aktuelle Messwerte von dem Modul in Data_1 umkopieren
    P2_Seq_Read24(module,16,Data_1,1)
```

P2_Seq_Read_Packed

P2_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
```

```
P2_Seq_Read_Packed(module, count, array[], array_idx)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu lesenden Messwerte-Paare (1...16).	LONG
array[]	Ziel-Feld, in das die Messwerte-Paare übertragen werden.	ARRAY LONG FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1...n).	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **P2_Seq_Init** die Ablaufsteuerung des Moduls aktiviert wurde.

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

[P2_Seq_Init](#), [P2_Seq_Read](#), [P2_Seq_Read24](#), [P2_Seq_Start](#), [P2_Seq_Wait](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

```
Dim Data_1[8], Data_2[8] As Long At DM_Local
```

Init:

```
P2_SE_Diff(1,0)           'Single-Ended Eingänge auf Modul 1
P2_SE_Diff(5,0)           'Single-Ended Eingänge auf Modul 5
Rem Module 1+5: Ablaufsteuerung auf Continuous Mode, Verstär-
Rem kungsfaktor 1, geradzahlige Kanäle (2...16) des Moduls,
Rem Standard-Einschwingzeit
P2_Seq_Init(1,3,0,0AAAAh,0)
P2_Seq_Init(5,3,0,0AAAAh,0)
P2_Seq_Start(10001b)      'Messsequenz auf Modulen 1+5 starten
P2_Seq_Wait(1)            'Warten, bis alle angegebenen
                          'Kanäle einmal gemessen wurden
```

Event:

```
Rem 16 Messwerte holen und in Data_1, Data_2 kopieren
P2_Seq_Read_Packed(1,8,Data_1,1)
P2_Seq_Read_Packed(5,8,Data_2,1)
```

P2_Seq_Start startet die Ablaufsteuerung auf allen angegebenen Modulen gleichzeitig.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_Seq_Start(module_pattern)
```

Parameter

module_ Bitmuster zum Auswählen der Moduladressen: LONG
pattern Bit = 0: Moduladresse ignorieren.
 Bit = 1: Moduladresse ansprechen.

Bitmuster	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Siehe auch

[P2_Seq_Init](#), [P2_Seq_Read](#), [P2_Seq_Read24](#), [P2_Seq_Wait](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

```
#Define module 4 'Moduladresse
```

```
Dim Data_1[32] As Long At DM_Local
```

```
Dim i As Long
```

Init:

```
P2_SE_Diff(module,0) 'Single-Ended Eingänge
Rem Ablaufsteuerung auf Single Shot,
Rem Verstärkungsfaktor 1, alle Kanäle des Moduls,
Rem Standard-Einschwingzeit
P2_Seq_Init(module,1,0,0FFFFFFFh,0)
P2_Seq_Start(Shift_Left(1,module-1)) 'Messsequenz starten
```

Event:

```
P2_Seq_Wait(module) 'Ende der Messung abwarten
P2_Seq_Read(module,32,Data_1,1) 'Alle 32 Kanäle einlesen ...
For i=1 To 32
  Rem Digit in Volt umrechnen und speichern
  Data_1[i] = (Data_1[i]-32768)*20/65536
Next i
P2_Seq_Start(Shift_Left(1,module-1)) 'Messsequenz starten
```

P2_Seq_Start

P2_Seq_Wait

P2_Seq_Wait wartet, bis die Ablaufsteuerung auf dem angegebenen Modul alle Kanäle der Messgruppen gewandelt und gespeichert hat.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_Seq_Wait(module)
```

Parameter

module Eingestellte Moduladresse (1...15).

LONG

Bemerkungen

Wenn Ablaufsteuerungen auf mehreren Modulen gleichzeitig (und mit gleichen Parametern) gestartet wurden, enden sie auch gleichzeitig.

Siehe auch

[P2_Seq_Init](#), [P2_Seq_Read](#), [P2_Seq_Read24](#), [P2_Seq_Start](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
#Define module 4                    'Moduladresse  
  
Dim Data_1[32] As Long At DM_Local  
Dim Data_2[32] As Float At DM_Local  
  
Dim i As Long  
  
Init:  
  Processdelay=100000  
  P2_SE_Diff(module,1)            'Eingänge differentiell  
  Rem Ablaufsteuerung auf Single Shot,  
  Rem Verstärkungsfaktor 1, alle Kanäle des Moduls,  
  Rem Standard-Einschwingzeit  
  P2_Seq_Init(module,1,0,0FFFFFFFh,0)  
  P2_Seq_Start(Shift_Left(1,module-1)) 'Messsequenz starten  
  
Event:  
  P2_Seq_Wait(module)'Ende der Messung abwarten  
  P2_Seq_Read(module,32,Data_1,1) 'Alle 32 Kanäle einlesen ...  
  For i=1 To 32  
    Rem Digit in Volt umrechnen und speichern  
    Data_2[i] = (Data_1[i]-32768)*20/65536  
  Next i  
  P2_Seq_Start(Shift_Left(1,module-1)) 'Messsequenz starten
```


P2_Set_Mux stellt den Multiplexer des angegebenen Moduls auf einen bestimmten Eingang und eine bestimmte Verstärkung ein.

Syntax

```
#Include ADwinPro_All.Inc

P2_Set_Mux(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...15). LONG

pattern Bitmuster zur Einstellung des Multiplexers (siehe Tabelle); die Bits 8...9 stellen die Verstärkung ein, die Bits 0...4 die Nummer des Eingangs. LONG

Bit-Nr.								
31:7	9	8	7...5	4	3	2	1	0
ohne Funktion	Verstärkung	–	Multiplexer-Eingang					
	1 = 00b	–	Eingang 1: 00000b					
	2 = 01b		Eingang 2: 00001b					
	4 = 10b		...					
	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 Binärformat verwenden oder sie in Hexadezimal- oder Dezimal-Format umrechnen. Beachten Sie für Hex- und Binärformat die angehängten Buchstaben **h** und **b**.

Bitte beachten Sie die erforderliche Einschwingzeit des Multiplexers (siehe Hardware-Dokumentation). Stellen Sie sicher, dass zwischen der Neueinstellung des Multiplexers und dem Konvertierungsbeginn mindestens diese Zeit vergeht.

Siehe auch

[P2_ADC](#), [P2_Start_Conv](#), [P2_Wait_EOC](#), [P2_Read_ADC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Init:
P2_Set_Mux(1,0100000010b)'MUX auf Eing. 3, Verstärkung 2 setzen
Rem Einschwingen des Multiplexers abwarten, hier 4 µs
P2_Sleep(400)

Event:
P2_Start_Conv(1)                   'Start AD-Wandlung
P2_Wait_EOC(1)                    'Warten auf Wandlung-Ende
valuel = P2_Read_ADC(1)           'Wert vom ADC einlesen
```

P2_Set_Mux

P2_Start_Conv

P2_Start_Conv startet die Wandlung auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_Start_Conv(module)
```

Parameter

module Eingestellte Moduladresse (1...15).

LONG

Bemerkungen

- / -

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_Read_ADC](#), [P2_Wait_EOC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
Dim value As Long                    'Deklaration  
  
Init:  
    P2_Set_Mux(1,0100000010b)'MUX auf Eing. 3, Verstärkung 2 setzen  
    Rem Einschwingen des Multiplexers abwarten, hier 4 µs  
    P2_Sleep(400)  
  
Event:  
    P2_Start_Conv(1)                'Start AD-Wandlung auf Kanal 1  
    P2_Wait_EOC(1)                  'Warten auf Wandlung-Ende  
    value = P2_Read_ADC(1)          'Wert vom ADC einlesen
```

P2_Wait_EOC wartet, bis die Wandlung auf dem angegebenen Modul abgeschlossen ist.

Syntax

```
#Include ADwinPro_All.Inc

P2_Wait_EOC(module)
```

Parameter

module Eingestellte Moduladresse (1...15).

LONG

Bemerkungen

- / -

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_Start_Conv](#), [P2_Read_ADC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Init:
  P2_Set_Mux(1,0100000010b) 'MUX auf Eing. 3, Verstärkung 2 setzen
  Rem Einschwingen des Multiplexers abwarten, hier 4 µs
  P2_Sleep(400)

Event:
  P2_Start_Conv(1)                    'Start AD-Wandlung
  P2_Wait_EOC(1)                    'Warten auf das Ende der Konvertierung
  value = P2_Read_ADC(1)            'Wert vom ADC einlesen
```

P2_Wait_EOC

P2_Wait_Mux

P2_Wait_Mux wartet, bis das Einschwingen des Multiplexers auf dem angegebenen Modul abgeschlossen ist.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_Wait_Mux(module)
```

Parameter

module Eingestellte Moduladresse (1...15).

LONG

Bemerkungen

Wenn Sie den Multiplexer mit **P2_Set_Mux** auf einen neuen Kanal einstellen, dauert es eine bestimmte Zeit, bis der Multiplexer eingeschwungen ist. Der Befehl **P2_Wait_Mux** wartet auf diesen Zeitpunkt, so dass Sie direkt anschließend eine Signalwandlung starten können.

Wenn der Multiplexer auf den gleichen Kanal eingestellt ist wie bei der vorherigen Wandlung, oder wenn der Multiplexer bereits eingeschwungen ist, entfällt die Wartezeit automatisch.

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_Set_Mux](#), [P2_Start_Conv](#), [P2_Read_ADC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E

Beispiel

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

Init:

```
P2_Seq_Init(module,0,0,0,0) 'switch off sequential control mode  
P2_Set_Mux(module,0b)      'set multiplexer to input 1, gain 1  
P2_Wait_Mux(module)  
P2_Start_Conv(module)      'start AD conversion  
Processdelay=30000         'cycle-time 0.1 ms
```

Event:

```
P2_Set_Mux(module,0100000001b) 'set MUX to input 2, gain 2  
P2_Wait_EOC(module)             'wait for end of conversion  
Par_1 = P2_Read_ADC(module)     'read channel value 1 from the ADC  
P2_Wait_Mux(module)             'wait for end of settling time  
P2_Start_Conv(module)           'start AD conversion  
P2_Set_Mux(module,0b)           'set MUX to input 1, gain 1  
P2_Wait_EOC(module)             'wait for end of conversion  
Par_2 = P2_Read_ADC(module)     'read channel value 2 from the ADC  
  
P2_Wait_Mux(module)             'wait for end of settling time  
P2_Start_Conv(module)           'start AD conversion
```

P2_Burst_CRead_Unpacked1 kopiert eine Anzahl der zuletzt gemessenen Messwerte eines Kanals aus dem Speicher des angegebenen Moduls in ein Feld.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Unpacked1(module, count, array1[],
    array_idx, flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu übertragenden Messwerte. Die Anzahl muss durch 8 teilbar sein.	LONG
array1[]	Ziel-Feld, in das die Messwerte übertragen werden. Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
flowrate	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit 1 Kanal eingerichtet wurde (siehe **P2_Burst_Init**, Parameter **mode**, **channels**).

Die Anweisung liest die Anzahl **count** der zuletzt im Modulspeicher abgelegten Messwerte. **count** sollte etwa um den Faktor 2 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der durchzuführenden Messungen (**samples**).

Die Anweisung legt die Messwerte einzeln nacheinander in den Elementen des Zielfelds ab.

In hochpriorien Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter **flowrate** muss trotzdem angegeben werden.

Je höher Sie – bei einem niederpriorien Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_CRead_Unpacked2](#), [P2_Burst_CRead_Unpacked4](#), [P2_Burst_CRead_Unpacked8](#), [P2_Burst_Start](#), [P2_Burst_Status](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

P2_Burst_CRead_Unpacked1



Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000] As Long
Dim pattern As Long

Init:
    Rem Kont. Burst-Messreihe für Kanal 1 einrichten mit 20ns
    Rem Periodendauer, 2^26 Daten speichern ab Adresse 0.
    P2_Burst_Init (module,1,0,67108864,1,010b)
    Rem Burst-Messreihe starten
    pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
    P2_Burst_Start(pattern)
    Processdelay=10000000

Event:
    Rem Die letzten 1000 Messwerte des Kanals (langsam) lesen und in
    Rem Data_1 ablegen
    P2_Burst_CRead_Unpacked1(module,1000,Data_1,1,1)
```

P2_Burst_CRead_Unpacked2 kopiert eine Anzahl der zuletzt gemessenen Messwerte zweier Kanäle aus dem Speicher des angegebenen Moduls in 2 Felder.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_Burst_CRead_Unpacked2(module, count, array1[],  
    array2[], array_idx, flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal. Die Anzahl muss durch 4 teilbar sein.	LONG
arrayx[]	Ziel-Felder für die Messwerte der Kanäle 1 und 2. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
flowrate	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit 2 Kanälen eingerichtet wurde (siehe **P2_Burst_Init**, Parameter **mode**, **channels**).

Die Anweisung liest die Anzahl **count** der zuletzt im Modulspeicher abgelegten Messwerte. **count** sollte etwa um den Faktor 2 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der durchzuführenden Messungen (**samples**).

Die Anweisung legt die Messwerte einzeln nacheinander in den Elementen der Zielfelder ab.

In hochpriorien Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter **flowrate** muss trotzdem angegeben werden.

Je höher Sie – bei einem niederpriorien Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_CRead_Unpacked1](#), [P2_Burst_CRead_Unpacked4](#), [P2_Burst_CRead_Unpacked8](#), [P2_Burst_Start](#), [P2_Burst_Status](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

P2_Burst_CRead_Unpacked2



Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000] As Long
Dim Data_2[1000] As Long
Dim pattern As Long

Init:
    Rem Kont. Burst-Messreihe für Kanäle 1...2 einrichten mit 60ns
    Rem Periodendauer, 2^26 Messwerte je Kanal ab Adresse 0.
    P2_Burst_Init (module,3,0,67108860,3,010b)
    Rem Burst-Messreihe starten
    pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
    P2_Burst_Start(pattern)
    P2_Set_LED(module,1)
    Processdelay=1000000

Event:
    Rem Die letzten 1000 Messwerte je Kanal (langsam) lesen und in
    Rem den Feldern Data_1 bis Data_2 ablegen
    P2_Burst_CRead_Unpacked2(module,1000,Data_1,Data_2,1,1)
```


P2_Burst_CRead_Unpacked4 kopiert eine Anzahl der zuletzt gemessenen Messwerte von 4 Kanälen aus dem Speicher des angegebenen Moduls in 4 Felder.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_Burst_CRead_Unpacked4(module, count, array1[],  
    array2[], array3[], array4[], array_idx,  
    flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal. Die Anzahl muss durch 2 teilbar sein.	LONG
arrayx[]	Ziel-Felder für die Messwerte der Kanäle 1...4. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
flowrate	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit 4 Kanälen eingerichtet wurde (siehe **P2_Burst_Init**, Parameter **mode**, **channels**).

Die Anweisung liest die Anzahl **count** der zuletzt im Modulspeicher abgelegten Messwerte. **count** sollte etwa um den Faktor 2 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der durchzuführenden Messungen (**samples**).

Die Anweisung legt die Messwerte einzeln nacheinander in den Elementen der Zielfelder ab.

In hochpriorien Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter **flowrate** muss trotzdem angegeben werden.

Je höher Sie – bei einem niederpriorien Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_CRead_Unpacked1](#), [P2_Burst_CRead_Unpacked2](#), [P2_Burst_CRead_Unpacked8](#), [P2_Burst_Start](#), [P2_Burst_Status](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

P2_Burst_CRead_Unpacked4



Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000], Data_2[1000] As Long
Dim Data_3[1000], Data_4[1000] As Long
Dim pattern As Long

Init:
    Rem Kont. Burst-Messreihe für Kanäle 1..4 einrichten mit 40ns
    Rem Periodendauer, 2^25 je Kanal speichern ab Adresse 0.
    P2_Burst_Init (module,15,0,3355444,2,010b)
    Rem Burst-Messreihe starten
    pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
    P2_Burst_Start(pattern)
    Processdelay=50000000

Event:
    Rem Die letzten 1000 Messwerte je Kanal (schnell) lesen und in
    Rem den Feldern Data_1 bis Data_4 ablegen
    P2_Burst_CRead_Unpacked4(module,1000,Data_1,Data_2,Data_3,
        Data_4,1,3)
```

P2_Burst_CRead_Unpacked8 kopiert eine Anzahl der zuletzt gemessenen Messwerte von 8 Kanälen aus dem Speicher des angegebenen Moduls in 8 Felder.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_Burst_CRead_Unpacked8(module, count, array1[],  
    array2[], array3[], array4[], array5[], array6[],  
    array7[], array8[], array_idx, flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal.	LONG
arrayx[]	Ziel-Felder für die Messwerte der Kanäle 1...8. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
flowrate	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit 8 Kanälen eingerichtet wurde (siehe **P2_Burst_Init**, Parameter **mode**, **channels**).

Die Anweisung liest die Anzahl **count** der zuletzt im Modulspeicher abgelegten Messwerte. **count** sollte etwa um den Faktor 2 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der durchzuführenden Messungen (**samples**).

Die Anweisung legt die Messwerte einzeln nacheinander in den Elementen der Zielfelder ab.

In hochpriorien Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter **flowrate** muss trotzdem angegeben werden.

Je höher Sie – bei einem niederpriorien Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_CRead_Unpacked1](#), [P2_Burst_CRead_Unpacked2](#), [P2_Burst_CRead_Unpacked4](#), [P2_Burst_Start](#), [P2_Burst_Status](#)

Gültig für

Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

P2_Burst_CRead_Unpacked8



Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000], Data_2[1000] As Long At DM_Local
Dim Data_3[1000], Data_4[1000] As Long At DM_Local
Dim Data_5[1000], Data_6[1000] As Long At DM_Local
Dim Data_7[1000], Data_8[1000] As Long At DM_Local
Dim pattern As Long

Init:
  Rem Kont. Burst-Messreihe für Kanäle 1...8 einrichten mit 40ns
  Rem Periodendauer, 1Mio. Messwerte je Kanal speichern ab
  Rem Adresse 100.
  P2_Burst_Init (module,255,100,1000000,2,010b)
  Rem Burst-Messreihe starten
  pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
  P2_Burst_Start(pattern)
  Processdelay=10000000

Event:
  Rem Die letzten 10000 Messwerte je Kanal (langsam) lesen und in
  Rem den Feldern Data_1 bis Data_8 ablegen
  P2_Burst_CRead_Unpacked8(module,1000,Data_1,Data_2,Data_3,
    Data_4,Data_5,Data_6,Data_7,Data_8,1,3)
```

P2_Burst_Init legt die Kennwerte für eine Burst-Messreihe auf dem angegebenen Modul fest.

Die Kennwerte sind: Anzahl und Nummern der Messkanäle, Startadresse im Modulspeicher, Anzahl der Messungen, Periodendauer der Messreihe, Art der Burst-Messreihe.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_Init (module, channels, startadr, samples,
               pulses, mode)
```

Parameter

module Eingestellte Moduladresse (1...15). LONG

channels legt Anzahl und Nummern der Messkanäle fest. LONG
Nur die unten aufgeführten Werte sind zulässig.
Gemeinsam mit der Speichergröße ergibt sich die max. Anzahl der Messwerte pro Kanal:

chann els	Kanalnr.	max. Anzahl der Messwerte pro Kanal für startadr=0:
1	1	134217720 = 7FFFFFF0h
3	1...2	67108860 = 3FFFFFFCh
15	1...4	33554428 = 1FFFFFFCh
255	1...8	16777212 = 0FFFFFFCh

Alternativ wird bei den folgenden Kennwerten der jeweils letzte Messkanal als Zeitkanal genutzt:

chann els	Kanalnr.	Zeit- kanal	max. Anzahl der Messwerte pro Kanal für startadr=0:
131	1	2	67108860
143	1...3	4	33554428
127	1...7	8	16777212

startadr Startadresse ($0 \dots 268435452 = 2^{28} - 4$) im Modulspeicher. Die Adresse muss durch 4 teilbar sein. LONG

samples Anzahl der durchzuführenden bzw. speicherbaren Messungen pro Kanal; der Maximalwert für **samples** wird durch **channels** bestimmt. Die Anzahl muss durch 4 teilbar sein, bei **channels**=1 (1 Kanal) durch 8 teilbar. LONG

P2_Burst_Init

Betriebsart

pulses

legt die Periodendauer für eine Messreihe fest als Anzahl der Zeittakte; nur gültig in Verbindung mit zeitgesteuerter Taktrate (siehe **mode**):

Periodendauer = **pulses** * 20ns.

Der Wertebereich ist 1...65535; bei 8 Kanälen (**mode** = 255 oder 127) beginnt der Wertebereich mit 2.

Eine Periodendauer ist die Zeit vom Beginn einer Messung bis zum Beginn der nächsten Messung.

Nur bei den Modultypen AIn-F-x/16 gilt:

Periodendauer = **pulses** * 10ns.

Der kleinste Wert für **pulses** hängt vom Parameter **mode** bei **P2_Set_Average_Filter** ab:

mode	pulses _{min}
0	25
1	67
2	154
3	313
4	645
5	1333

mode

Bitmuster, das die Art der Burst-Messreihe angibt:

Bit-Nr.	03...31	02	01	00
Funktion	–	Art der Taktrate: Bit = 0: Zeitgesteuert (pulses). Bit = 1: Extern gesteuert (Event-Eingang).	Betriebsart der Burst-Messreihe: Bit = 0: Einfach Bit = 1: Kontinuierlich	–

Bemerkungen

Sie können mit **P2_Read_ADCF** den aktuellen Messwert auch eines solchen Kanals einlesen, der nicht abgespeichert wird, z.B. zum Prüfen einer Trigger-Bedingung.

In dem Zeitkanal wird bei jeder Burst-Messung der aktuelle Wert des moduleigenen Timers gespeichert. Damit können z.B. bei extern gesteuerter Taktrate die Messwerte zeitlich genau zugeordnet werden. Eine Zeiteinheit des 16 Bit-Timers entspricht 20ns.

Die Anzahl der speicherbaren Messwerte pro Kanal ist abhängig von der angegebenen Startadresse (und der Speichergröße des Moduls).

Bei einer einfachen Burst-Messreihe erfasst das Modul eine feste Anzahl **samples** von Messungen. Sobald alle Messwerte gewandelt sind, endet die Burst-Messreihe. Die Messdaten sollen mit **P2_Burst_Read_Unpacked...** gelesen werden.

Bei einer kontinuierlichen Burst-Messreihe wandelt das Modul dauernd Messwerte mit der festgelegten Periodendauer. Sie brechen die Messreihe mit **P2_Burst_Stop** ab und lesen mit **P2_Burst_CRead_Unpacked...** die Messwerte aus. Das Modul speichert die Messwerte im reservierten Speicher (siehe **P2_Burst_Init**) in einem Ringspeicher, d.h. die jüngsten Messwerte überschreiben die jeweils ältesten Daten.

Bei zeitgesteuerter Taktrate wird der Takt der Burst-Messungen vom Timer des Moduls gesteuert; in diesem Fall legt `pulses` die Taktrate fest.

Bei extern gesteuerter Taktrate löst jedes Event-Signal eine Burst-Messung aus; beachten Sie die Einstellung mit `P2_Event_Config`. Die maximale Taktrate beträgt 50MHz. Burst-Messreihen auf mehreren Modulen können mit `P2_Sync_Mode` synchronisiert werden.

Siehe auch

`P2_Burst_CRead_Unpacked1`, `P2_Burst_CRead_Unpacked2`, `P2_Burst_CRead_Unpacked4`, `P2_Burst_CRead_Unpacked8`, `P2_Set_Average_Filter`

`P2_Burst_Read_Index`, `P2_Burst_Read_Unpacked1`, `P2_Burst_Read_Unpacked2`, `P2_Burst_Read_Unpacked4`, `P2_Burst_Read_Unpacked8`

`P2_Burst_Start`, `P2_Burst_Status`, `P2_Burst_Stop`, `P2_Read_ADC`, `P2_Sync_Mode`

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4
```

```
Dim Data_1[1000] As Long
Dim pattern As Long
```

Init:

```
Rem Kont. Burst-Messreihe für Kanal 1 einrichten mit 20ns
Rem Periodendauer, 2^26 Daten speichern ab Adresse 0.
P2_Burst_Init (module,1,0,67108864,1,010b)
Rem Burst-Messreihe starten
pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
P2_Burst_Start(pattern)
Processdelay=10000000
```

Event:

```
Rem Die letzten 1000 Messwerte des Kanals (langsam) lesen und in
Rem Data_1 ablegen
P2_Burst_CRead_Unpacked1 (module,1000,Data_1,1,1)
```

Taktrate

P2_Burst_Read_Index

P2_Burst_Read_Index gibt die Adresse im Modulspeicher zurück, an der zuletzt Messwerte abgelegt wurden.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_Burst_Read_Index(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Adresse (0...268435452 = $2^{28} - 4$) im Modulspeicher. Die Adresse ist durch 4 teilbar.	LONG

Bemerkungen

P2_Burst_Read_Index ist ein elementarer Befehl, der in Verbindung mit **P2_Burst_Read** spezielle Lösungen ermöglicht, dafür aber auch besondere Sorgfalt und Kenntnisse bei der Programmierung voraussetzt. Die einfachere Alternative sind die Befehle **P2_Burst_Read_Unpacked...** oder **P2_Burst_CRead_Unpacked...**.

Aus der zurückgegebenen Adresse **ret_val** kann die Anzahl **n** der gespeicherten Messdaten berechnet werden. Startadresse und Kanalzahl werden mit **P2_Burst_Init** festgelegt:

$$n = (\text{ret_val} - \text{startadr}) \cdot \frac{2}{\text{Kanalzahl}}$$

Der Modulspeicher wird immer in 4er-Schritten (4 mal 32 Bit) adressiert. Nach den Befehlen **P2_Burst_Init** und **P2_Burst_Reset** steht der Adresszeiger auf der letztmöglichen Adresse des reservierten Modulspeichers, also auf **startadr + samples * (Kanalzahl) - 4**.

Von welchen Kanälen die zuletzt im Speicher abgelegten Messwerte stammen, hängt vom Messmodus ab. Näheres über die Zuordnung siehe **P2_Burst_Read**.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read](#)

[P2_Burst_Read_Unpacked1](#), [P2_Burst_Read_Unpacked2](#), [P2_Burst_Read_Unpacked4](#), [P2_Burst_Read_Unpacked8](#), [P2_Burst_Reset](#), [P2_Burst_Start](#), [P2_Burst_Status](#), [P2_Read_ADC](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

Beispiel

```
#Include ADWINPRO_ALL.Inc

#Define module 4
#Define samples 500000
#Define channels 4
#Define frq_Hz 5000
#Define mem_idx Par_1
#Define count Par_2
#Define overflow Par_3

Dim Data_1[samples], Data_2[samples] As Long
Dim Data_3[samples], Data_4[samples] As Long
Dim i, prev_mem_idx, start_idx As Long

LowInit:
  For i = 1 To samples
    Data_1[i] = 0 : Data_2[i] = 0 : Data_3[i] = 0 : Data_4[i] = 0
  Next i

Init:
  Processdelay = 300000000 / frq_Hz
  P2_Set_LED(module, 1) 'LED einschalten
  Rem Kont. Burst-Messreihe, Kanäle 1...4, 100ns Periodendauer
  P2_Burst_Init(module, 15, 0, samples, 5, 2)
  P2_Burst_Start(Shift_Left(1, module - 1))
  start_idx = 1
  prev_mem_idx = 0
  overflow = 0

Event:
  Rem Aktuelle Speicheradresse
  mem_idx = P2_Burst_Read_Index(module)
  Rem Anzahl neuer Messwerte/Kanal seit dem letzten Zyklus
  count = (mem_idx - prev_mem_idx) * 2 / channels

  If (count > 0) Then
    Rem Messwerte aus dem F8/14 Modul auslesen
    P2_Burst_Read_Unpacked4(module, count, prev_mem_idx,
      Data_1, Data_2, Data_3, Data_4, start_idx, 0)
    Rem Start-Index für den nächsten Zyklus
    start_idx = start_idx + count
    Rem Index im F8/14 Modul merken
    prev_mem_idx = mem_idx
  Endif

  If (count < 0) Then
    Rem Anzahl der Messwerte bis zum Data Ende
    count = samples - prev_mem_idx * 2 / channels
    Rem Messwerte aus dem F8/14 Modul auslesen
    P2_Burst_Read_Unpacked4(module, count, prev_mem_idx,
      Data_1, Data_2, Data_3, Data_4, start_idx, 0)
    Rem Start-Index im Data für den nächsten Zyklus
    start_idx = 1
    Rem Index im F8/14 Modul für den nächsten Zyklus
    prev_mem_idx = 0
    Inc(overflow) 'Überlaufzähler erhöhen
  Endif

Finish:
  P2_Set_LED(module, 0) 'LED ausschalten
```

P2_Burst_Read

P2_Burst_Read kopiert 32 Bit-Werte aus dem Speicher des angegebenen Moduls in ein bestimmtes Feld.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_Read(module, count, startadr, array[],
              array_idx, flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu insgesamt übertragenden 32-Bit-Werte. Die Anzahl muss durch 4 teilbar sein.	LONG
startadr	Startadresse (0...268435452 = $2^{28} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse muss durch 4 teilbar sein.	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
flowrate	Auswertung nur für niederpriori Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

P2_Burst_Read ist ein elementarer Befehl, der in Verbindung mit **P2_Burst_Read_Index** spezielle Lösungen ermöglicht, dafür aber auch besondere Sorgfalt und Kenntnisse bei der Programmierung voraussetzt. Die einfachere Alternative sind die Befehle **P2_Burst_Read_Unpacked...** oder **P2_Burst_CRead_Unpacked...**.

P2_Burst_Read kopiert die 32 Bit-Werte aus dem Speicher, ohne sie zu verändern; ein 32 Bit-Wert enthält 2 Messwerte zu 16 Bit. Welchen Kanälen die Messwerte zugeordnet sind, hängt von der Kanalanzahl ab (siehe **P2_Burst_Init**, Parameter **channels**). Die folgende Übersicht zeigt, wie die 16 Bit-Messwerte M den Kanalnummern K zugeordnet sind:

Adresse	Bits 31:16	Bits 15:00
startadr	K1 / M2	K1 / M1
startadr+1	K1 / M4	K1 / M3
startadr+2	K1 / M6	K1 / M5
...
1 Kanal		

Adresse	Bits 31:16	Bits 15:00
startadr	K2 / M1	K1 / M1
startadr+1	K2 / M2	K1 / M2
startadr+2	K2 / M3	K1 / M3
...
2 Kanäle		

Adresse	Bits 31:16	Bits 15:00
startadr	K2 / M1	K1 / M1
startadr+1	K4 / M1	K3 / M1
startadr+2	K2 / M2	K1 / M2
startadr+3	K4 / M2	K3 / M2
startadr+4	K2 / M3	K1 / M3
...
4 Kanäle		

Adresse	Bits 31:16	Bits 15:00
startadr	K2 / M1	K1 / M1
startadr+1	K4 / M1	K3 / M1
startadr+2	K6 / M1	K5 / M1
startadr+3	K7 / M1	K7 / M1
startadr+4	K2 / M2	K1 / M2
...
8 Kanäle		

In hochprioren Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter [flowrate](#) muss trotzdem angegeben werden.

Je höher Sie – bei einem niederprioren Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Index](#)

[P2_Burst_Read_Unpacked1](#), [P2_Burst_Read_Unpacked2](#), [P2_Burst_Read_Unpacked4](#), [P2_Burst_Read_Unpacked8](#), [P2_Burst_Start](#), [P2_Burst_Status](#), [P2_Burst_Stop](#), [P2_Read_ADC](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

Beispiel

siehe [P2_Burst_Read_Index](#)



P2_Burst_Read_Unpacked1

P2_Burst_Read_Unpacked1 kopiert die Messwerte eines Kanals aus dem Speicher des angegebenen Moduls in ein Feld.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_Burst_Read_Unpacked1(module, count, startadr,  
                        array[], array_idx, flowrate)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>count</code>	Anzahl der zu übertragenden Messwerte. Die Anzahl muss durch 8 teilbar sein.	LONG
<code>startadr</code>	Startadresse (0...268435452 = $2^{28} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse muss durch 4 teilbar sein.	LONG
<code>array[]</code>	Ziel-Feld, in das die Messwerte übertragen werden. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
<code>array_idx</code>	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriori Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine Burst-Messreihe mit einem einzigen Kanal eingerichtet wurde (siehe **P2_Burst_Init**, Parameter `channels`).

Die Anweisung legt die Messwerte einzeln nacheinander in den Elementen des Zielfelds ab.

In hochpriori Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter `flowrate` muss trotzdem angegeben werden.

Je höher Sie – bei einem niederpriori Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Unpacked2](#), [P2_Burst_Read_Unpacked4](#), [P2_Burst_Read_Unpacked8](#), [P2_Burst_Reset](#), [P2_Burst_Start](#), [P2_Burst_Status](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E



Beispiel

Siehe auch Beispiele für [Kontinuierliche Messwertwandlung](#) auf [Seite 281](#).

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

Dim Data_1[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long

Init:
    Rem Einfache Burst-Messreihe für Kanal 1 einrichten mit 20ns
    Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
    P2_Burst_Init (module,1,0,1000,1,0)
    Rem Burst-Messreihe starten
    pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
    P2_Burst_Start(pattern)
    Processdelay=10000000
    state=0 'Status: Burst-Messreihe läuft

Event:
    Rem Anzahl der restlichen Messwerte holen
    rest = P2_Burst_Status(module)
    Rem Alle Messwerte liegen vor: Status ändern
    If (rest=0) Then state=1
    If (state=1) Then
        Rem Alle Messwerte liegen vor: 1000 Messwerte (schnell)
        Rem abholen und in Data_1 ablegen
        P2_Burst_Read_Unpacked1(module,1000,0,Data_1,1,3)
        Rem Nächste Burst-Messreihe starten
        state=0
        P2_Burst_Reset(pattern)
        P2_Burst_Start(pattern)
    EndIf
```

P2_Burst_Read_Unpacked2

P2_Burst_Read_Unpacked2 kopiert die Messwerte von 2 Kanälen aus dem Speicher des angegebenen Moduls in 2 Felder.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_Burst_Read_Unpacked2(module, count, startadr,  
    array1[], array2[], array_idx, flowrate)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>count</code>	Anzahl der zu übertragenden Messwerte je Kanal. Die Anzahl muss durch 4 teilbar sein.	LONG
<code>startadr</code>	Startadresse ($0 \dots 268435452 = 2^{28} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse muss durch 4 teilbar sein.	LONG
<code>arrayx[]</code>	Ziel-Felder für die Messwerte der Kanäle 1 und 2. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
<code>array_idx</code>	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine Burst-Messreihe mit 2 Kanälen eingerichtet wurde (siehe **P2_Burst_Init**, Parameter `channels`).

Die Anweisung legt die Messwerte nacheinander in den Elementen der Zielfelder ab: Kanal 1 in `array1`, Kanal 2 in `array2`.

In hochprioren Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter `flowrate` muss trotzdem angegeben werden.

Je höher Sie – bei einem niederprioren Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Unpacked1](#), [P2_Burst_Read_Unpacked4](#), [P2_Burst_Read_Unpacked8](#), [P2_Burst_Reset](#), [P2_Burst_Start](#), [P2_Burst_Status](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E



Beispiel

Siehe auch Beispiele für [Kontinuierliche Messwertwandlung](#) auf [Seite 281](#).

```
#Include ADwinPro_All.Inc
```

```
#Define module 1
```

```
Dim Data_1[1000], Data_2[1000] As Long
```

```
Dim state As Long
```

```
Dim rest As Long
```

```
Dim pattern As Long
```

Init:

```
Rem Einfache Burst-Messreihe für Kanäle 1+2 einrichten mit 40ns
```

```
Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
```

```
P2_Burst_Init (module,3,0,1000,2,0)
```

```
Rem Burst-Messreihe starten
```

```
pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
```

```
P2_Burst_Start(pattern)
```

```
Processdelay=10000000
```

```
state=0
```

Event:

```
Rem Anzahl der restlichen Messwerte holen
```

```
rest = P2_Burst_Status(module)
```

```
Rem Alle Messwerte liegen vor: Status ändern
```

```
If (rest = 0) Then state=1
```

```
If (state = 1) Then
```

```
Rem Alle Messwerte liegen vor: Von jedem Kanal 1000 Messwerte
```

```
Rem (schnell) abholen und in Data_1 ablegen
```

```
P2_Burst_Read_Unpacked2(module,1000,0,Data_1,Data_2,1,3)
```

```
Rem Nächste Burst-Messreihe starten
```

```
state=0
```

```
P2_Burst_Reset(pattern)
```

```
P2_Burst_Start(pattern)
```

```
EndIf
```

P2_Burst_Read_Unpacked4

P2_Burst_Read_Unpacked4 kopiert die Messwerte von 4 Kanälen aus dem Speicher des angegebenen Moduls in 4 Felder.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_Read_Unpacked4(module, count, startadr,
    array1[], array2[], array3[], array4[],
    array_idx, flowrate)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>count</code>	Anzahl der zu übertragenden Messwerte je Kanal. Die Anzahl muss durch 2 teilbar sein.	LONG
<code>startadr</code>	Startadresse ($0 \dots 268435452 = 2^{28} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse muss durch 4 teilbar sein.	LONG
<code>arrayx[]</code>	Ziel-Felder für die Messwerte der Kanäle 1...4. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
<code>array_idx</code>	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriori Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine Burst-Messreihe mit 4 Kanälen eingerichtet wurde (siehe **P2_Burst_Init**, Parameter `channels`).

Die Anweisung legt die Messwerte nacheinander in den Elementen der Zielfelder ab: Kanal 1 in `array1`, Kanal 2 in `array2` etc.

In hochpriori Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter `flowrate` muss trotzdem angegeben werden.

Je höher Sie – bei einem niederpriori Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Unpacked1](#), [P2_Burst_Read_Unpacked2](#), [P2_Burst_Read_Unpacked8](#), [P2_Burst_Reset](#), [P2_Burst_Start](#), [P2_Burst_Status](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E



Beispiel

Siehe auch Beispiele für [Kontinuierliche Messwertwandlung](#) auf [Seite 281](#).

```
#Include ADwinPro_All.Inc
```

```
#Define module 4
```

```
Dim Data_1[1000], Data_2[1000] As Long
```

```
Dim Data_3[1000], Data_4[1000] As Long
```

```
Dim state As Long
```

```
Dim rest As Long
```

```
Dim pattern As Long
```

Init:

```
Rem Einfache Burst-Messreihe für Kanäle 1...4 einrichten mit 40ns
```

```
Rem Periodendauer, 3000 Messwerte je Kanal ab Adr. 0 speichern.
```

```
P2_Burst_Init (module,15,0,3000,2,0)
```

```
Rem Burst-Messreihe starten
```

```
pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
```

```
P2_Burst_Start(pattern)
```

```
Processdelay=10000000
```

```
state=0
```

Event:

```
Rem Anzahl der restlichen Messwerte holen
```

```
rest=P2_Burst_Status(module)
```

```
Rem Alle Messwerte liegen vor: Status ändern
```

```
If (rest=0) Then state=1
```

```
If (state=1) Then
```

```
Rem Alle Messwerte liegen vor: Von jedem Kanal 3000 Messwerte
```

```
Rem (schnell) abholen und in Data_1 bis Data_4 ablegen
```

```
P2_Burst_Read_Unpacked4(module,1000,0,Data_1,Data_2,Data_3,
```

```
    Data_4,1,3)
```

```
Rem Nächste Burst-Messreihe starten
```

```
state=0
```

```
P2_Burst_Reset(pattern)
```

```
P2_Burst_Start(pattern)
```

```
EndIf
```

P2_Burst_Read_Unpacked8

P2_Burst_Read_Unpacked8 kopiert die Messwerte von 8 Kanälen aus dem Speicher des angegebenen Moduls in 8 Felder.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_Read_Unpacked8(module, count, startadr,
    array1[], array2[], array3[], array4[], array5[],
    array6[], array7[], array8[], array_idx,
    flowrate)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>count</code>	Anzahl der zu übertragenden Messwerte je Kanal.	LONG
<code>startadr</code>	Startadresse ($0 \dots 268435452 = 2^{28} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse muss durch 4 teilbar sein.	LONG
<code>arrayx[]</code>	Ziel-Felder für die Messwerte der Kanäle 1...8. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
<code>array_idx</code>	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriori Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine Burst-Messreihe mit 8 Kanälen eingerichtet wurde (siehe **P2_Burst_Init**, Parameter `channels`).

Die Anweisung legt die Messwerte nacheinander in den Elementen der Zielfelder ab: Kanal 1 in `array1`, Kanal 2 in `array2` etc.

In hochpriori Prozessen wird automatisch der maximale Datendurchsatz verwendet. Der Parameter `flowrate` muss trotzdem angegeben werden.

Je höher Sie – bei einem niederpriori Prozess – den Datendurchsatz wählen, umso eher kann es vorkommen, dass ein Prozess mit höherer Priorität auf seine Bearbeitung warten muss.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Unpacked1](#), [P2_Burst_Read_Unpacked2](#), [P2_Burst_Read_Unpacked4](#), [P2_Burst_Reset](#), [P2_Burst_Start](#), [P2_Burst_Status](#)

Gültig für

Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E



Beispiel

Siehe auch Beispiele für [Kontinuierliche Messwertwandlung](#) auf Seite 281.

```
#Include ADwinPro_All.Inc
#Define module 4
```

```
Dim Data_1[1000] As Long
Dim Data_2[1000] As Long
Dim Data_3[1000] As Long
Dim Data_4[1000] As Long
Dim Data_5[1000] As Long
Dim Data_6[1000] As Long
Dim Data_7[1000] As Long
Dim Data_8[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long
```

Init:

```
Rem Einfache Burst-Messreihe für Kanal 1 einrichten mit 20ns
Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
P2_Burst_Init (module,255,0,1000,2,0)
Rem Burst-Messreihe starten
pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
P2_Burst_Start(pattern)
Processdelay=10000000
state=0
```

Event:

```
Rem Anzahl der restlichen Messwerte holen
rest=P2_Burst_Status(module)
Rem Alle Messwerte liegen vor: Status ändern
If (rest=0) Then state=1
If (state=1) Then
    Rem Alle Messwerte liegen vor: 1000 Messwerte (schnell)
    Rem abholen und in Data_1 ablegen
    P2_Burst_Read_Unpacked8(module,1000,0,Data_1,Data_2,Data_3,
        Data_4,Data_5,Data_6,Data_7,Data_8,1,3)
    Rem Nächste Burst-Messreihe starten
    state=0
    P2_Burst_Reset(pattern)
    P2_Burst_Start(pattern)
EndIf
```

P2_Burst_Reset

P2_Burst_Reset setzt den Datenzeiger der Burst-Messreihe auf allen angegebenen Modulen zurück.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_Reset(module_pattern)
```

Parameter

module_ Bitmuster zum Ansprechen der Moduladressen: LONG
pattern Bit = 0: Modul ignorieren.
 Bit = 1: Modul ansprechen.

Bit-Nr.	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen

Der Befehl wirkt auf alle eingestellten Module gleichzeitig. Wenn der Befehl für das angesprochene Modul ungültig ist, kann das unvorhergesehene Folgen haben.

Der Datenzeiger gibt an, an welcher Adresse im Modulspeicher die letzten Messwerte abgelegt wurden. Durch das Rücksetzen werden die nächsten Messwerte ab der mit **P2_Burst_Init** eingestellten Startadresse gespeichert. Der Datenzeiger kann mit **P2_Burst_Read_Index** gelesen werden.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Index](#), [P2_Burst_CRead_Unpacked1](#), [P2_Burst_CRead_Unpacked2](#), [P2_Burst_CRead_Unpacked4](#), [P2_Burst_CRead_Unpacked8](#), [P2_Burst_Status](#), [P2_Burst_Stop](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E



Beispiel

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

Dim Data_1[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long

Init:
  Rem Einfache Burst-Messreihe für Kanal 1 einrichten mit 20ns
  Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
  P2_Burst_Init (module,1,0,1000,1,0)
  Rem Burst-Messreihe starten
  pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
  P2_Burst_Start(pattern)
  Processdelay=10000000
  state=0 'Status: Burst-Messreihe läuft

Event:
  Rem Anzahl der restlichen Messwerte holen
  rest = P2_Burst_Status(module)
  Rem Alle Messwerte liegen vor: Status ändern
  If (rest=0) Then state=1
  If (state=1) Then
    Rem Alle Messwerte liegen vor: 1000 Messwerte (schnell)
    Rem abholen und in Data_1 ablegen
    P2_Burst_Read_Unpacked1(module,1000,0,Data_1,1,3)
    Rem Nächste Burst-Messreihe starten
    state=0
    P2_Burst_Reset(pattern)
    P2_Burst_Start(pattern)
  EndIf
```

P2_Burst_Start

P2_Burst_Start startet eine Burst-Messreihe auf allen angegebenen Modulen gleichzeitig.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_Start(module_pattern)
```

Parameter

module_ Bitmuster zum Ansprechen der Moduladressen: LONG
pattern Bit = 0: Modul ignorieren.
 Bit = 1: Modul ansprechen.

Bit-Nr.	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen



Der Befehl wirkt auf alle eingestellten Module gleichzeitig. Wenn der Befehl für das angesprochene Modul ungültig ist, kann das unvorhergesehene Folgen haben.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Index](#), [P2_Burst_CRead_Unpacked1](#), [P2_Burst_CRead_Unpacked2](#), [P2_Burst_CRead_Unpacked4](#), [P2_Burst_CRead_Unpacked8](#), [P2_Burst_Reset](#), [P2_Burst_Status](#), [P2_Burst_Stop](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4
```

```
Dim Data_1[1000] As Long
Dim pattern As Long
```

Init:

```
Rem Kont. Burst-Messreihe für Kanal 1 einrichten mit 20ns
Rem Periodendauer, 2^26 Daten speichern ab Adresse 0.
P2_Burst_Init (module,1,0,67108864,1,010b)
Rem Burst-Messreihe starten
pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
P2_Burst_Start(pattern)
Processdelay=10000000
```

Event:

```
Rem Die letzten 1000 Messwerte des Kanals (langsam) lesen und in
Rem Data_1 ablegen
P2_Burst_CRead_Unpacked1(module,1000,Data_1,1,1)
```

P2_Burst_Status ermittelt die Anzahl der noch durchzuführenden Burst-Messungen auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_Burst_Status(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Anzahl der noch auszuführenden Messungen.	LONG

Bemerkungen

Die Anweisung soll nur bei einer einfachen Burst-Messreihe (siehe **P2_Burst_Init**) verwendet werden.

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

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Index](#), [P2_Burst_Read_Unpacked1](#), [P2_Burst_Read_Unpacked2](#), [P2_Burst_Read_Unpacked4](#), [P2_Burst_Read_Unpacked8](#), [P2_Burst_Reset](#), [P2_Burst_Start](#), [P2_Read_ADC](#), [P2_Burst_Stop](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

P2_Burst_Status

Beispiel

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

Dim Data_1[1000] As Long
Dim state As Long
Dim rest As Long
Dim pattern As Long

Init:
    Rem Einfache Burst-Messreihe für Kanal 1 einrichten mit 20ns
    Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
    P2_Burst_Init (module,1,0,1000,1,0)
    Rem Burst-Messreihe starten
    pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
    P2_Burst_Start(pattern)
    Processdelay=10000000
    Rem Status: Burst-Messreihe läuft
    state=0

Event:
    Rem Anzahl der restlichen Messwerte holen
    rest=P2_Burst_Status(module)
    Rem Alle Messwerte liegen vor: Status ändern
    If (rest=0) Then state=1
    If (state=1) Then
        Rem Alle Messwerte liegen vor: 1000 Messwerte (schnell)
        Rem abholen und in Data_1 ablegen
        P2_Burst_Read_Unpacked1(module,1000,0,Data_1,1,3)
        Rem Nächste Burst-Messreihe starten
        state=0
        P2_Burst_Reset(pattern)
        P2_Burst_Start(pattern)
    EndIf
```


P2_Burst_Stop unterbricht eine laufende Burst-Messreihe auf allen angegebenen Modulen gleichzeitig.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_Stop(module_pattern)
```

Parameter

module_ Bitmuster zum Ansprechen der Moduladressen: LONG
pattern Bit = 0: Modul ignorieren.
 Bit = 1: Modul ansprechen.

Bit-Nr.	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen

Ein interner Datenzeiger gibt an, an welcher Adresse im Modulspeicher die letzten Messwerte abgelegt wurden. Der Datenzeiger kann mit **P2_Burst_Read_Index** gelesen werden.

Eine unterbrochene Burst-Messreihe kann mit **Burst_Start** wieder aufgenommen werden.

Mit **P2_Burst_Reset** wird der Datenzeiger auf die mit **P2_Burst_Init** eingestellte Startadresse zurückgesetzt. Bisher gespeicherte Messwerte werden dadurch überschrieben.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_CRead_Unpacked1](#), [P2_Burst_Read_Unpacked1](#), [P2_Burst_Read](#), [P2_Burst_Status](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

P2_Burst_Stop

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4

Dim Data_1[1000] As Long
Dim i As Long
Dim pattern As Long

Init:
    Rem Rinfache Burst-Messreihe für Kanal 1 einrichten,
    Rem zeitgesteuert mit 20ns Periodendauer, 2^26 Daten speichern
    Rem ab Adresse 0.
    P2_Burst_Init (module,1,0,67108864,1,0)
    Rem Burst-Messreihe starten
    pattern = Shift_Left(1,module-1) 'nur ein Modul ansprechen
    P2_Burst_Start(pattern)
    Processdelay=10000000

Event:
    Rem Die letzten 1000 Messwerte des Kanals (langsam) lesen und in
    Rem Data_1 ablegen
    P2_Burst_CRead_Unpacked1(module,1000,Data_1,1,1)
    Rem Burst-Messreihe unterbrechen, wenn Grenzwert überschritten
    For i = 1 To 1000
        If (Data_1[i]>5) Then
            P2_Burst_Stop(pattern)
        EndIf
    Next
```

P2_Set_Average_Filter legt fest, ob und aus wievielen Messwerten das angegebene Modul einen Mittelwert berechnet.

Syntax

```
#Include ADwinPro_All.Inc

P2_Set_Average_Filter(module, mode)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
mode	Filtermodus (0...5): 0: Filter aus = Original-Messwerte. 1: Mittelwert bilden aus $2^1 = 2$ Werten. 2: Mittelwert bilden aus $2^2 = 4$ Werten. 3: Mittelwert bilden aus $2^3 = 8$ Werten. 4: Mittelwert bilden aus $2^4 = 16$ Werten. 5: Mittelwert bilden aus $2^5 = 32$ Werten.	LONG

Bemerkungen

Der Filtermodus gilt gleichermaßen für Einzelwert-Messungen und Burst-Messungen.

Der Mittelwert wird immer aus den zuletzt gewandelten Messwerten berechnet. Durch deren kontinuierliche Wandlung ergibt sich bei den Modulen Aln-F-x/14 die Bildung eines gleitenden Mittelwerts.

Siehe auch

[P2_Burst_Init](#), [P2_Burst_Read_Unpacked1](#), [P2_Burst_CRead_Unpacked1](#), [P2_Read_ADC](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
Rem Mittelwert aus den 2 zuletzt gewandelten Messwerten bilden
P2_Set_Average_Filter(1,1)
```

P2_Set_Average_Filter

P2_ADCF

P2_ADCF führt eine komplette Messung auf einem Fast-ADC durch. Der Rückgabewert hat 16 Bit Auflösung.

Syntax

```
#Include ADwinPro_All.Inc

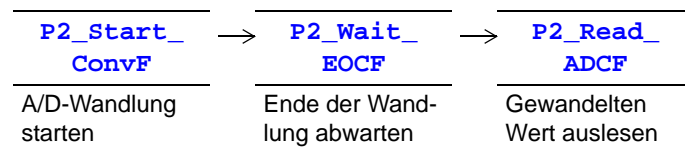
ret_val = P2_ADCF(module, input_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
input_no	Nummer (1...4 oder 1...8) des analogen Eingangs.	LONG
ret_val	Wandlungsergebnis (0...65535).	LONG

Bemerkungen

Die Anweisung **P2_ADCF** ist eine Zusammenstellung von aufeinander folgenden Funktionen:



Siehe auch

[P2_ADCF24](#), [P2_ADCF_Mode](#), [P2_ADCF_Read_Limit](#), [P2_ADCF_Set_Limit](#), [P2_Read_ADCF](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Event:

```
Rem 16Bit-Wert am analogen Eingang 4 messen
value = P2_ADCF(1, 4)
```

P2_ADCF24 führt eine komplette Messung auf einem Fast-ADC durch. Der Rückgabewert hat 24 Bit Auflösung.

Syntax

```
#Include ADwinPro_All.Inc

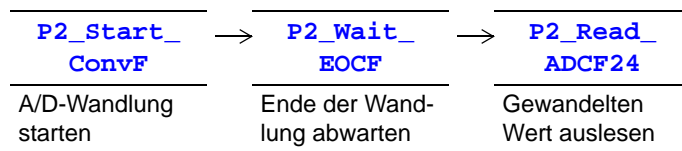
ret_val = P2_ADCF24(module, input_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
input_no	Nummer (1...4 oder 1...8) des analogen Eingangs.	LONG
ret_val	Wandlungsergebnis ($0 \dots 16777215 = 2^{24}-1$).	LONG

Bemerkungen

Die Anweisung **P2_ADCF24** ist eine Zusammenstellung von aufeinander folgenden Funktionen:



Siehe auch

[P2_ADCF](#), [P2_ADCF_Mode](#), [P2_ADCF_Read_Limit](#), [P2_ADCF_Set_Limit](#), [P2_Read_ADCF24](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/18 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Event:

```
Rem 24Bit-Wert am analogen Eingang 4 messen
value = P2_ADCF24(1, 4)
```

P2_ADCF24

P2_ADCF_Mode

P2_ADCF_Mode stellt den Arbeitsmodus für alle Kanäle der angegebenen Module ein.

Syntax

```
#Include ADwinPro_All.Inc

P2_ADCF_Mode(module_pattern, mode)
```

Parameter

module_pattern Bitmuster zum Auswählen der Moduladressen: LONG
 Bit = 0: Moduladresse ignorieren.
 Bit = 1: Moduladresse ansprechen.

Bitmuster	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

mode Arbeitsmodus des Moduls: LONG

mode	Modus
0	Standard-Modus (Default)
1	Timer-Modus
3	Timer-Modus mit Multiplex-Option ^a
4	Event-Modus
6	Event-Modus mit Multiplex-Option ^a

^a. nicht verfügbar für Aln-F-4/16 und Aln-F-8/16

Bemerkungen

Der Befehl wirkt auf alle gewählten Module gleichzeitig. Wenn der Befehl für ein angesprochenes Modul ungültig ist, kann das unvorhergesehene Folgen haben.

Im Standard-Modus startet das Prozessormodul jede Wandlung für jeden Kanal einzeln, z.B. mit dem Befehl **P2_Start_Conv**.

Im Timer-Modus wandelt das Modul eigenständig und zyklisch alle Kanäle. Dadurch wird das Prozessormodul entlastet, das im Prozess nur noch die gewandelten Werte liest und verarbeitet. Die Wandlung auf dem Modul geschieht synchron zum **Processdelay** des Prozesses.

Der Timer-Modus kann nur im Abschnitt **Init**: eingeschaltet werden; die Anweisung sollte möglichst am Ende des Abschnitts stehen.

Das Prozessormodul sollte im Abschnitt **Event**: den gewandelten Wert möglichst sofort auslesen.

Im Detail geschieht Folgendes:

Die Anweisung **P2_ADCF_Mode** übergibt das eingestellte **Processdelay** des Prozesses an das Modul. Eine bestimmte Zeit später beginnt das Modul eigenständig mit der Wandlung auf allen Kanälen. Der moduleigene Timer startet die Wandlung zyklisch und – durch das übergebene **Processdelay** – synchron zum Prozesstakt; die maximale Wandlungsrate ist in der Hardware-Modulbeschreibung angegeben.

Im Timer-Modus wird das Wandlungsende regelmäßig gerade dann erreicht, wenn das Prozessormodul seinen Prozesszyklus beginnt. Wenn der Prozessor den Messwert – z.B. weil der Prozesszyklus verzögert startet oder weil der Lese-Befehl nicht am Zyklusbeginn steht – erst später liest, kann die nächste Wandlung bereits anlaufen oder gar abgeschlossen sein. Auf diese Weise kann das Prozessormodul einzelne Messwerte überspringen oder mehrfach lesen.



Standard-Modus

Timer-Modus

Der Timer-Modus sollte möglichst nur in Kombination mit einem einzigen hochprioritären Prozess genutzt werden.

Im Timer-Modus mit Multiplex-Option wandelt das Modul doppelt so schnell wie im einfachen Timer-Modus, jedoch nur mit der Hälfte der Kanäle. Das Prozessormodul liest und verarbeitet in jedem Prozesszyklus ein Messwertpaar.

Die Messwerte können nur paarweise gelesen werden, also mit den Befehlen **P2_Read_ADCF32**, **P2_Read_ADCF4_Packed**, **P2_Read_ADCF8_Packed**. Der ältere der beiden Werte steht im oberen Wort, der neuere Wert im unteren Wort.

Jedes Messsignal muss an einem Eingangspaar angeschlossen sein: 1+2, 3+4, 5+6, 7+8; andere Paarkombinationen sind nicht möglich.

Das **Processdelay** des Prozesses muss geradzahlig sein, damit die Wandlung synchron getaktet wird.

Im Event-Modus startet jedes Signal am Event-Eingang des Moduls eine Wandlung auf allen Kanälen.

Wenn der Event-Eingang am Modul mit **P2_Event_Enable** freigegeben ist, sendet das Modul ein Event-Signal an das Prozessormodul. Das Event-Signal startet den (extern gesteuerten) Prozesszyklus gerade dann, wenn das Wandlungsende erreicht ist. Das Prozessormodul ist dadurch entlastet, es liest nur noch die gewandelten Werte ein und verarbeitet sie.

Im Event-Modus mit Multiplex-Option kann das Modul doppelt so schnell wandeln wie im einfachen Event-Modus, jedoch nur mit der Hälfte der Kanäle.

Jedes Messsignal muss an einem Eingangspaar angeschlossen sein: 1+2, 3+4, 5+6, 7+8; andere Paarkombinationen sind nicht möglich.

Wenn der Event-Eingang am Modul mit **P2_Event_Enable** freigegeben ist, sendet das Modul zum Ende jeder zweiten Wandlung ein Event-Signal an das Prozessormodul.

Das Prozessormodul muss in jedem Prozesszyklus ein Messwertpaar lesen, gut geeignet sind die Befehle **P2_Read_ADCF32**, **P2_Read_ADCF4_Packed**, **P2_Read_ADCF8_Packed**. Der ältere der beiden Werte steht im oberen Wort, der neuere Wert im unteren Wort.

Der Event-Eingang ist nur bei Modulen mit DSub-Stecker vorhanden.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_ADCF_Read_Limit](#), [P2_ADCF_Set_Limit](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#), [P2_Read_ADCF](#), [P2_Read_ADCF32](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF4_24B](#), [P2_Read_ADCF8_24B](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E



Timer-Modus mit Multiplex-Option

Event-Modus

Event-Modus mit Multiplex-Option



Beispiel

```
#Include ADwinPro_All.Inc
Dim value[4] As Long

Init:
    Rem ...
    P2_ADCF_Mode(1,1)           'Timer-Modus einschalten.
                                'Letzter Befehl im Abschnitt!

Event:
    P2_Read_ADCF4(1, value, 1) 'Werte der ADC 1-4 einlesen
    Rem Werte verarbeiten
```


P2_ADCF_Read_Limit liest die Flags für Grenzwertüber- und unterschreitungen auf allen F-ADC des angegebenen Moduls aus.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_ADCF_Read_Limit(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Bitmuster aus den Flags für Grenzwertüber- und unterschreitungen:	LONG

Bit Nr.	31:24	23	22	21	20	19	18	17	16
Überschreitung der Obergrenze									
F-ADC-Nr.	–	8	7	6	5	4	3	2	1
Bit Nr.	15:08	7	6	5	4	3	2	1	0
Unterschreitung der Untergrenze									
F-ADC-Nr.	–	8	7	6	5	4	3	2	1

Bemerkungen

Sie stellen die Grenzwerte mit **P2_ADCF_Set_Limit** ein.

Das Lesen der Flags setzt alle Flags auf Null zurück.

Wir empfehlen, im Abschnitt **Init:** die Flags einmal zu lesen, damit eventuelle vorherige Grenzwertüber- und unterschreitungen gelöscht sind. Dies ist bei einem extern gesteuerten Prozess besonders wichtig.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_ADCF_Mode](#), [P2_ADCF_Set_Limit](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Init:
P2_ADCF_Set_Limit(1, 2, 32768, 256) 'Grenzwerte Kanal 2 setzen
flags = P2_ADCF_Read_Limit(1) 'Flags lesen und rücksetzen

Event:
flags = P2_ADCF_Read_Limit(1) 'Flags lesen
If (flags And 10b = 10b) Then
    Rem Untergrenze ist unterschritten
    Rem ...
EndIf
If (flags And 2000h = 2000h) Then
    Rem Obergrenze ist überschritten
    Rem ...
EndIf
```

P2_ADCF_Read_Limit

P2_ADCF_Set_Limit

P2_ADCF_Set_Limit setzt den oberen und unteren Grenzwert für einen F-ADC des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_ADCF_Set_Limit(module, input_no, high, low)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
input_no	Nummer (1...4 oder 1...8) des analogen Eingangs.	LONG
high	Oberer Grenzwert (0...65535) des Kanals. Voreinstellung: 65535.	LONG
low	Unterer Grenzwert (0...65535) des Kanals. Voreinstellung: 0.	LONG

Bemerkungen

Dieser Befehl ist nur sinnvoll, wenn das Modul nicht im Standard-Arbeitsmodus arbeitet (siehe **P2_ADCF_Mode**).

Wenn ein Messwert den oberen Grenzwert überschreitet, wird für diesen Kanal ein Flag gesetzt, das mit **P2_ADCF_Read_Limit** gelesen und zurückgesetzt wird.

In gleicher Weise wird ein Flag für den Kanal gesetzt, wenn ein Messwert den unteren Grenzwert unterschreitet.

Grenzwertübertretungen können keine Event-Signale auslösen.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_ADCF_Mode](#), [P2_ADCF_Read_Limit](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Init:

```
P2_ADCF_Set_Limit(1, 2, 32768, 256) 'Grenzwerte Kanal 2 setzen  
flags = P2_ADCF_Read_Limit(1) 'Flags lesen und rücksetzen
```

Event:

```
flags = P2_ADCF_Read_Limit(1) 'Flags lesen  
If (flags And 10b = 10b) Then  
    Rem Untergrenze ist unterschritten  
    Rem ...  
EndIf  
If (flags And 20000h = 20000h) Then  
    Rem Obergrenze ist überschritten  
    Rem ...  
EndIf
```

P2_ADCF_Reset_Min_Max setzt die Minimal- und Maximalwerte bestimmter Kanäle auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

P2_ADCF_Reset_Min_Max(module, channel_pattern)
```

Parameter

module Eingestellte Moduladresse (1...15). LONG

channel_ Bitmuster zur Auswahl der Kanäle, auf denen die LONG

pattern Extremwerte zurückgesetzt werden.

Bit Nr.	15:08	7	6	5	4	3	2	1	0
F-ADC-Nr.	–	8	7	6	5	4	3	2	1

Bemerkungen

Die Maximalwerte werden zurückgesetzt auf Null, die Minimalwerte auf **0FFFFh**.

Siehe auch

[P2_ADCF_Read_Min_Max4](#), [P2_ADCF_Read_Min_Max8](#), [P2_ADCF_Read_Limit](#), [P2_ADCF_Set_Limit](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-8/16 Rev. E

Beispiel

siehe [P2_ADCF_Read_Min_Max8](#)

P2_ADCF_Reset_Min_Max

P2_ADCF_Read_Min_Max4

P2_ADCF_Read_Min_Max4 gibt die Minimal- und Maximalwerte von 4 F-ADC des angegebenen Moduls in einem Feld zurück.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_ADCF_Read_Min_Max4(module, array[], array_index)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Zielfeld, in dem die Extremwerte gespeichert werden. Das Feld muss mindestens <code>array_index + 7</code> Elemente haben.	ARRAY LONG
<code>array_index</code>	Index des Elements im Zielfeld, in dem der erste Extremwert gespeichert wird.	LONG

Bemerkungen

Die Extremwerte werden durch das Auslesen nicht zurückgesetzt. Verwenden Sie dafür den Befehl **P2_ADCF_Reset_Min_Max**.

Im Feld `array[]` liegen die Extremwerte in der folgenden Reihenfolge vor (mit `array_index = n`):

Feldelement	Wert, Kanal
<code>array[n]</code>	Min. Kanal 1
<code>array[n+1]</code>	Max. Kanal 1
<code>array[n+2]</code>	Min. Kanal 2
<code>array[n+3]</code>	Max. Kanal 2
<code>array[n+4]</code>	Min. Kanal 3
<code>array[n+5]</code>	Max. Kanal 3
<code>array[n+6]</code>	Min. Kanal 4
<code>array[n+7]</code>	Max. Kanal 4

Siehe auch

[P2_ADCF_Read_Min_Max8](#), [P2_ADCF_Reset_Min_Max](#), [P2_ADCF_Read_Limit](#), [P2_ADCF_Set_Limit](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-8/16 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
Dim Data_10[8] As Long
Dim i As Long

Init:
  P2_ADCF_Reset_Min_Max(1,1111b)'reset all 4 F-ADC

Event:
  Rem read high and low values of F-ADC 1...4
  P2_ADCF_Read_Min_Max4(1,Data_10,1)
  For i = 1 To 8 Step 2
    If (Data_10[i] < 2500) Then
      Rem minimum is below limit
      Rem ...
      P2_ADCF_Reset_Min_Max(1,1111b)'reset all 4 F-ADC
    EndIf

    If (Data_10[i+1] > 50000) Then
      Rem value is above limit
      Rem ...
    EndIf
  Next i
```

P2_ADCF_Read_Min_Max8

P2_ADCF_Read_Min_Max8 gibt die Minimal- und Maximalwerte von 8 F-ADC des angegebenen Moduls in einem Feld zurück.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_ADCF_Read_Min_Max8(module, array[], array_index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in dem die Extremwerte gespeichert werden. Das Feld muss mindestens array_index + 15 Elemente haben.	ARRAY LONG
array_index	Index des Elements im Zielfeld, in dem der erste Extremwert gespeichert wird.	LONG

Bemerkungen

Die Extremwerte werden durch das Auslesen nicht zurückgesetzt. Verwenden Sie dafür den Befehl **P2_ADCF_Reset_Min_Max**.

Im Feld **array[]** liegen die Extremwerte in der folgenden Reihenfolge vor (mit **array_index** = n):

Feldelement	Wert, Kanal
array[n]	Min. Kanal 1
array[n+1]	Max. Kanal 1
array[n+2]	Min. Kanal 2
array[n+3]	Max. Kanal 2
array[n+4]	Min. Kanal 3
array[n+5]	Max. Kanal 3
array[n+6]	Min. Kanal 4
array[n+7]	Max. Kanal 4

Feldelement	Wert, Kanal
array[n+8]	Min. Kanal 5
array[n+9]	Max. Kanal 5
array[n+10]	Min. Kanal 6
array[n+11]	Max. Kanal 6
array[n+12]	Min. Kanal 7
array[n+13]	Max. Kanal 7
array[n+14]	Min. Kanal 8
array[n+15]	Max. Kanal 8

Siehe auch

[P2_ADCF_Read_Min_Max4](#), [P2_ADCF_Reset_Min_Max](#), [P2_ADCF_Read_Limit](#), [P2_ADCF_Set_Limit](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-8/16 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
Dim Data_4[16] As Long
Dim i As Long

Init:
  P2_ADCF_Reset_Min_Max(1,11111111b)'reset all 8 F-ADC

Event:
  Rem read high and low values of F-ADC 1...8
  P2_ADCF_Read_Min_Max8(1,Data_4,1)
  For i = 1 To 16 Step 2
    If (Data_4[i] < 2500) Then
      Rem minimum is below limit
      Rem ...
      P2_ADCF_Reset_Min_Max(1,11111111b)'reset all 8 F-ADC
    EndIf

    If (Data_4[i+1] > 50000) Then
      Rem value is above limit
      Rem ...
    EndIf
  Next i
```

P2_Read_ADCF

P2_Read_ADCF liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus. Das Ergebnis hat 16 Bit Auflösung.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_Read_ADCF(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

Mit den Befehlen **P2_Read_ADCF4**, **P2_Read_ADCF8**, **P2_Read_ADCF4_Packed**, **P2_Read_ADCF8_Packed** können mehrere Ergebnisse sehr schnell ausgelesen werden.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#), [P2_ADCF_Mode](#), [P2_ADCF_Read_Limit](#), [P2_ADCF_Set_Limit](#), [P2_Read_ADCF32](#), [P2_Read_ADCF_SConv](#), [P2_Read_ADCF_SConv32](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Event:

```
Rem Start AD-Wandlung; nicht erforderlich für Aln-F-8/14  
P2_Start_ConvF(1,1)  
Rem Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14  
P2_Wait_EOCF(1,1)  
value1 = P2_Read_ADCF(1,1)'Wert vom ADC einlesen
```


P2_Read_ADCF24 liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus. Das Ergebnis hat 24 Bit Auflösung.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_Read_ADCF24(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Nummer des zu lesenden ADC (1...4 oder 1...8).	LONG
ret_val	Im F-ADC-Register enthaltener Messwert (0...16777215 = $2^{24}-1$).	LONG

Bemerkungen

Mit den Befehlen **Read_ADCF4_24B**, **Read_ADCF8_24B** können mehrere Ergebnisse sehr schnell ausgelesen werden.

Siehe auch

[P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#), [P2_ADCF_Mode](#), [P2_ADCF_Read_Limit](#), [P2_ADCF_Set_Limit](#), [P2_Read_ADCF_SConv24](#), [P2_Read_ADCF4_24B](#), [P2_Read_ADCF8_24B](#)

Gültig für

Aln-F-4/18 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Event:
P2_Start_ConvF(1,1)         'Start AD-Wandlung
P2_Wait_EOCF(1,1)           'Warten auf Wandlung-Ende
value1 = P2_Read_ADCF24(1,1) '24Bit-Wert vom ADC einlesen
```

P2_Read_ADCF24

P2_Read_ADCF4

P2_Read_ADCF4 liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus.

Syntax

```
#Include ADWINPRO_ALL.Inc  
  
P2_Read_ADCF4(module, array[], index)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY LONG FLOAT
<code>index</code>	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 **P2_Read_ADCF**.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Read_ADCF](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF_SConv](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADWINPRO_ALL.Inc  
Dim value[4] As Long 'Feld für Messwerte  
  
Init:  
    Rem Start AD-Wandlung Kanäle 1...4; nicht erforderl. für Aln-F-8/14  
    P2_Start_ConvF(1,0Fh)  
  
Event:  
    Rem Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14  
    P2_Wait_EOCF(1,0Fh)  
    P2_Read_ADCF4(1,value,1) 'Werte der ADC 1...4 lesen  
    Rem Neue AD-Wandlung starten; nicht erforderlich für Aln-F-8/14  
    P2_Start_ConvF(1,0Fh)
```

P2_Read_ADCF4_24B liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus. Die Ergebnisse haben eine Auflösung von 24 Bit.

Syntax

```
#Include ADWINPRO_ALL.Inc

P2_Read_ADCF4_24B(module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in dem die Messwerte (24 Bit Auflösung) gespeichert werden.	ARRAY LONG FLOAT
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 **P2_Read_ADCF24**.

Siehe auch

[P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Read_ADCF](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF_SConv24](#), [P2_Read_ADCF8_24B](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/18 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Init:
    P2_Start_ConvF(1,0Fh)     'Start AD-Wandlung Kanäle 1...4

Event:
    P2_Wait_EOCF(1,0Fh)       'Warten auf Wandlungsende
    P2_Read_ADCF4_24B(1,value,1)'Werte der ADC 1...4 lesen
    P2_Start_ConvF(1,0Fh)     'Neue AD-Wandlung starten
```

P2_Read_ADCF4_24B

P2_Read_ADCF8

P2_Read_ADCF8 liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus.

Syntax

```
#Include ADWINPRO_ALL.Inc  
  
P2_Read_ADCF8(module, array[], index)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY LONG FLOAT
<code>index</code>	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 **P2_Read_ADCF**.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF_SConv](#)

Gültig für

Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADWINPRO_ALL.Inc  
Dim value[8] As Long 'Feld für Messwerte  
  
Init:  
    Rem Start AD-Wandlung Kanäle 1...8; nicht erforderl. für Aln-F-8/14  
    P2_Start_ConvF(1,0FFh)  
  
Event:  
    Rem Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14  
    P2_Wait_EOCF(1,0FFh)  
    P2_Read_ADCF8(1,value,1) 'Werte der ADC 1...8 lesen  
    Rem Neue AD-Wandlung starten; nicht erforderlich für Aln-F-8/14  
    P2_Start_ConvF(1,0FFh)
```

P2_Read_ADCF8_24B liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus. Die Ergebnisse haben eine Auflösung von 24 Bit.

Syntax

```
#Include ADWINPRO_ALL.Inc

P2_Read_ADCF8_24B(module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in dem die Messwerte (24 Bit Auflösung) gespeichert werden.	ARRAY LONG FLOAT
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 **P2_Read_ADCF24**.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF_SConv24](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-8/18 Rev. E

Beispiel

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

Init:
    P2_Start_ConvF(1,0FFh)  'Start AD-Wandlung Kanäle 1...8

Event:
    P2_Wait_EOCF(1,0FFh)    'Warten auf Wandlungsende
    P2_Read_ADCF8_24B(1,value,1)'Werte der ADC 1...8 lesen
    P2_Start_ConvF(1,0FFh)  'Neue AD-Wandlung starten
```

P2_Read_ADCF8_24B

P2_Read_ADCF4_Packed

P2_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

P2_Read_ADCF4_Packed(module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY LONG FLOAT
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
array[index]	F-ADC 2	F-ADC 1
array[index+1]	F-ADC 4	F-ADC 3

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF_SConv](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Init:
    Rem Start AD-Wandlung Kanäle 1...4; nicht erforderl. für Aln-F-8/14
    P2_Start_ConvF(1, 0Fh)

Event:
    Rem Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14
    P2_Wait_EOCF(1, 0Fh)
    P2_Read_ADCF4_Packed(1, value, 1) 'Werte der ADC 1...4 lesen
    Rem Neue AD-Wandlung starten; nicht erforderlich für Aln-F-8/14
    P2_Start_ConvF(1, 0Fh)
```

P2_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

P2_Read_ADCF8_Packed(module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY LONG FLOAT
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
array[index]	F-ADC 2	F-ADC 1
array[index+1]	F-ADC 4	F-ADC 3
array[index+2]	F-ADC 6	F-ADC 5
array[index+3]	F-ADC 8	F-ADC 7

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF_SConv](#)

Gültig für

Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Init:
    Rem Start AD-Wandlung Kanäle 1...8; nicht erforderl. für Aln-F-8/14
    P2_Start_ConvF(1, 0FFh)

Event:
    Rem Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14
    P2_Wait_EOCF(1, 0FFh)
    P2_Read_ADCF8_Packed(1, value, 1) 'Werte der ADC 1...8 lesen
    Rem Neue AD-Wandlung starten; nicht erforderlich für Aln-F-8/14
    P2_Start_ConvF(1, 0FFh)
```

P2_Read_ADCF8_Packed

P2_Read_ADCF32

P2_Read_ADCF32 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 = P2_Read_ADCF32(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Kennziffer (1...2 oder 1...4) für das zu lesende F-ADC-Paar.	LONG

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

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.

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

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Read_ADCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF_SConv](#), [P2_Read_ADCF_SConv32](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/14 Rev. E, Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Event:

```
Rem Start AD-Wandlung Kanäle 1,2; nicht erfordl. für Aln-F-8/14
P2_Start_ConvF(1,11b)
Rem Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14
P2_Wait_EOCF(1,3)
value1 = P2_Read_ADCF32(1,1)'Wert von ADC1 und ADC2 einlesen
```


P2_Read_ADCF_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 = P2_Read_ADCF_SConv(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Read_ADCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF32](#), [P2_Read_ADCF_SConv32](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[1000] As Long 'Feld für Messwerte

Init:
  i=1
  P2_Start_ConvF(1,1) 'A/D-Wandler starten

Event:
  P2_Wait_EOCF(1,1)
  Data_1[i] = P2_Read_ADCF_SConv(1,1) 'A/D-Wandler auslesen +
                                     'starten
  Inc(i) 'Index erhöhen
  If (i=1001) Then End 'Nach 1000 Messwerten Prozess beenden
```

P2_Read_ADCF_SConv

P2_Read_ADCF_SConv24

P2_Read_ADCF_SConv24 liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung. Der Rückgabewert hat eine Auflösung von 24 Bit.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_Read_ADCF_SConv24(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Nummer des zu lesenden ADC (1...4 oder 1...8).	LONG
ret_val	Im F-ADC-Register enthaltener Messwert (0...16777215 = $2^{24}-1$).	LONG

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Read_ADCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF32](#), [P2_Read_ADCF_SConv32](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/18 Rev. E, Aln-F-8/18 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
Dim i As Long  
Dim Data_1[1000] As Long 'Deklaration  
  
Init:  
  i=1  
  P2_Start_ConvF(1,1) 'A/D-Wandler starten  
  
Event:  
  P2_Wait_EOCF(1,1)  
  Data_1[i] = P2_Read_ADCF_SConv24(1,1) 'A/D-Wandler 24 Bit  
                                     'auslesen + starten  
  Inc(i) 'Index erhöhen  
  If (i=1001) Then End 'Nach 1000 Messwerten Prozess beenden
```

P2_Read_ADCF_SConv32 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 = P2_Read_ADCF_SConv32(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG										
adc_no	Nummer des ersten zu lesenden F-ADC (1...2 oder 1...4).	LONG										
<table><tr><td>adc_no</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>F-ADC-Nr.</td><td>1, 2</td><td>3, 4</td><td>5, 6</td><td>7, 8</td></tr></table>			adc_no	1	2	3	4	F-ADC-Nr.	1, 2	3, 4	5, 6	7, 8
adc_no	1	2	3	4								
F-ADC-Nr.	1, 2	3, 4	5, 6	7, 8								
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										

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Read_ADCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Read_ADCF32](#), [P2_Read_ADCF_SConv](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Init:
    P2_Start_ConvF(1,3)    'Start AD-Wandlung

Event:
    P2_Wait_EOCF(1,3)      'Warten auf das Ende der Konvertierung
    value = P2_Read_ADCF_SConv32(1,1) 'Wert vom ADC1 und ADC2
                                     'einlesen und die Wandlung
                                     'beider ADC neu starten
```

P2_Read_ADCF_SConv32

P2_Set_Gain

P2_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

P2_Set_Gain(module, channel, mode)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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	±10V
1	2	±5V
2	4	±2,5V
3	8	±1,25V

Siehe auch

[P2_ADCF24](#), [P2_Read_ADCF](#), [P2_Start_ConvF](#), [P2_Wait_EOCF](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-8/16 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.Inc
#Define ainadr 1 'Moduladresse AIN Modul

Init:
    Rem Spannungsbereich im Kanal 4 auf Betriebsart 1 stellen
    Rem Messbereich: +5V...-5V
    P2_Set_Gain(ainadr,4,1)

Event:
    Par_1 = P2_ADCF(1,4) 'Misst einen Wert vom Kanal 4
```

P2_Start_ConvF startet die Wandlung auf einem oder mehreren F-ADC des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc

P2_Start_ConvF(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Bitmuster, das die ADC festlegt, deren Konvertierung gestartet werden soll (siehe Tabelle).	LONG

Bit-Nr..	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 einzelne Wandlung mit dem Befehl **P2_Sync_All** synchron zu anderen Messungen starten, falls Sie das Modul mittels **P2_Sync_Enable** zur Synchronisation frei gegeben haben.

Sie können mehrere Wandlungen ebenfalls synchron ausführen, wenn Sie die entsprechenden Module mit **P2_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

[P2_ADCF](#), [P2_ADCF24](#), [P2_Read_ADCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#), [P2_Wait_EOCF](#), [P2_Sync_All](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Event:
P2_Start_ConvF(1,1) 'Start AD-Wandlung auf Kanal 1
P2_Wait_EOCF(1,1) 'Warten auf Wandlung-Ende
value = P2_Read_ADCF(1,1) 'Wert vom ADC einlesen
```

P2_Start_ConvF

P2_Wait_EOCF

P2_Wait_EOCF wartet, bis die Wandlung auf allen angegebenen F-ADC des Moduls abgeschlossen ist.

Syntax

```
#Include ADwinPro_All.Inc

P2_Wait_EOCF(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Bitmuster, das die ADC festlegt, auf deren Konvertierungsende gewartet werden soll:	LONG

Bit-Nr.	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.

Für das Modul Aln-F-x/14 ist **P2_Wait_EOCF** überflüssig, weil die ADC kontinuierlich mit fester Abtastrate arbeiten. Der Befehl hat keine Wirkung außer dem Verlust von Prozessorzeit.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_Start_ConvF](#), [P2_Read_ADCF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Read_ADCF4_Packed](#), [P2_Read_ADCF8_Packed](#)

Gültig für

Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E

Beispiel

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

Event:
P2_Start_ConvF(1,1) 'Start AD-Wandlung
P2_Wait_EOCF(1,1) 'Warten auf das Ende der Konvertierung
value = P2_Read_ADCF(1,1) 'Wert vom ADC einlesen
```

3.4 Pro II: Analoge Ausgänge

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit analogen Ausgängen gelten:

- [P2_DAC](#) (Seite 116)
- [P2_DAC4](#) (Seite 117)
- [P2_DAC4_Packed](#) (Seite 118)
- [P2_DAC8](#) (Seite 120)
- [P2_DAC8_Packed](#) (Seite 121)
- [P2_Start_DAC](#) (Seite 123)
- [P2_Write_DAC](#) (Seite 124)
- [P2_Write_DAC4](#) (Seite 125)
- [P2_Write_DAC4_Packed](#) (Seite 126)
- [P2_Write_DAC8](#) (Seite 127)
- [P2_Write_DAC8_Packed](#) (Seite 128)
- [P2_Write_DAC32](#) (Seite 129)

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.



P2_DAC

P2_DAC gibt auf einem Kanal des angegebenen Moduls eine (analoge) Spannung aus, die dem angegebenen Digitalwert entspricht.

Syntax

```
#Include ADwinPro_All.Inc

P2_DAC(module, dac_no, value)
```

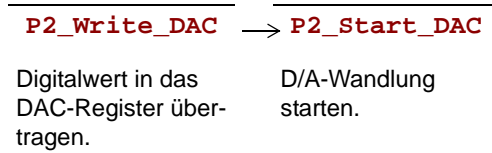
Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dac_no	Nummer (1...4 oder 1...8) des Ausgangs.	LONG
value	Auszugebender Wert (0...65535).	LONG

Bemerkungen

Wir empfehlen, die Befehle **P2_DAC4** oder **P2_DAC8** zu verwenden, weil sie in der gleichen Zeit wie **P2_DAC** eine größere Anzahl von Werten ausgeben können.

Der Befehl **P2_DAC** besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Siehe auch

[P2_DAC4](#), [P2_DAC8_Packed](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-4/16 Rev. E, AOut-8/16 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
Rem Digitaler P-Regler
#include ADwinPro_All.Inc
Dim sp, dev As Long
Dim g, actuate As Long
```

Event:

```
sp = Par_1           ' Sollwert
g = Par_2            ' Verstärkung
dev = sp - P2_ADC(1,1) ' Regelabweichung berechnen
actuate = dev * g     ' Stellgröße berechnen
P2_DAC(1,1,actuate)  ' Ausgabe der Stellgröße
```


P2_DAC4 gibt 4 Digitalwerte aus einem Feld auf die DAC 1...4 des angegebenen Moduls als (analoge) Spannung aus.

Syntax

```
#Include ADwinPro_All.Inc

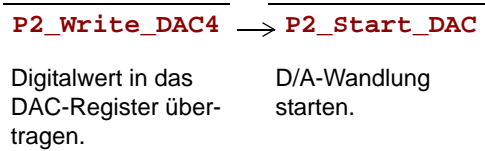
P2_DAC4(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535).	ARRAY
		LONG
		FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Bemerkungen

Der Befehl **P2_DAC4** besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Siehe auch

[P2_DAC](#), [P2_DAC8_Packed](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-4/16 Rev. E, AOut-8/16 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

Rem Digitaler P-Regler für 4Kanäle

```
#Include ADwinPro_All.Inc
Dim i, sp, dev As Long
Dim g, actuate As Long
Dim array[4] As Long
```

Event:

```
sp = Par_1           'Sollwert
g = Par_2            'Verstärkung
P2_Read_ADCF4(1,array,1) '4 Eingangswerte lesen
For i = 1 To 4
    dev = sp - array[i] 'Regelabweichung berechnen
    array[i] = dev * g   'Stellgröße berechnen
Next i
P2_DAC4(2,array,1)     '4 Stellgrößen ausgeben
```

P2_DAC4_Packed

P2_DAC4_Packed gibt 4 gepackte Digitalwerte aus einem Feld auf die DAC 1...4 des angegebenen Moduls als (analoge) Spannung aus.

Syntax

```
#Include ADwinPro_All.Inc
```

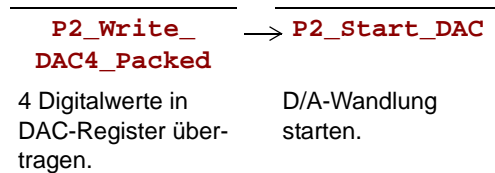
```
P2_DAC4_Packed(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535) in gepackter Form: Je 2 Werte zu 16 Bit in einem 32 Bit-Wert.	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Bemerkungen

Der Befehl **P2_DAC4_Packed** besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Jeweils 2 Werte zu 32 Bit im Feld enthalten 4 Digitalwerte zu 16 Bit in folgender Form:

Feldelement	array[n+1]		array[n]	
Bit-Nr.	31:16	15:00	31:16	15:00
Digitalwert für	DAC4	DAC3	DAC2	DAC1

Siehe auch

[P2_DAC](#), [P2_DAC8_Packed](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-4/16 Rev. E, AOut-8/16 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

Rem Digitaler P-Regler für 4Kanäle

```
#Include ADwinPro_All.Inc
```

```
Dim sp, dev1, dev2 As Long
```

```
Dim i, g As Long
```

```
Dim array[2] As Long
```

Event:

```
sp = Par_1 'Sollwert
```

```
g = Par_2 'Verstärkung
```

```
P2_Read_ADCF4_Packed(1,array,1)'4 Eingangswerte lesen
```

```
For i = 1 To 2
```

```
    Rem Regelabweichungen berechnen
```

```
    dev1 = sp - (array[i] And 0FFh)
```

```
    dev2 = sp - (Shift_Right(array[i],16) And 0FFh)
```

```
    Rem Stellgrößen berechnen und speichern
```

```
    array[i] = Shift_Left(dev2*g, 16) + dev1*g
```

```
Next i
```

```
P2_DAC4_Packed(2,array,1)'4 Stellgrößen ausgeben
```

P2_DAC8

P2_DAC8 gibt 8 Digitalwerte aus einem Feld auf die DAC 1...8 des angegebenen Moduls als (analoge) Spannung aus.

Syntax

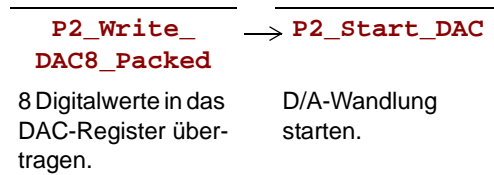
```
#Include ADwinPro_All.Inc  
  
P2_DAC8(module,array[],index)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Feld mit den auszugebenden Werten (0...65535).	ARRAY
		LONG
		FLOAT
<code>index</code>	Index des ersten auszugebenden Feldelements.	LONG

Bemerkungen

Der Befehl **P2_DAC8** besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4](#),
[P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#),
[P2_Write_DAC32](#)

Gültig für

AOut-8/16 Rev. E

Beispiel

Rem Digitaler P-Regler für 4Kanäle

```
#Include ADwinPro_All.Inc  
Dim sp, dev As Long  
Dim i, g, actuate As Long  
Dim array[8] As Long
```

Event:

```
sp = Par_1           'Sollwert  
g = Par_2            'Verstärkung  
P2_Read_ADCF8(1,array,1) '8 Eingangswerte lesen  
For i = 1 To 8  
    dev = sp - array[i] 'Regelabweichung berechnen  
    array[i] = dev * g  'Stellgröße berechnen  
Next i  
P2_DAC8(2,array,1)    '8 Stellgrößen ausgeben
```

P2_DAC8_Packed gibt die Digitalwerte aus einem Feld auf den DAC 1...8 des angegebenen Moduls als (analoge) Spannung aus.

Syntax

```
#Include ADwinPro_All.Inc
```

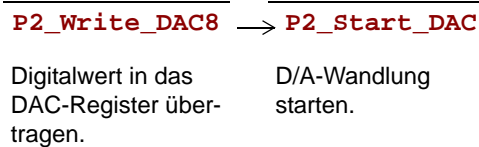
```
P2_DAC8_Packed(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535) in gepackter Form: Je 2 Werte zu 16 Bit in einem 32 Bit-Wert.	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Bemerkungen

Der Befehl **P2_DAC8_Packed** besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Jeweils 4 Werte zu 32 Bit im Feld enthalten 8 Digitalwerte zu 16 Bit in folgender Form:

Feldelement	array[n+3]	array[n+2]	array[n+1]	array[n]
Bit-Nr.	31:16	15:00	31:16	15:00
Digitalwert für	DAC8	DAC7	DAC6	DAC5
			DAC4	DAC3
				DAC2
				DAC1

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-8/16 Rev. E

P2_DAC8_Packed

Beispiel

```
Rem Digitaler P-Regler für 8 Kanäle
#include ADwinPro_All.inc
Dim sp, dev1, dev2 As Long
Dim i, g As Long
Dim array[4] As Long

Event:
    sp = Par_1                'Sollwert
    g = Par_2                'Verstärkung
    P2_Read_ADCF8_Packed(1,array,1)'8 Eingangswerte lesen
    For i = 1 To 4
        Rem Regelabweichungen berechnen
        dev1 = sp - (array[i] And 0FFh)
        dev2 = sp - (Shift_Right(array[i],16) And 0FFh)
        Rem Stellgrößen berechnen und speichern
        array[i] = Shift_Left(dev2*g, 16) + dev1*g
    Next i
    P2_DAC8_Packed(2,array,1) '8 Stellgrößen ausgeben
```

P2_Start_DAC startet die Wandlung bzw. Ausgabe aller DAC des angegebenen D/A-Moduls.

Syntax

```
#Include ADwinPro_All.Inc

P2_Start_DAC(module)
```

Parameter

module Eingestellte Moduladresse (1...15). LONG

Bemerkungen

Die Wandlung kann auch synchron mit anderen Modulen gestartet werden. Verwenden Sie hierzu den Befehl **P2_Sync_All**.

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_Packed](#), [P2_Sync_All](#), [P2_Write_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-4/16 Rev. E, AOut-8/16 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

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
```

```
Init:
```

```
  i = 0
```

```
Event:
```

```
  P2_Write_DAC(1,1,i)            'Ausgaberegister DAC1 setzen
```

```
  P2_Write_DAC(1,2,65535-i) 'Ausgaberegister DAC2 setzen
```

```
  P2_Start_DAC(1)            'Ausgabe auf allen DAC starten
```

```
  Inc(i)
```

```
  If (i=65535) Then i=0
```

P2_Start_DAC

P2_Write_DAC

P2_Write_DAC schreibt einen Digitalwert in das Ausgaberegister eines bestimmten DAC des angegebenen Moduls.

Der Befehl **P2_Start_DAC** startet die Wandlung des Digitalwerts in eine Ausgangsspannung.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_Write_DAC(module, dac_no, value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dac_no	Nummer (1...4 oder 1...8) des Ausgangs.	LONG
value	Auszugebender Wert (0...65535)	LONG

Bemerkungen

Wir empfehlen, die Befehle **P2_Write_DAC4**, **P2_Write_DAC4** oder **P2_Write_DAC8** zu verwenden, weil sie in der gleichen Zeit wie **P2_Write_DAC** eine größere Anzahl von Werten ausgeben können.

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_Packed](#), [P2_Start_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-4/16 Rev. E, AOut-8/16 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

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  
Dim Data_1[1000], Data_2[1000], Data_3[1000] As Long  
Dim Data_4[1000] As Long  
  
Init:  
    i = 1  
  
Event:  
    P2_Write_DAC(1,1,Data_1[i]) 'Ausgaberegister DAC1 setzen  
    P2_Write_DAC(1,2,Data_2[i]) 'Ausgaberegister DAC2 setzen  
    P2_Write_DAC(1,3,Data_3[i]) 'Ausgaberegister DAC3 setzen  
    P2_Write_DAC(1,4,Data_4[i]) 'Ausgaberegister DAC4 setzen  
    P2_Start_DAC(1)             'Ausgabe auf allen DAC starten  
    Inc(i)  
    If (i>1000) Then i = 1
```


P2_Write_DAC4 schreibt 4 Digitalwerte aus einem Feld in die Ausgaberegister der DAC 1...4 des angegebenen Moduls.

Der Befehl **P2_Start_DAC** startet die Wandlung der Digitalwerte in die Ausgangsspannungen.

Syntax

```
#Include ADwinPro_All.Inc

P2_Write_DAC4(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535).	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_Packed](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-4/16 Rev. E, AOut-8/16 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

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 nacheinander in einem DATA-Feld
Rem abgelegt und können vor dem Programmstart vom PC übergeben
Rem werden.*

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[4000] As Long
```

Init:

```
i = 1
```

Event:

```
'Ausgaberegister DAC 1...4 setzen
P2_Write_DAC4(1,Data_1,(i-1)*4+i)
P2_Start_DAC(1) 'Ausgabe auf allen DAC starten
Inc(i)
If (i>1000) Then i = 1
```

P2_Write_DAC4

P2_Write_DAC4_Packed

P2_Write_DAC4_Packed schreibt 4 Digitalwerte aus einem Feld in die Ausgaberegister der DAC 1...4 des angegebenen Moduls.

Der Befehl **P2_Start_DAC** startet die Wandlung der Digitalwerte in die Ausgangsspannungen.

Syntax

```
#Include ADwinPro_All.Inc

P2_Write_DAC4_Packed(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535) in gepackter Form: Je 2 Werte zu 16 Bit in einem 32 Bit-Wert.	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Bemerkungen

Jeweils 2 Werte zu 32 Bit im Feld enthalten 4 Digitalwerte zu 16 Bit in folgender Form:

Feldelement	array[n+1]		array[n]	
Bit-Nr.	31:16	15:00	31:16	15:00
Digitalwert für	DAC4	DAC3	DAC2	DAC1

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_Packed](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-4/16 Rev. E, AOut-8/16 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

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 nacheinander in einem DATA-Feld
Rem gepackt abgelegt und können vor dem Programmstart vom PC
Rem übergeben werden.*

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[4000] As Long
```

```
Init:
    i = 1
```

Event:

```
'Ausgaberegister DAC 1...4 setzen
P2_Write_DAC4_Packed(1,Data_1,(i-1)*2+i)
P2_Start_DAC(1)          'Ausgabe auf allen DAC starten
Inc(i)
If (i>1000) Then i = 1
```

P2_Write_DAC8 schreibt 8 Digitalwerte aus einem Feld in die Ausgaberegister der DAC 1...8 des angegebenen Moduls.

Der Befehl **P2_Start_DAC** startet die Wandlung der Digitalwerte in die Ausgangsspannungen.

Syntax

```
#Include ADwinPro_All.Inc

P2_Write_DAC8(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535).	ARRAY
		LONG
		FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Siehe auch

P2_DAC, P2_DAC4, P2_DAC8_Packed, P2_Start_DAC, P2_Write_DAC, P2_Write_DAC4, P2_Write_DAC4_Packed, P2_Write_DAC8_Packed, P2_Write_DAC32

Gültig für

AOut-8/16 Rev. E

Beispiel

Beispiel

*Rem Simultane Ausgabe von vier verschiedenen Signalverläufen
Rem auf den Ausgängen 1...8 eines D/A-Moduls.
Rem Die Signalverläufe sind nacheinander in einem DATA-Feld
Rem abgelegt und können vor dem Programmstart vom PC übergeben
Rem werden.*

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[8000] As Long
```

Init:

```
i = 1
```

Event:

```
Rem Ausgaberegister DAC 1...8 setzen
P2_Write_DAC8(1,Data_1,(i-1)*8+i)
P2_Start_DAC(1) 'Ausgabe auf allen DAC starten
Inc(i)
If (i>1000) Then i = 1
```

P2_Write_DAC8

P2_Write_DAC8_Packed

P2_Write_DAC8_Packed schreibt 8 Digitalwerte aus einem Feld in die Ausgaberegister der DAC 1...8 des angegebenen Moduls.

Der Befehl **P2_Start_DAC** startet die Wandlung der Digitalwerte in die Ausgangsspannungen.

Syntax

```
#Include ADwinPro_All.Inc

P2_Write_DAC8_Packed(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535) in gepackter Form: Je 2 Werte zu 16 Bit in einem 32 Bit-Wert.	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Bemerkungen

Jeweils 4 Werte zu 32 Bit im Feld enthalten 8 Digitalwerte zu 16 Bit in folgender Form:

Feldelement	array[n+3]	array[n+2]	array[n+1]	array[n]
Bit-Nr.	31:16	15:00	31:16	15:00
Digitalwert für	DAC8	DAC7	DAC6	DAC5
				DAC4
				DAC3
				DAC2
				DAC1

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_Packed](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#), [P2_Write_DAC32](#)

Gültig für

AOut-8/16 Rev. E

Beispiel

Beispiel

*Rem Simultane Ausgabe von vier verschiedenen Signalverläufen
Rem auf den Ausgängen 1...8 eines D/A-Moduls.*

*Rem Die Signalverläufe sind nacheinander in einem DATA-Feld
Rem gepackt abgelegt und können vor dem Programmstart vom PC
Rem übergeben werden.*

```
#Include ADwinPro_All.Inc
Dim i As Long
Dim Data_1[8000] As Long
```

Init:

```
i = 1
```

Event:

Rem Ausgaberegister DAC 1...8 setzen

```
P2_Write_DAC8_Packed(1,Data_1,(i-1)*4+i)
```

```
P2_Start_DAC(1) 'Ausgabe auf allen DAC starten
```

```
Inc(i)
```

```
If (i>1000) Then i = 1
```

P2_Write_DAC32 kopiert aus einem 32 Bit-Wert zwei 16 Bit-Werte in die Ausgaberegister eines DAC-Paars des angegebenen Moduls.
Die Wandlung in eine Ausgangsspannung erfolgt durch den Aufruf des Befehls **P2_Start_DAC**.

Syntax

```
#Include ADwinPro_All.Inc

P2_Write_DAC32(module,dac_no,value32)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dac_no	Wahl des DAC-Paars: 0: DAC 1 und 2 1: DAC 3 und 4 2: DAC 5 und 6 3: DAC 7 und 8	LONG
value32	Ausgebender Wert (0h...0FFFFFFFh).	LONG

Siehe auch

Das untere Wort (Bits 0...15) des Digitalwerts **value32** wird in den DAC mit der ungeraden Nummer geschrieben, das obere Wort (Bits 16...31) in den DAC mit der geraden Nummer.

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_Packed](#), [P2_Start_DAC](#), [P2_Write_DAC](#), [P2_Write_DAC4](#), [P2_Write_DAC4_Packed](#), [P2_Write_DAC8](#), [P2_Write_DAC8_Packed](#)

Gültig für

AOut-4/16 Rev. E, AOut-8/16 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

*Rem Simultane Ausgabe von zwei verschiedenen Signalverläufen
Rem auf den Ausgängen 3 und 4 eines D/A-Moduls.
Rem Die Signalverläufe sind in zwei 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] As Long
Dim array[1000] As Long

Init:
For i = 1 To 1000
    array[i] = Shift_Left(Data_2[i],16) + Data_1[i]
Next i
i = 1

Event:
P2_Write_DAC32(1,2,array[i]) 'Ausgaberegister DAC 3+4 setzen
P2_Start_DAC(1) 'Ausgabe auf allen DAC starten
Inc(i)
If (i>1000) Then i=1
```

P2_Write_DAC32

3.5 Pro II: Digitale Ein-/Ausgänge

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

- [P2_Dig_FIFO_Mode](#) (Seite 131)
- [P2_Dig_Latch](#) (Seite 133)
- [P2_Dig_Read_Latch](#) (Seite 134)
- [P2_Dig_Write_Latch](#) (Seite 135)
- [P2_Digin_Edge](#) (Seite 136)
- [P2_Digin_FIFO_Clear](#) (Seite 137)
- [P2_Digin_FIFO_Enable](#) (Seite 138)
- [P2_Digin_FIFO_Full](#) (Seite 140)
- [P2_Digin_FIFO_Read](#) (Seite 141)
- [P2_Digin_Fifo_Read_Fast](#) (Seite 143)
- [P2_Digin_FIFO_Read_Timer](#) (Seite 145)
- [P2_Digin_Long](#) (Seite 146)
- [P2_Digout](#) (Seite 147)
- [P2_Digout_Bits](#) (Seite 148)
- [P2_Digout_FIFO_Clear](#) (Seite 149)
- [P2_Digout_FIFO_Empty](#) (Seite 150)
- [P2_Digout_FIFO_Enable](#) (Seite 151)
- [P2_Digout_FIFO_Read_Timer](#) (Seite 152)
- [P2_Digout_FIFO_Start](#) (Seite 153)
- [P2_Digout_FIFO_Write](#) (Seite 154)
- [P2_Digout_Long](#) (Seite 156)
- [P2_Digout_Reset](#) (Seite 157)
- [P2_Digout_Set](#) (Seite 158)
- [P2_DigProg](#) (Seite 159)
- [P2_Get_Digout_Long](#) (Seite 160)

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 DIO-Modul die Adresse 1 eingestellt ist.



P2_Dig_FIFO_Mode stellt den Betriebsmodus des FIFO auf dem angegebenen Modul ein als Eingang mit Flankenüberwachung oder als Ausgang mit Flankenausgabe.

Syntax

```
#Include ADwinPro_All.inc

P2_Dig_FIFO_Mode(module, mode)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
mode	Betriebsmodus des FIFO: 0: Eingangs-FIFO zur Flankenüberwachung. Default. 1: Ausgangs-FIFO zur Flankenausgabe, Zeitwerte absolut. 3: Ausgangs-FIFO zur Flankenausgabe, Zeitwerte relativ.	LONG

Bemerkungen

Der Ausgangs-FIFO steht seit Revision DIO-32-TiCo Rev. E 03 zur Verfügung.

Zeitstempel des Ausgangs-FIFO geben den Ausgabezeitpunkt einer Flanke an. Die Werte eines Zeitstempels können absolut oder relativ angegeben werden:

- Absolutwert: Der Zeitstempel bezieht sich auf den Startzeitpunkt 0 des Modulzählers (**P2_Digout_FIFO_Start**). In diesem Modus kann der aktuelle Zählerstand mit **P2_Digout_FIFO_Read_Timer** gelesen werden.
- Relativwert: Der Zeitstempel wird relativ zum vorherigen Zeitstempel angegeben.

Siehe auch

[P2_Digin_FIFO_Enable](#), [P2_Digout_FIFO_Read_Timer](#), [P2_Digout_FIFO_Start](#), [P2_Digout_FIFO_Write](#)

Gültig für

DIO-32-TiCo Rev. E

P2_Dig_FIFO_Mode

Beispiel

```
#Include ADwinPro_All.inc
#Define module 2
Dim value[4] As Long

Init:
    Processdelay = 6000      '6000 x 3.3 ns = 20µs
    value[1] = 01b           'output value n
    value[2] = 5000          ' with output time 50 µs (relative)
    value[3] = 10b           'output value n+1
    value[4] = 7000          ' with output time 70 µs (relative)
    P2_Digprog(module,0Fh)   'set all channels as output
    P2_Dig_FIFO_Mode(module,3) 'Set FIFO as relative output
    P2_Digout_FIFO_Clear(module) 'clear FIFO
    P2_Digout_FIFO_Enable(module,11b) 'Enable output channels 0+1
    Rem write 2 value pairs into output FIFO and start output
    P2_Digout_FIFO_Write(module,2,value,1)
    P2_Digout_FIFO_Start(Shift_Left(1,module-1))

Event:
    Rem write new value pairs into FIFO, if possible
    If (P2_Digout_FIFO_Empty(module) > 2) Then
        P2_Digout_FIFO_Write(module,2,value,1)
    EndIf
```


P2_Dig_Latch überträgt auf dem angegebenen Modul Digital-Informationen von den Eingängen in die Eingangs-Latches und von den Ausgangs-Latches zu den Ausgängen.

Syntax

```
#Include ADwinPro_All.inc

P2_Dig_Latch(module)
```

Parameter

module Eingestellte Moduladresse (1...15).

LONG

Bemerkungen

Bei digitalen Eingängen überträgt die Anweisung die Eingangs-Signale an die Eingangs-Latches. Bei digitalen Ausgängen überträgt die Anweisung die Werte der Ausgangs-Latches an die Ausgänge.

Wenn das angegebene Modul durch **P2_Sync_Enable** zur Synchronisation freigeschaltet ist, hat der Befehl **P2_Sync_All** die gleiche Funktion wie **P2_Dig_Latch**.

Siehe auch

[P2_Dig_Read_Latch](#), [P2_Dig_Write_Latch](#), [P2_DigProg](#), [P2_Digin_Long](#), [P2_Digout_Bits](#), [P2_Digout](#), [P2_Digout_Long](#), [P2_Get_Digout_Long](#), [P2_Sync_All](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

Rem Kanäle 0...15 als Ausgang setzen, 16...31 als Eingang

```
P2_Digprog(1,0011b)
```

```
P2_Dig_Write_Latch(1,0) 'Alle Ausgangs-Bits auf 0 setzen
```

Event:

```
P2_Dig_Latch(1) 'Eingänge latchen, Inhalt des
                'Ausgangs-Latches ausgeben
```

Rem weitere Programmschritte

```
Par_1 = P2_Dig_Read_Latch(1) 'Eingangsbits einlesen und beim...
```

```
P2_Dig_Write_Latch(1,Par_1) 'nächsten Event ausgeben
```

P2_Dig_Latch

P2_Dig_Read_Latch

P2_Dig_Read_Latch liefert die Bitwerte aus dem Latch-Register für digitale Eingänge auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc  
ret_val = P2_Dig_Read_Latch(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Bitwerte des Latch-Registers. Jedes Bit entspricht einem digitalen Eingang (siehe Tabelle).	LONG

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **P2_DigProg** als Eingänge zu programmieren.

Sie können den aktuellen Zustand der digitalen Eingänge mit folgenden Anweisungen in das Zwischenregister übernehmen:

- **P2_Dig_Latch**
- **P2_Sync_All** (falls für das Modul aktiviert).

Siehe auch

[P2_Dig_Latch](#), [P2_Dig_Write_Latch](#), [P2_DigProg](#), [P2_Digin_Long](#), [P2_Sync_All](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

Beispiel

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

Init:

```
Rem DIO31:00 der Module 1+2 als Eingänge setzen  
P2_Digprog(1,0000b)  
P2_Digprog(2,0000b)
```

Event:

```
P2_Sync_All(11b)           'Pegel an den digitalen Eingängen von  
                           'beiden Modulen synchron in die  
                           'Zwischenregister übernehmen  
Par_1 = P2_Dig_Read_Latch(1)'Zwischenregister Modul 1 auslesen  
Par_2 = P2_Dig_Read_Latch(2)'Zwischenregister Modul 2 auslesen
```

P2_Dig_Write_Latch schreibt einen 32 Bit-Wert in das Latch-Register für digitale Ausgänge auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

P2_Dig_Write_Latch(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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 der Anweisung **P2_DigProg** als Ausgänge programmiert werden.

Sie können den Wert des Zwischenregisters für die digitalen Ausgänge auch mit der Anweisung **P2_Digout** setzen.

Bei der Modulversion TRA-16-G Rev. E schaltet der Pegel High nach Masse, nicht nach V_{CC} .

Siehe auch

[P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_DigProg](#), [P2_Get_Digout_Long](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    P2_Digprog(1,1111b)      'DIO31:00 des Moduls als Ausgang

Event:
    P2_Dig_Latch(1)          'Informationen des Ausgangs-Latches
                             'auf einer DIO-32-Karte ausgeben
    P2_Dig_Write_Latch(1,Par_1)'Long-Word ins Ausgangs-Latch
                             'schreiben
```

P2_Dig_Write_Latch

P2_Digin_Edge

P2_Digin_Edge gibt zurück, ob an den Digitaleingängen des angegebenen Moduls eine positive oder negative Flanke aufgetreten ist.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Digin_Edge(module, edge)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
edge	Art der zu prüfenden Flanke: 1: Auf positive Flanke prüfen. 0: Auf negative Flanke prüfen.	LONG
ret_val	Bitmuster, das angibt, an welchen Eingängen eine Flanke aufgetreten ist. Die Zuordnung der Bits zu den Eingängen ist unten dargestellt. Bit = 1: Flanke ist aufgetreten. Bit = 0: Keine Flanke aufgetreten.	LONG

Bitnr.	31	30	...	2	1	0
Eingang	31	30	...	2	1	0

Bemerkungen

Ein gesetztes Bit in **ret_val** bedeutet, dass die gesuchte Flanke seit dem vorigen Abfragen mindestens einmal am Digitaleingang aufgetreten ist. Für Ausgangskanäle sind die Bits immer Null.

Der Aufruf von **P2_Digin_Edge** setzt alle Bits zurück auf 0.

Siehe auch

[P2_Digin_FIFO_Clear](#), [P2_Digin_FIFO_Enable](#), [P2_Digin_FIFO_Full](#), [P2_Digin_FIFO_Read](#), [P2_Digin_FIFO_Read_Timer](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    P2_Digprog(1,1100b)           'Kanäle 0:15 als Eingänge

Event:
    Rem positive und negative Flanken prüfen, Ausgänge ausmaskieren
    Par_1 = P2_Digin_Edge(1,1) And 0Fh
    Par_2 = P2_Digin_Edge(1,0) And 0Fh

    Rem Flankenänderungen auf Ausgänge geben
    If (Par_1 + Par_2 > 0) Then
        P2_Digout_Bits(1,Shift_Left(Par_1,16),Shift_Left(Par_2,16))
    EndIf
```

P2_Digin_FIFO_Clear löscht den FIFO der Flankenüberwachung auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

P2_Digin_FIFO_Clear(module)
```

Parameter

module Eingestellte Moduladresse (1...15).

LONG

Bemerkungen

- / -

Siehe auch

[P2_Digin_FIFO_Enable](#), [P2_Digin_FIFO_Full](#), [P2_Digin_FIFO_Read](#),
[P2_Digin_FIFO_Read_Timer](#), [P2_Digin_Edge](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

```
Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long
```

Init:

```
P2_Digprog(1,1100b)            'Kanäle 0:15 als Eingänge
P2_Digin_FIFO_Enable(1,0) 'Überwachung aus
P2_Digin_FIFO_Clear(1)        'FIFO löschen
P2_Digin_FIFO_Enable(1,10011b) 'Kanäle 1,2,5 überwachen
index = 1
```

Event:

```
num = P2_Digin_FIFO_Full(1) 'Anzahl Wertepaare
If (num>50) Then
  If (index+num>10000) Then index = 1
  Rem Wertepaare auslesen
  P2_Digin_FIFO_Read(1, num, Data_1, Data_2, index)
  index=index + num
EndIf
```

P2_Digin_FIFO_Clear

P2_Digin_FIFO_Enable

P2_Digin_FIFO_Enable legt fest, an welchen Eingangskanälen des angegebenen Moduls die Flanken überwacht werden.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_Digin_FIFO_Enable(module, channels)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channels	Bitmuster, das die zu überwachenden Eingangskanäle festlegt.	LONG

Bitnr.	31	30	...	2	1	0
Eingang	31	30	...	2	1	0

Bemerkungen

Es können nur Eingangskanäle überwacht werden. Die Kanäle werden mit **P2_DigProg** als Eingänge oder Ausgänge programmiert.

Der FIFO sollte mit **P2_Dig_FIFO_Mode** für die Flankenüberwachung aktiviert werden.

Die Flankenüberwachung prüft alle 10ns, ob an den festgelegten Eingangskanälen eine Flanke aufgetreten ist bzw. ob sich ein Pegel geändert hat. Sobald eine Flanke aufgetreten ist, wird ein Wertepaar in ein FIFO-Feld kopiert:

- Wert 1 enthält den Pegelzustand aller Kanäle als Bitmuster.
- Wert 2 enthält einen Zeitstempel, den aktuellen Stand eines 100MHz-Zählers.

Das FIFO-Feld kann maximal 511 Wertepaare (Pegelzustand und Zeitstempel) enthalten. Wenn das FIFO-Feld voll ist, können keine weiteren Wertepaare gespeichert werden und gehen damit verloren.

Siehe auch

[P2_Dig_FIFO_Mode](#), [P2_Digin_FIFO_Clear](#), [P2_Digin_FIFO_Full](#), [P2_Digin_FIFO_Read](#), [P2_Digin_FIFO_Read_Timer](#), [P2_Digin_Edge](#), [P2_DigProg](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
  P2_Digprog(1,1100b)      'Kanäle 0:15 als Eingänge
  P2_Digin_FIFO_Enable(1,0)'Überwachung aus
  P2_Digin_FIFO_Clear(1)   'FIFO löschen
  P2_Digin_FIFO_Enable(1,10011b)'Kanäle 1,2,5 überwachen
  index = 1

Event:
  num = P2_Digin_FIFO_Full(1)'Anzahl Wertepaare
  If (num>50) Then
    Rem Wertepaare auslesen
    P2_Digin_FIFO_Read(1, num, Data_1, Data_2, index)
    index = index + num
    If (index > 10000) Then index = 1
  EndIf
```

P2_Digin_FIFO_Full

P2_Digin_FIFO_Full gibt die Anzahl der gespeicherten Wertepaare im FIFO der Flankenüberwachung zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_value = P2_Digin_FIFO_Full(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_value	Anzahl (0...511) der belegten Wertepaare im FIFO.	LONG

Bemerkungen

Das FIFO-Feld kann maximal 511 Wertepaare (Pegelszustand und Zeitstempel) enthalten. Wenn das FIFO-Feld voll ist, können keine weiteren Wertepaare gespeichert werden und gehen damit verloren.

Siehe auch

[P2_Digin_FIFO_Clear](#), [P2_Digin_FIFO_Enable](#), [P2_Digin_FIFO_Read](#), [P2_Digin_FIFO_Read_Timer](#), [P2_Digin_Edge](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc  
  
Dim Data_1[10000], Data_2[10000] As Long  
Dim num, index As Long  
  
Init:  
P2_Digprog(1,1100b) 'Kanäle 0:15 als Eingänge  
P2_Digin_FIFO_Enable(1,0) 'Überwachung aus  
P2_Digin_FIFO_Clear(1) 'FIFO löschen  
P2_Digin_FIFO_Enable(1,10011b) 'Kanäle 1,2,5 überwachen  
index = 1  
  
Event:  
num = P2_Digin_FIFO_Full(1) 'Anzahl Wertepaare  
If (num>50) Then  
Rem Wertepaare auslesen  
P2_Digin_FIFO_Read(1, num, Data_1, Data_2, index)  
index = index + num  
If (index > 10000) Then index = 1  
EndIf
```


P2_Digin_FIFO_Read liest die Wertepaare aus dem FIFO der Flankenüberwachung und schreibt sie in 2 Felder.

Syntax

```
#Include ADwinPro_All.inc

P2_Digin_FIFO_Read(module, count, value[],
    timestamp[], start_index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl (1...511) der zu lesenden Wertepaare.	LONG
value[]	Feld, in das die Bitmuster der Pegelzustände geschrieben werden. Die Zuordnung der Bits zu den Eingängen ist unten dargestellt.	LONG ARRAY
timestamp[]	Feld, in das die Zeitstempel geschrieben werden.	LONG ARRAY
start_index	Startindex für die Felder value[] und timestamp[] , ab dem die Daten geschrieben werden.	LONG

Bitnr.	31	30	...	2	1	0
Eingang	31	30	...	2	1	0

Bemerkungen

Es dürfen nicht mehr Wertepaare gelesen werden als im FIFO gespeichert sind. Dazu muss vor dem Auslesen mit **P2_Digin_FIFO_Full** gepüft werden, wieviele Wertepaare im FIFO gespeichert sind.

Die Felder müssen so groß dimensioniert sein, dass alle gelesenen Werte gespeichert werden können.

Der Zeitabstand zwischen 2 Pegelzuständen ist die Differenz der zugehörigen Zeitstempel, gemessen in Einheiten von 10ns:

$$\Delta t = (\text{stamp}_1 - \text{stamp}_2) \cdot 10 \text{ ns}$$

Siehe auch

[P2_Digin_FIFO_Clear](#), [P2_Digin_FIFO_Enable](#), [P2_Digin_FIFO_Full](#), [P2_Digin_FIFO_Read_Timer](#), [P2_Digin_Edge](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

P2_Digin_FIFO_Read

Beispiel

```
#Include ADwinPro_All.inc

Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
    P2_Digprog(1,1100b)           'Kanäle 0:15 als Eingänge
    P2_Digin_FIFO_Enable(1,0) 'Überwachung aus
    P2_Digin_FIFO_Clear(1)       'FIFO löschen
    P2_Digin_FIFO_Enable(1,10011b) 'Kanäle 1,2,5 überwachen
    index = 1

Event:
    num = P2_Digin_FIFO_Full(1) 'Anzahl Wertepaare
    If (num>50) Then
        Rem Wertepaare auslesen
        P2_Digin_FIFO_Read(1, num, Data_1, Data_2, index)
        index=index + num
        If (index>10000) Then index = 1
    EndIf
```

P2_Digin_Fifo_Read_Fast liest die Wertepaare aus dem FIFO der Flankenüberwachung und schreibt sie in ein einzelnes Feld.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_Digin_Fifo_Read_Fast(module, count, valuepairs[],  
start_index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl (1...511) der zu lesenden Wertepaare.	LONG
valuepairs[]	Feld, in das die Wertepaare geschrieben werden, abwechselnd ein Bitmuster der Pegelzustände und ein Zeitstempel. Die Zuordnung der Bits zu den Eingängen ist unten dargestellt.	LONG ARRAY
start_index	Startindex für das Feld valuepairs[] , ab dem die Daten geschrieben werden.	LONG

Bitnr.	31	30	...	2	1	0
Eingang	31	30	...	2	1	0

Bemerkungen

P2_Digin_Fifo_ReadEs dürfen nicht mehr Wertepaare gelesen werden als im FIFO gespeichert sind. Dazu muss vor dem Auslesen mit **P2_Digin_FIFO_Full** geprüft werden, wieviele Wertepaare im FIFO gespeichert sind.

Das Feld muss so groß dimensioniert sein, dass alle gelesenen Wertepaare gespeichert werden können.

Im Feld **value[]** werden die Wertepaare aus Pegelzustand und zugehörigem Zeitstempel abgelegt:

- Ein Feldelement enthält den Pegelzustand der Kanäle 0...31 als Bitmuster.
- Das nächste Feldelement enthält einen Zeitstempel (absolut oder relativ, siehe **P2_Dig_FIFO_Mode**).

Der Zeitabstand zwischen 2 Pegelzuständen ist die Differenz der zugehörigen Zeitstempel, gemessen in Einheiten von 10ns:

$$\Delta t = (\text{stamp}_1 - \text{stamp}_2) \cdot 10 \text{ ns}$$

Siehe auch

[P2_Digin_FIFO_Clear](#), [P2_Digin_FIFO_Enable](#), [P2_Digin_FIFO_Full](#), [P2_Digin_FIFO_Read_Timer](#), [P2_Digin_Edge](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

P2_Digin_Fifo_Read_Fast

Beispiel

```
#Include ADwinPro_All.inc

Dim Data_1[20000] As Long
Dim num, index As Long

Init:
    P2_Digprog(1,1100b)          'Kanäle 0:15 als Eingänge
    P2_Digin_FIFO_Enable(1,0) 'Überwachung aus
    P2_Digin_FIFO_Clear(1)      'FIFO löschen
    P2_Digin_FIFO_Enable(1,10011b) 'Kanäle 1,2,5 überwachen
    index = 1

Event:
    num = P2_Digin_FIFO_Full(1) 'Anzahl Wertepaare
    If (num > 50) Then
        Rem Wertepaare auslesen
        P2_Digin_Fifo_Read_Fast(1, num, Data_1, index)
        index = index + num
        If (index > 10000) Then index = 1
    EndIf
```

P2_Digin_FIFO_Read_Timer gibt den aktuellen Stand des 100MHz-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Digin_FIFO_Read_Timer(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Aktueller Stand ($-2^{31}-1 \dots 2^{31}$) des 100MHz-Zählers.	LONG

Bemerkungen

Der Modulzähler wird für das Erzeugen der Zeitstempel bei der Flankenüberwachung benutzt, siehe **P2_Digin_FIFO_Enable**.

Der Zähler wird alle 10ns um 1 erhöht, so dass der Zähler nach jeweils etwa 43 Sekunden ($= 10\text{ns} \times 2^{32}$) seinen ursprünglichen Wert erneut erreicht. Bei Zeitvergleichen muss dieser „Überlauf“ berücksichtigt werden, der Zählerstand muss daher im Programm regelmäßig vor dem Überlauf abgefragt werden.

Siehe auch

[P2_Digin_FIFO_Enable](#), [P2_Digin_FIFO_Read](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Dim Data_1[10000], Data_2[10000] As Long
Dim num, index As Long

Init:
  P2_Digprog(1,1100b)      'Kanäle 0:15 als Eingänge
  P2_Digin_FIFO_Enable(1,0)'Überwachung aus
  P2_Digin_FIFO_Clear(1)   'FIFO löschen
  P2_Digin_FIFO_Enable(1,10011b)'Kanäle 1,2,5 überwachen
  index = 1

Event:
  num = P2_Digin_FIFO_Full(1)'Anzahl Wertepaare
  If (num>50) Then
    Rem Wertepaare auslesen
    P2_Digin_FIFO_Read(1, num, Data_1, Data_2, index)
    index=index + num
    If (index>10000) Then index = 1
  EndIf
```

P2_Digin_FIFO_Read_Timer

P2_Digin_Long

P2_Digin_Long gibt den Zustand der Eingänge (Bits 31...00) des angegebenen Moduls als Bitmuster zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Digin_Long(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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	...	2	1	0
Eingang	31	30	...	2	1	0

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **P2_DigProg** als Eingänge zu programmieren.

Siehe auch

[P2_Dig_Latch](#), [P2_DigProg](#), [P2_Digout_Long](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, OPT-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    P2_Digprog(1,0000b)      'DIO 31:00 als Eingang

Event:
    Par_1 = P2_Digin_Long(1) 'Alle Eingänge einlesen
```

P2_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

P2_Digout(module,output,value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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 der Anweisung **P2_DigProg** als Ausgänge programmiert werden.

Mit dieser Anweisung kann ein beliebiger Ausgang gelöscht oder gesetzt werden, ohne den Zustand der anderen Ausgänge zu ändern.

Bei der Modulversion TRA-16-G Rev. E schaltet der Pegel High nach Masse, nicht nach V_{CC} .

Siehe auch

[P2_Digout_Long](#), [P2_Digout_Bits](#), [P2_DigProg](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem nur für DIO32: Kanäle 0...15 als Eingang, 16...31 als Ausgang
P2_Digprog(1,1100b)
```

Event:

```
Rem Eingangsbits einlesen und prüfen, ob Kanal 15 gesetzt ist
If (P2_Digin_Long(1) And 8000h = 1) Then
    P2_Digout(1,31,0)      'Kanal 15 gesetzt: Bit 31 löschen
Else
    P2_Digout(1,31,1)      'Kanal 15 gelöscht: Bit 31 setzen
EndIf
```

P2_Digout

P2_Digout_Bits

P2_Digout_Bits setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel High oder den Pegel Low.

Syntax

```
#Include ADwinPro_All.inc

P2_Digout_Bits(module, set, clear)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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 Leitungen müssen zunächst mit der Anweisung **P2_DigProg** als Ausgänge programmiert werden.

Mit dieser Anweisung können beliebige Ausgänge gelöscht oder gesetzt werden, ohne den Zustand der anderen Ausgänge zu ändern.

Zur Klarheit weisen wir darauf hin, dass Sie die im Bitmuster **set** gesetzten Bits nicht gleichzeitig im Bitmuster **clear** setzen dürfen und umgekehrt.

Bei der Modulversion TRA-16-G Rev. E schaltet der Pegel High nach Masse, nicht nach V_{CC}.

Siehe auch

[P2_Digout](#), [P2_Digout_Long](#), [P2_DigProg](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Kanäle 0...31 als Ausgang setzen
P2_Digprog(1,1111b)
```

Event:

```
If (Par_1 = 1) Then      'Bedingung abfragen
    Rem unteres Wort: MSB der Bytes setzen, andere Bits löschen
    P2_Digout_Bits(1,8080h,7F7Fh)
Else
    Rem unteres Wort: ungerade Bits setzen, gerade Bits löschen
    P2_Digout_Bits(1,5555h,0AAAh)
EndIf
```


P2_Digout_FIFO_Clear stoppt die Flankenausgabe und löscht den FIFO der Flankenausgabe auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_Digout_FIFO_Clear (module)
```

Parameter

module Eingestellte Moduladresse (1...15). LONG

Bemerkungen

Der FIFO muss vor der ersten Anwendung gelöscht werden. Anschließend kann der FIFO über **P2_Digout_FIFO_Write** mit Daten befüllt werden.

Wenn die Flankenausgabe mit **P2_Digout_FIFO_Clear** gestoppt wurde, kann sie nur mit **P2_Digout_FIFO_Start** neu gestartet werden.

Siehe auch

[P2_Digout_FIFO_Enable](#), [P2_Dig_FIFO_Mode](#), [P2_Digout_FIFO_Start](#), [P2_Digout_FIFO_Write](#), [P2_DigProg](#)

Gültig für

DIO-32-TiCo Rev. E03

Beispiel

siehe [P2_Dig_FIFO_Mode](#)

P2_Digout_FIFO_Clear

P2_Digout_FIFO_Empty

P2_Digout_FIFO_Empty gibt die Anzahl der freien Wertepaare im FIFO der Flankenausgabe zurück.

Syntax

```
#Include ADwinPro_All.inc  
ret_value = P2_Digout_FIFO_Empty(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_value	Anzahl (0...511) der freien Wertepaare im FIFO.	LONG

Bemerkungen

Das FIFO-Feld kann maximal 511 Wertepaare (Pegelzustand und Zeitstempel) enthalten.

Siehe auch

[P2_Dig_FIFO_Mode](#), [P2_Digout_FIFO_Read_Timer](#), [P2_Digout_FIFO_Start](#), [P2_Digout_FIFO_Write](#)

Gültig für

DIO-32-TiCo Rev. E03

Beispiel

siehe [P2_Dig_FIFO_Mode](#)

P2_Digout_FIFO_Enable legt fest, an welchen Ausgangskanälen des angegebenen Moduls Flanken ausgegeben werden.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_Digout_FIFO_Enable (module, channels)
```

Parameter

module Eingestellte Moduladresse (1...15). LONG

channels Bitmuster, das die auszugebenden Ausgangskanäle festlegt. LONG

Bitnr.	31	30	...	2	1	0
Ausgang	31	30	...	2	1	0

Bemerkungen

Flanken können nur auf Ausgangskanäle ausgegeben werden. Sie programmieren die Kanäle mit **P2_DigProg** als Eingänge oder Ausgänge.

Der FIFO muss mit **P2_Dig_FIFO_Mode** für die Flankenausgabe aktiviert werden.

Die Flankenausgabe gibt Flanken nur an den Ausgangskanälen aus, die mit **P2_Digout_FIFO_Enable** festgelegt sind. An den übrigen Ausgabekanälen – und zwar an diesen – können Sie die Pegel mit Befehlen wie **P2_Digout_Long** setzen.

Die Pegel und Zeitpunkte für die Flankenausgabe werden mit **P2_Digout_FIFO_Write** festgelegt.

Siehe auch

[P2_Digout_FIFO_Clear](#), [P2_Dig_FIFO_Mode](#), [P2_Digout_FIFO_Start](#), [P2_Digout_FIFO_Write](#), [P2_DigProg](#), [P2_Digout_Long](#)

Gültig für

DIO-32-TiCo Rev. E03

Beispiel

siehe [P2_Dig_FIFO_Mode](#)

P2_Digout_FIFO_Enable

P2_Digout_FIFO_Read_Timer

P2_Digout_FIFO_Read_Timer gibt den aktuellen Stand des 100MHz-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_Digout_FIFO_Read_Timer(module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>ret_val</code>	Aktueller Stand ($-2^{31}-1$... 2^{31}) des 100MHz-Zählers.	LONG

Bemerkungen

Der Modulzähler wird für die zeitlich exakte Ausgabe von Flanken zu vorgegebenen Zeitpunkten benutzt, siehe **P2_Digout_FIFO_Write**.

Der Zählerstand kann nur im FIFO-Betriebsmodus mit absoluten Zeitwerten verwendet werden, d.h. Parameter `mode` = 1 bei **P2_Dig_FIFO_Mode**.

Der Zähler wird alle 10ns um 1 erhöht, so dass der Zähler nach jeweils etwa 43 Sekunden ($= 10\text{ns} \times 2^{32}$) seinen ursprünglichen Wert erneut erreicht. Eine „verpasste“ Flankenausgabe wird erst nach diesem „Überlauf“ ausgegeben.

Siehe auch

[P2_Dig_FIFO_Mode](#), [P2_Digout_FIFO_Empty](#), [P2_Digout_FIFO_Start](#), [P2_Digout_FIFO_Write](#), [P2_DigProg](#)

Gültig für

DIO-32-TiCo Rev. E03

Beispiel

- / -

P2_Digout_FIFO_Start startet die Ausgabe der Flankenausgabe auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

P2_Digout_FIFO_Start(module_pattern)
```

Parameter

module_ Bitmuster zum Ansprechen der Module: LONG
pattern Bit = 0: Modul ignorieren.
 Bit = 1: Flankenausgabe auf dem Modul starten.

Bits in module_pattern	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen

Mit dem Start beginnt der Modulzähler von 0 aus zu zählen. Der Modulzähler wird für die zeitgenaue Flankenausgabe benutzt, siehe **P2_Digout_FIFO_Write**.

Der Zähler wird alle 10ns um 1 erhöht, so dass der Zähler nach jeweils etwa 43 Sekunden ($= 10\text{ns} \times 2^{32}$) seinen ursprünglichen Wert erneut erreicht. Bei Zeitvergleichen muss dieser „Überlauf“ berücksichtigt werden, der Zählerstand muss daher im Programm regelmäßig vor dem Überlauf abgefragt werden.

Siehe auch

[P2_Digout_FIFO_Clear](#), [P2_Digout_FIFO_Enable](#), [P2_Dig_FIFO_Mode](#), [P2_Digout_FIFO_Read_Timer](#), [P2_Digout_FIFO_Write](#), [P2_DigProg](#)

Gültig für

DIO-32-TiCo Rev. E03

Beispiel

siehe [P2_Dig_FIFO_Mode](#)

P2_Digout_FIFO_Start

P2_Digout_FIFO_Write

P2_Digout_FIFO_Write schreibt Wertepaare in den FIFO der Flankenausgabe.

Syntax

```
#Include ADwinPro_All.inc

P2_Digout_FIFO_Read_Timer(module, count, value[],
    start_index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl (1...511) der zu schreibenden Wertepaare.	LONG
value[]	Feld, das abwechselnd Bitmuster der Pegelzustände und Zeitstempel für Ausgabezeitpunkte enthält. Die Zuordnung der Bits zu den Ausgängen ist unten dargestellt.	LONG ARRAY
start_index	Startindex für das Feld value[] , ab dem die Daten gelesen werden.	LONG

Bitnr.	31	30	...	2	1	0
Ausgang	31	30	...	2	1	0

Bemerkungen

Es dürfen nicht mehr Wertepaare geschrieben werden als im FIFO frei sind.

Das FIFO-Feld kann maximal 511 Wertepaare (Pegelzustand und Zeitstempel) enthalten. Wenn das FIFO-Feld voll ist, können keine weiteren Wertepaare hineingeschrieben werden.

Im Feld **value[]** müssen Wertepaare aus Pegelzustand und zugehörigem Zeitstempel abgelegt sein:

- Ein Feldelement mit ungeradem Index enthält den Pegelzustand der Kanäle 0...31 als Bitmuster.
- Ein Feldelement mit geradem Index enthält einen Zeitstempel (absolut oder relativ, siehe **P2_Dig_FIFO_Mode**). Der Abstand zwischen zwei Zeitpunkten muss mindestens 20ns betragen.

Die Ausgabe läuft ab wie folgt:

- Der Modulzähler wird alle 10ns um 1 hochgezählt.
- Wenn der Zählerstand gleich dem Zeitstempel des aktuellen Wertepaars im FIFO ist, wird das Bitmuster auf den festgelegten Kanälen ausgegeben.
- Wenn ein Bitmuster ausgegeben wurde, wird das Wertepaar aus dem FIFO gelöscht.
- Die Wertepaare werden in der Reihenfolge abgearbeitet, wie sie in den FIFO geschrieben wurden.

Es gilt daher:

Ein Zeitstempel definiert also den Ausgabezeitpunkt und zwar in Einheiten von 10ns. Der Wert kann auf zwei Weisen angegeben werden:

- als Absolutwert mit Bezug zum Start des Modulzählers mit **P2_Digout_FIFO_Start**.

Bei einem Zeitstempel von 153 wird das zugehörige Bitmuster genau 1,53µs nach Start des Modulzählers ausgegeben.

- als Relativwert mit Bezug zum vorherigen Zeitstempel.
Bei einem Zeitstempel von 153 wird das zugehörige Bitmuster genau 1,53µs nach der Ausgabe des vorherigen Bitmusters ausgegeben.

Zeitstempel müssen in aufsteigender Reihenfolge abgelegt werden.

Der FIFO muss immer so mit Daten gefüllt werden, dass der jeweils nächste Ausgabezeitpunkt in der Zukunft liegt. Das heißt:

- Bei Absolutwerten muss der Zeitstempel größer sein als der aktuelle Zählerstand. Anderenfalls wird die Flankenausgabe „verpasst“ und erst nach einem zusätzlichen Zählerumlauf von etwa 43 Sekunden ausgeführt.
- Bei Relativwerten muss der Zeitstempel größer sein als der Zeitraum seit dem vorherigen Ausgabezeitpunkt. Gelingt das nicht, wird das Bitmuster sofort ausgegeben (jedoch verspätet); der nächste Zeitstempel bezieht sich auf den verspäteten Ausgabezeitpunkt.

Siehe auch

[P2_Digout_FIFO_Empty](#), [P2_Digout_FIFO_Enable](#), [P2_Dig_FIFO_Mode](#), [P2_Digout_FIFO_Read_Timer](#), [P2_Digout_FIFO_Start](#), [P2_Digout_Long](#), [P2_DigProg](#)

Gültig für

DIO-32-TiCo Rev. E03

Beispiel

siehe [P2_Dig_FIFO_Mode](#)

P2_Digout_Long

P2_Digout_Long setzt oder löscht durch den übergebenen 32 Bit-Wert alle Ausgänge des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.inc

P2_Digout_Long(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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 der Anweisung **P2_DigProg** als Ausgänge programmiert werden.

Bei der Modulversion TRA-16-G Rev. E schaltet der Pegel High nach Masse, nicht nach V_{CC}.

Siehe auch

[P2_Digout](#), [P2_Digout_Bits](#), [P2_DigProg](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    P2_Digprog(1, 01111b)      'DIO31:00 als Ausgang

Event:
    P2_Digout_Long(1, 1000000) 'Den Wert 1 Mio. als Binärwert
                                'auf die DIOs ausgeben
```


P2_Digout_Reset setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel Low.

Syntax

```
#Include ADwinPro_All.inc

P2_Digout_Reset(module, clear)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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 Leitungen müssen zunächst mit der Anweisung **P2_DigProg** als Ausgänge programmiert werden.

Mit dieser Anweisung können beliebige Ausgänge gelöscht werden, ohne den Zustand der anderen Ausgänge zu ändern.

Bei der Modulversion TRA-16-G Rev. E schaltet der Pegel High nach Masse, nicht nach V_{CC} .

Siehe auch

[P2_Digout](#), [P2_Digout_Bits](#), [P2_Digout_Long](#), [P2_Digout_Set](#), [P2_DigProg](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    Rem Kanäle 0...31 als Ausgang setzen
    P2_Digprog(1, 1111b)

Event:
    If (Par_1 = 1) Then          'Bedingung abfragen
        Rem unteres Wort: geradzahlige Bits löschen
        P2_Digout_Reset(1, 0AAAh)
    EndIf
```

P2_Digout_Reset

P2_Digout_Set

P2_Digout_Set setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel High.

Syntax

```
#Include ADwinPro_All.inc

P2_Digout_Set(module, set)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

Bitnr.	31	30	...	2	1	0
Ausgang	31	30	...	2	1	0

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **P2_DigProg** als Ausgänge programmiert werden.

Mit dieser Anweisung können beliebige Ausgänge gesetzt werden, ohne den Zustand der anderen Ausgänge zu ändern.

Zur Klarheit weisen wir darauf hin, dass Sie die im Bitmuster **set** gesetzten Bits nicht gleichzeitig im Bitmuster **clear** setzen dürfen und umgekehrt.

Bei der Modulversion TRA-16-G Rev. E schaltet der Pegel High nach Masse, nicht nach V_{CC}.

Siehe auch

[P2_Digout](#), [P2_Digout_Bits](#), [P2_Digout_Long](#), [P2_Digout_Reset](#), [P2_DigProg](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

Init:
    Rem Kanäle 0...31 als Ausgang setzen
    P2_Digprog(1,1111b)

Event:
    If (Par_1 = 1) Then          'Bedingung abfragen
        Rem unteres Wort: MSB der Bytes setzen
        P2_Digout_Set(1,8080h)
    EndIf
```

P2_DigProg programmiert die digitalen Kanäle 0...31 des angegebenen Moduls in Gruppen zu je 8 als Ein- oder Ausgang.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_Digprog(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

Bitnr.	31...4	3	2	1	0
Kanalnr.	–	31:24	23:16	15:08	07:00

Bemerkungen

Nach dem Einschalten des Systems sind alle Kanäle als Eingänge konfiguriert.

Die Kanäle können nur in Gruppen zu je 8 als Ein- oder Ausgang gesetzt werden (nur 4 relevante Bits, die anderen Bits werden ignoriert).

Siehe auch

[P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Dig_Write_Latch](#)

[P2_Digin_Long](#), [P2_Digout_Bits](#), [P2_Digout](#), [P2_Digout_Long](#), [P2_Get_Digout_Long](#)

[P2_Digin_FIFO_Enable](#), [P2_Digout_FIFO_Start](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Kanäle 0...7 des Moduls Nr. 1 als Eingang konfigurieren
Rem und Kanäle 8...31 als Ausgang
```

```
P2_Digprog(1, 1110b)
```

P2_DigProg

P2_Get_Digout_Long

P2_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 = P2_Get_Digout_Long(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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.

Bei der Modulversion TRA-16-G Rev. E schaltet der Pegel High nach Masse, nicht nach V_{CC} .

Siehe auch

[P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Dig_Write_Latch](#)
[P2_DigProg](#), [P2_Digin_Long](#), [P2_Digout_Bits](#), [P2_Digout](#), [P2_Digout_Long](#)

Gültig für

DIO-32 Rev. E, DIO-32-TiCo Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Event:

```
Rem Bits 31:00 aus dem Latch zurücklesen  
Par_1 = P2_Get_Digout_Long(1)
```

3.6 Pro II: Zähler

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit Zählern gelten:

- [P2_Cnt_Clear](#) (Seite 162)
- [P2_Cnt_Enable](#) (Seite 163)
- [P2_Cnt_PW_Enable](#) (Seite 164)
- [P2_Cnt_Get_Status](#) (Seite 165)
- [P2_Cnt_Get_PW](#) (Seite 166)
- [P2_Cnt_Get_PW_HL](#) (Seite 167)
- [P2_Cnt_Latch](#) (Seite 168)
- [P2_Cnt_Mode](#) (Seite 169)
- [P2_Cnt_PW_Latch](#) (Seite 171)
- [P2_Cnt_Read](#) (Seite 172)
- [P2_Cnt_Read4](#) (Seite 173)
- [P2_Cnt_Read_Int_Register](#) (Seite 174)
- [P2_Cnt_Read_Latch](#) (Seite 176)
- [P2_Cnt_Read_Latch4](#) (Seite 177)
- [P2_Cnt_Sync_Latch](#) (Seite 178)
- [P2_SSI_Mode](#) (Seite 180)
- [P2_SSI_Read](#) (Seite 181)
- [P2_SSI_Read2](#) (Seite 182)
- [P2_SSI_Set_Bits](#) (Seite 183)
- [P2_SSI_Set_Clock](#) (Seite 184)
- [P2_SSI_Set_Delay](#) (Seite 185)
- [P2_SSI_Start](#) (Seite 186)
- [P2_SSI_Status](#) (Seite 187)

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 Modul die Adresse 1 eingestellt ist.



P2_Cnt_Clear

P2_Cnt_Clear setzt einen oder mehrere Zähler auf Null, gemäß dem Bitmuster in **pattern**.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_Clear(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster Bit = 0: Kein Einfluss Bit = 1: Zähler auf Null setzen	LONG

Bit-Nr.	31...4	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

Nach Ausführung von **P2_Cnt_Clear** wird das Bitmuster automatisch auf 0 (Null) zurückgesetzt, d.h. die zurückgesetzten Zähler beginnen zu zählen.

Siehe auch

[P2_Cnt_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Latch](#), [P2_Cnt_Mode](#), [P2_Cnt_Read](#), [P2_Cnt_Read4](#), [P2_Cnt_Read_Latch](#), [P2_Cnt_Read_Latch4](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren

Init:
    old_1 = 0 'Variablen...
    old_2 = 0 'initialisieren
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,1,10000b)
    P2_Cnt_Mode(1,2,0) 'Alle Zähler auf externen
    Takteingang
    P2_Cnt_Clear(1,11b) 'Zähler 1+2 auf 0 zurücksetzen
    P2_Cnt_Enable(1,11b) 'Zähler 1+2 starten, Zähler 3+4
    stoppen

Event:
    P2_Cnt_Latch(1,11b) 'Zähler 1+2 gleichzeitig latchen
    new_1 = P2_Cnt_Read_Latch(1,1) 'Latch Zähler 1 und...
    new_2 = P2_Cnt_Read_Latch(1,2) 'Latch Zähler 2 auslesen.
    Par_1 = new_1 - old_1 'Differenz bilden (f = Impulse / Zeit)
    Par_2 = new_2 - old_2 ' - - -
    old_1 = new_1 'Neuen Zählerstand als alten
    speichern
    old_2 = new_2 ' - - -
```

P2_Cnt_Enable hält die gewählten Zähler an oder gibt sie frei, um eingehende Impulse zu zählen.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_Enable(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster Bit = 0: Zähler anhalten Bit = 1: Zähler freigeben	LONG

Bit-Nr.	31...4	3	2	1	0
Zähler-Nr.	–	VR4	VR3	VR2	VR1

Bemerkungen

PWM-Zähler werden mit **P2_Cnt_PW_Enable** angehalten oder freigegeben.

Siehe auch

[P2_Cnt_Clear](#), [P2_Cnt_PW_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Latch](#), [P2_Cnt_Mode](#), [P2_Cnt_Read](#), [P2_Cnt_Read4](#), [P2_Cnt_Read_Latch](#), [P2_Cnt_Read_Latch4](#), [P2_Cnt_Sync_Latch](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren

Init:
    old_1 = 0 'Variablen...
    old_2 = 0 'initialisieren
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,1,10000b)
    P2_Cnt_Mode(1,2,0) 'Alle Zähler auf externen
    Takteingang
    P2_Cnt_Clear(1,11b) 'Zähler 1+2 auf 0 zurücksetzen
    P2_Cnt_Enable(1,11b) 'Zähler 1+2 starten, Zähler 3+4
    stoppen

Event:
    P2_Cnt_Latch(1,11b) 'Zähler 1+2 gleichzeitig latches
    new_1 = P2_Cnt_Read_Latch(1,1) 'Latch Zähler 1 und...
    new_2 = P2_Cnt_Read_Latch(1,2) 'Latch Zähler 2 auslesen.
    Par_1 = new_1 - old_1 'Differenz bilden (f = Impulse / Zeit)
    Par_2 = new_2 - old_2 '---
    old_1 = new_1 'Neuen Zählerstand als alten
    speichern
    old_2 = new_2 '---'
```

P2_Cnt_Enable

P2_Cnt_PW_Enable

P2_Cnt_PW_Enable hält die gewählten PWM-Zähler an oder gibt sie frei, um eingehende Impulse zu zählen.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_PW_Enable(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster Bit = 0: Zähler anhalten Bit = 1: Zähler freigeben	LONG

Bit-Nr.	31...4	3	2	1	0
Zähler-Nr.	–	PW 4	PW 3	PW 2	PW 1

Bemerkungen

Standard-Zähler werden mit **P2_Cnt_Enable** angehalten oder freigegeben.

Siehe auch

[P2_Cnt_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Get_PW](#), [P2_Cnt_Get_PW_HL](#), [P2_Cnt_Mode](#), [P2_Cnt_PW_Latch](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren
Init:
    old_1 = 0 'Variablen...
    old_2 = 0 'initialisieren
    P2_Cnt_Mode(1,1,0) 'Zähler 1: PWM-Messung am
    Eingang A
    P2_Cnt_Mode(1,2,0) 'Zähler 2: PWM-Messung am
    Eingang A
    P2_Cnt_PW_Enable(1,0011b) 'PWM-Zähler 1+2 starten, 3+4
    stoppen

Event:
    P2_Cnt_PW_Latch(1,11b) 'Zähler 1+2 gleichzeitig latches
    REM High-/Low-Zeit lesen
    P2_Cnt_Get_PW_HL(1,1,Par_1,Par_2)
    REM Frequenz und Taktverhältnis lesen
    P2_Cnt_Get_PW(1,1,FPar_1,FPar_2)
```


P2_Cnt_Get_Status gibt den Inhalt des Statusregisters für einen Zähler zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Cnt_Get_Status(module, counter_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
counter_no	Zählernummer: 1...4.	LONG
ret_val	Inhalt des Statusregisters für den Zähler: Hinweise auf mögliche Fehlerquellen. Bedeutung der Bits 0...4 siehe Tabelle.	LONG

Bit Nr.	31...5	4	3	2	1	0
Signal	–	C	L	N	B	A
- :don't care (Signalzustände undefiniert, mit 01Fh ausmaskieren)						
A: Signal A (statisch)						
B: Signal B (statisch)						
N: CLR-/LATCH-Eingang (statisch)						
L: Leitungsfehler (Kabel abgezogen oder Leitung unterbrochen)						
C: Korrelationsfehler (Signal A und B sind identisch, d.h. nicht um ca. 90° phasenverschoben)						

Bemerkungen

Ein Leitungsfehler (L) kann nur bei differentiellen Eingängen detektiert werden! Bei TTL-Eingängen sind diese Bits stets 0.

Das Statusregister wird beim Auslesen automatisch zurückgesetzt.

Siehe auch

[P2_Cnt_Enable](#), [P2_Cnt_PW_Enable](#), [P2_Cnt_Get_PW](#), [P2_Cnt_Mode](#), [P2_Cnt_Read](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

- / -

P2_Cnt_Get_Status

P2_Cnt_Get_PW

P2_Cnt_Get_PW gibt Frequenz und Tastverhältnis eines PWM-Zählers zurück.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_Cnt_Get_PW(module, pwm_no, frequency, dutycycle)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pwm_no	Nummer (1...4) des PWM-Zählers.	LONG
frequency	Frequenz in Hertz: 0,023 Hz ...100MHz.	FLOAT
		CONST
dutycycle	Tastverhältnis in Prozent (0.0...100.0).	FLOAT
		CONST

Bemerkungen

Die Rückgabewerte werden in den Parametern **frequency** und **dutycycle** übergeben.

Siehe auch

[P2_Cnt_PW_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Get_PW_HL](#), [P2_Cnt_Mode](#), [P2_Cnt_PW_Latch](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren
Init:
    old_1 = 0 'Variablen...
    old_2 = 0 'initialisieren
    P2_Cnt_Mode(1,1,0) 'Zähler 1: PWM-Messung am
    Eingang A
    P2_Cnt_Mode(1,2,0) 'Zähler 2: PWM-Messung am
    Eingang A
    P2_Cnt_PW_Enable(1,0011b) 'PWM-Zähler 1+2 starten, 3+4
    stoppen

Event:
    P2_Cnt_PW_Latch(1,11b) 'Zähler 1+2 gleichzeitig latches
    REM High-/Low-Zeit lesen
    P2_Cnt_Get_PW_HL(1,1,Par_1,Par_2)
    REM Frequenz und Taktverhältnis lesen
    P2_Cnt_Get_PW(1,1,FPar_1,FPar_2)
```

P2_Cnt_Get_PW_HL gibt die Eintastzeit und die Austastzeit eines PWM-Zählers zurück.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_Get_PW_HL(module, counter_
no, hightime, lowtime)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
counter_ no	Zählernummer: 1...4.	LONG
hightime	Eintastzeit des PWM-Signals in Einheiten von 10ns.	LONG CONST
lowtime	Austastzeit des PWM-Signals in Einheiten von 10ns.	LONG CONST

Bemerkungen

Die Rückgabewerte werden in den Parametern **hightime** und **lowtime** übergeben.

Siehe auch

[P2_Cnt_PW_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Get_PW](#), [P2_Cnt_Mode](#), [P2_Cnt_PW_Latch](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren
Init:
    old_1 = 0 'Variablen...
    old_2 = 0 'initialisieren
    P2_Cnt_Mode(1,1,0) 'Zähler 1: PWM-Messung am
    Eingang A
    P2_Cnt_Mode(1,2,0) 'Zähler 2: PWM-Messung am
    Eingang A
    P2_Cnt_PW_Enable(1,0011b) 'PWM-Zähler 1+2 starten, 3+4
    stoppen

Event:
    P2_Cnt_PW_Latch(1,11b) 'Zähler 1+2 gleichzeitig latchen
    REM High-/Low-Zeit lesen
    P2_Cnt_Get_PW_HL(1,1,Par_1,Par_2)
    REM Frequenz und Taktverhältnis lesen
    P2_Cnt_Get_PW(1,1,FPar_1,FPar_2)
```

P2_Cnt_Latch

P2_Cnt_Latch überträgt den aktuellen Zählerstand eines oder mehrerer Zähler in das zugehörige Latch, je nach Bitmuster in **pattern**.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_Latch(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster Bit = 0: keine Funktion Bit = 1: Zählerstand in Latch übertragen	LONG

Bit-Nr.	31...4	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

Nach Ausführung des Befehls wird das Bitmuster automatisch auf 0 (Null) zurückgesetzt.

Das Latch wird mit **P2_Cnt_Read_Latch** in eine Variable ausgelesen.

Siehe auch

[P2_Cnt_Clear](#), [P2_Cnt_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Mode](#), [P2_Cnt_Read](#), [P2_Cnt_Read4](#), [P2_Cnt_Read_Latch](#), [P2_Cnt_Read_Latch4](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren

Init:
    old_1 = 0 'Variablen...
    old_2 = 0 'initialisieren
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,1,10000b)
    P2_Cnt_Mode(1,2,0) 'Alle Zähler auf externen
    Takteingang
    P2_Cnt_Clear(1,11b) 'Zähler 1+2 auf 0 zurücksetzen
    Rem Zähler 1+2 starten, Zähler 3+4 stoppen
    P2_Cnt_Enable(1,11b)

Event:
    P2_Cnt_Latch(1,11b) 'Zähler 1+2 gleichzeitig latchen
    new_1 = P2_Cnt_Read_Latch(1,1) 'Latch Zähler 1 und...
    new_2 = P2_Cnt_Read_Latch(1,2) 'Latch Zähler 2 auslesen.
    Par_1 = new_1 - old_1 'Differenz bilden (f = Impulse / Zeit)
    Par_2 = new_2 - old_2 ' - " -
    old_1 = new_1 'Neuen Zählerstand als alten
    speichern
    old_2 = new_2 ' - " -
```

P2_Cnt_Mode definiert die Betriebsart eines Zählers.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_Cnt_Mode(module, counter_no, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
counter_no	Zählernummer: 1...4.	LONG
pattern	Bitmuster zur Einstellung des Betriebsmodus des Zählers.	LONG

Bit-Nr.	Bedeutung
Bit 0	Zählermodus: Bit = 0: Takt-Richtungs-Modus. Bit = 1: A-B-Modus.
Bit 1	Löschmodus: Bit = 0: TTL-Pegel high am Eingang CLR setzt den Zählerstand auf Null. Bit = 1: Zähler löschen, wenn an allen Eingänge A, B, CLR der TTL-Pegel high anliegt. Nur im A-B-Modus.
Bit 2	Eingang A / CLK invertieren im Takt-Richtungs-Modus: Bit = 0: Eingang ist nicht invertiert. Bit = 1: Eingang ist invertiert.
Bit 3	Eingang B / DIR invertieren im Takt-Richtungs-Modus: Bit = 0: Eingang ist nicht invertiert. Bit = 1: Eingang ist invertiert.
Bit 4	Eingang CLR / LATCH einstellen. Bit = 0: Eingang CLR. Bit = 1: Eingang LATCH.
Bit 5	Eingang CLR / LATCH freigeben. Bit = 0: Eingang ist gesperrt. Bit = 1: Eingang ist freigegeben.
Bit 6	Auswahl der Signalfanke für PWM-Auswertung. Bit = 0: steigende Flanke. Bit = 1: fallende Flanke.
Bit 7,8	Auswahl eines Eingangs für PWM-Auswertung. 00b: Eingang A / CLK 01b: Eingang B / DIR 10b: Eingang CLR / LATCH
Bits 9...31	reserviert.

Bemerkungen

Verwenden Sie **P2_Cnt_Mode** möglichst nur bei gesperrtem Zähler, siehe **P2_Cnt_Enable**.

Im Standard-Löschmodus (Bit 1=0) wird der Zählerstand so lange auf Null gesetzt, wie der TTL-Pegel high anliegt. Zum Löschen muss der Eingang CLR mit Bit 5=1 freigegeben werden.

P2_Cnt_Mode

Siehe auch

[P2_Cnt_Clear](#), [P2_Cnt_Enable](#), [P2_Cnt_PW_Enable](#), [P2_Cnt_Get_Status](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.Inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren

Init:
    old_1 = 0 'Variablen...
    old_2 = 0 'initialisieren
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,1,10000b)
    P2_Cnt_Mode(1,2,0) 'Alle Zähler auf externen
    Takteingang
    P2_Cnt_Clear(1,11b) 'Zähler 1+2 auf 0 zurücksetzen
    Rem Zähler 1+2 starten, Zähler 3+4 stoppen
    P2_Cnt_Enable(1,11b)

Event:
    P2_Cnt_Latch(1,11b) 'Zähler 1+2 gleichzeitig latchen
    new_1 = P2_Cnt_Read_Latch(1,1) 'Latch Zähler 1 und...
    new_2 = P2_Cnt_Read_Latch(1,2) 'Latch Zähler 2 auslesen.
    Par_1 = new_1 - old_1 'Differenz bilden (f = Impulse / Zeit)
    Par_2 = new_2 - old_2 '---
    old_1 = new_1 'Neuen Zählerstand als alten
    speichern
    old_2 = new_2 '---
```

P2_Cnt_PW_Latch kopiert den Inhalt eines oder mehrerer PWM-Zähler in einen Zwischenspeicher.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_PW_Latch(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster Bit = 0: Kein Einfluss. Bit = 1: PWM-Zählerinhalt latchen.	LONG

Bit-Nr.	31...4	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

Der Zwischenspeicher wird mit **P2_Cnt_Get_PW** oder **P2_Cnt_Get_PW_HL** ausgelesen.

Siehe auch

[P2_Cnt_PW_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Get_PW](#), [P2_Cnt_Get_PW_HL](#), [P2_Cnt_Mode](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim old_1, new_1 As Long
Dim old_2, new_2 As Long

Init:
    old_1 = 0                'Variablen...
    old_2 = 0                'initialisieren
    P2_Cnt_Mode(1,1,0)      'Zähler 1: PWM-Messung am
    Eingang A
    P2_Cnt_Mode(1,2,0)      'Zähler 2: PWM-Messung am
    Eingang A
    P2_Cnt_PW_Enable(1,0011b)'PWM-Zähler 1+2 starten, 3+4
    stoppen

Event:
    P2_Cnt_PW_Latch(1,11b)   'Zähler 1+2 gleichzeitig latchen
    REM High-/Low-Zeit lesen
    P2_Cnt_Get_PW_HL(1,1,Par_1,Par_2)
    REM Frequenz und Taktverhältnis lesen
    P2_Cnt_Get_PW(1,1,FPar_1,FPar_2)
```

P2_Cnt_PW_Latch

P2_Cnt_Read

P2_Cnt_Read überträgt einen aktuellen Zählerstand in das zugehörige Latch und gibt ihn als Rückgabewert zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Cnt_Read(module, counter_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
counter_no	Zählernummer: 1...4.	LONG
ret_val	Zählerstand	LONG

Bemerkungen

Verwenden Sie den Rückgabewert in Berechnungen (z.B. Differenzen oder Zählrichtung) nur mit Variablen vom Typ [Long](#).

Siehe auch

[P2_Cnt_Clear](#), [P2_Cnt_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Latch](#), [P2_Cnt_Mode](#), [P2_Cnt_Read4](#), [P2_Cnt_Read_Latch](#), [P2_Cnt_Read_Latch4](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.Inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren

Init:
    P2_Cnt_Enable(1,0)           'Zähler stoppen
    old_1 = 0                    'Variablen...
    old_2 = 0                    'initialisieren
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,1,10000b)
    P2_Cnt_Mode(1,2,0)           'Alle Zähler auf externen
    Takteingang
    P2_Cnt_Clear(1,11b)          'Zähler 1+2 auf 0 zurücksetzen
    Rem Zähler 1+2 starten, Zähler 3+4 stoppen
    P2_Cnt_Enable(1,11b)

Event:
    new_1 = P2_Cnt_Read(1,1)     'Stand des Zählers 1 lesen
    new_2 = P2_Cnt_Read(1,2)     'Stand des Zählers 2 lesen
    Par_1 = new_1 - old_1 'Differenz bilden (f = Impulse / Zeit)
    Par_2 = new_2 - old_2 '---
    old_1 = new_1               'Neuen Zählerstand als alten
    speichern
    old_2 = new_2               '---
```


P2_Cnt_Read4 überträgt alle 4 Zählerstände in die zugehörigen Latches A und gibt sie in einem Feld zurück.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_Read4(module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in das die Zählerstände geschrieben werden.	ARRAY LONG
index	Erstes Element in array[] , das beschrieben wird.	LONG

Bemerkungen

Verwenden Sie die Rückgabewerte in **array[]** in Berechnungen (z.B. Differenzen oder Zählrichtung) nur mit Variablen vom Typ **Long**.

Siehe auch

[P2_Cnt_Clear](#), [P2_Cnt_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Latch](#), [P2_Cnt_Mode](#), [P2_Cnt_Read](#), [P2_Cnt_Read_Latch](#), [P2_Cnt_Read_Latch4](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim Data_1[4] AS LONG
Dim old[4], new[4] AS LONG
Dim i AS LONG

Init:
    P2_Cnt_Enable(1,0)           'Zähler stoppen
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,1,10000b)
    P2_Cnt_Mode(1,2,0)           'Alle Zähler auf externen
    Takteingang
    Rem Zähler 3: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,3,10000b)
    P2_Cnt_Mode(1,4,0)           'Alle Zähler auf externen
    Takteingang
    P2_Cnt_Clear(1,1111b)        'Alle Zähler auf 0 zurücksetzen
    P2_Cnt_Enable(1,1111b) 'Zähler starten

Event:
    P2_Cnt_Read4(1,new,1)        'Zählerstände in Feld new
    einlesen
    For i = 1 To 4
        Data_1[i] = new[i]-old[i] 'Differenz bilden (f = Impulse /
    Zeit)
        old[i] = new[i]           'Neuen Zählerstand als alten
    speichern
    Next i
```

P2_Cnt_Read4

P2_Cnt_Read_Int_Register

P2_Cnt_Read_Int_Register gibt den Inhalt eines Zählerregisters zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Cnt_Read_Int_Register(counter_no, reg_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
counter_no	Zählernummer: 1...4.	LONG
reg_no	Kennzahl (0...15) für ein Zählerregister, Zuordnungstabelle siehe unten.	LONG
ret_val	Inhalt des Zählerregisters.	LONG

reg_no	Register
0	Latch 1 für positive Flanken.
1	Latch 2 für positive Flanken.
2	Latch 3 für positive Flanken.
3	Latch 1 für negative Flanken.
4	Latch 2 für negative Flanken.
5	Latch 3 für negative Flanken.
6	Software-Latch für VR-Zähler.
7	Software-Latch für PWM-Zähler.
8	Schattenregister für Latch 1, positive Flanken.
9	Schattenregister für Latch 2, positive Flanken.
10	Schattenregister für Latch 3, positive Flanken.
11	Schattenregister für Latch 1, negative Flanken.
12	Schattenregister für Latch 2, negative Flanken.
13	Schattenregister für Latch 3, negative Flanken.
14	Schattenregister für Software-Latch, VR-Zähler.
15	Zählerstatus.

Bemerkungen

- / -

Siehe auch

[P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

siehe [P2_Cnt_Sync_Latch](#)



P2_Cnt_Read_Latch

P2_Cnt_Read_Latch gibt den Wert aus dem Latch eines Zählers als Rückgabewert zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Cnt_Read_Latch(module, counter_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
counter_no	Zählernummer: 1...4.	LONG
ret_val	Inhalt des Latch des Zählers	LONG

Bemerkungen

Verwenden Sie den Rückgabewert in Berechnungen (z.B. Differenzen oder Zählrichtung) nur mit Variablen vom Typ [Long](#).

Siehe auch

[P2_Cnt_Clear](#), [P2_Cnt_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Latch](#), [P2_Cnt_Mode](#), [P2_Cnt_Read](#), [P2_Cnt_Read4](#), [P2_Cnt_Read_Latch4](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.Inc
Dim old_1, new_1 AS LONG 'Variablen...
Dim old_2, new_2 AS LONG 'Dimensionieren

Init:
    old_1 = 0 'Variablen...
    old_2 = 0 'initialisieren
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,1,10000b)
    P2_Cnt_Mode(1,2,0) 'Alle Zähler auf externen
    Takteingang
    P2_Cnt_Clear(1,11b) 'Zähler 1+2 auf 0 zurücksetzen
    Rem Zähler 1+2 starten, Zähler 3+4 stoppen
    P2_Cnt_Enable(1,11b)

Event:
    P2_Cnt_Latch(1,11b) 'Zähler 1+2 gleichzeitig latchen
    new_1 = P2_Cnt_Read_Latch(1,1) 'Latch Zähler 1 und...
    new_2 = P2_Cnt_Read_Latch(1,2) 'Latch Zähler 2 auslesen.
    Par_1 = new_1 - old_1 'Differenz bilden (f = Impulse / Zeit)
    Par_2 = new_2 - old_2 '---
    old_1 = new_1 'Neuen Zählerstand als alten
    speichern
    old_2 = new_2 '---'
```

P2_Cnt_Read_Latch4 gibt die Werte aus den Latches A aller 4 Zähler in einem Feld zurück.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_Read_Latch4(module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in das die Zählerstände geschrieben werden.	ARRAY LONG
index	Erstes Element in array[] , das beschrieben wird.	LONG

Bemerkungen

Verwenden Sie die Rückgabewerte in **array[]** in Berechnungen (z.B. Differenzen oder Zählrichtung) nur mit Variablen vom Typ **Long**.

Siehe auch

[P2_Cnt_Clear](#), [P2_Cnt_Enable](#), [P2_Cnt_Get_Status](#), [P2_Cnt_Latch](#), [P2_Cnt_Mode](#), [P2_Cnt_Read](#), [P2_Cnt_Read4](#), [P2_Cnt_Read_Latch](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.Inc
Dim Data_1[4] AS LONG
Dim old[4], new[4] AS LONG
Dim i AS LONG

Init:
    P2_Cnt_Enable(1,0)           'Zähler stoppen
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,1,10000b)
    P2_Cnt_Mode(1,2,0)           'Alle Zähler auf externen
    Takteingang
    Rem Zähler 3: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(1,3,10000b)
    P2_Cnt_Mode(1,4,0)           'Alle Zähler auf externen
    Takteingang
    P2_Cnt_Clear(1,1111b)        'Alle Zähler auf 0 zurücksetzen
    P2_Cnt_Enable(1,1111b) 'Zähler starten

Event:
    P2_Cnt_Latch(1,1111b)        'Zähler gleichzeitig latchen
    P2_Cnt_Read_Latch4(1,new,1) 'Zählerstände in Feld new
    einlesen
    For i = 1 To 4
        Data_1[i] = new[i]-old[i] 'Differenz bilden (f = Impulse /
    Zeit)
        old[i] = new[i]           'Neuen Zählerstand als alten
    speichern
    Next i
```

P2_Cnt_Read_Latch4

P2_Cnt_Sync_Latch

P2_Cnt_Sync_Latch kopiert die Inhalte der gewählten Zähler und PWM-Zähler in Zwischenspeicher.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_Sync_Latch(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster Bit = 0: Kein Einfluss. Bit = 1: Zählerinhalt in Zwischenspeicher kopieren.	LONG

Bit-Nr.	31...4	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

Jedem Bit sind sowohl ein VR-Zähler als auch ein PWM-Zähler zugeordnet. Beide Zählerinhalte werden gleichzeitig kopiert. Der Befehl hat damit die gleiche Funktion wie **P2_Cnt_Latch** und **P2_Cnt_PW_Latch** zusammen.

Die Zwischenspeicher werden beispielsweise mit **P2_Cnt_Read_Latch** oder **P2_Cnt_Get_PW** ausgelesen.

Siehe auch

[P2_Cnt_Get_PW](#), [P2_Cnt_Latch](#), [P2_Cnt_Mode](#), [P2_Cnt_PW_Latch](#), [P2_Cnt_Read_Latch](#), [P2_Sync_All](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define frequency FPar_1
Dim time, edges As Long
Dim rest As Float
Dim oldpw, oldcnt, newpw, newcnt As Long
Dim pw_cnt As Long

Init:
    Processdelay = 3000000      '100Hz
    P2_Cnt_Enable(1,0001b)
    P2_Cnt_Mode(1,1,00000000b) 'mode: clock/dir
    P2_Cnt_Clear(1,0Fh)
    P2_Cnt_Enable(1,0Fh)      'enable standard counters
    P2_Cnt_PW_Enable(1,0Fh)   'enable PW counters
    P2_Cnt_PW_Latch(1,0Fh)    'copy all counter values
    oldpw = 0
    oldcnt = 0
    frequency = 0

Event:
    P2_Cnt_Sync_Latch(1,0001b) 'latch all counter values
    newcnt = P2_Cnt_Read_Latch(1,1) 'vr-cnt
    edges = (newcnt-oldcnt)      'number of edges between events
    If (edges <> 0) Then
        PW_cnt = P2_Cnt_Read_Int_Register(1,1,8)
        time = PW_cnt - oldpw    'calculate timebase
        frequency = edges*100000000/time 'frequency
        '(100000000->frequency of

P2-CNT-Module)
    oldcnt=newcnt              'store VR-counter value
    oldpw =newpw              'store PW-counter value
    EndIf
```

P2_SSI_Mode

P2_SSI_Mode stellt den Modus aller SSI-Decoder auf dem angegebenen Modul ein, entweder „single shot“ (einzelne lesen) und „continuous“ (kontinuierlich lesen).

Syntax

```
#Include ADwinPro_All.Inc

P2_SSI_Mode(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Betriebsmodus der SSI-Decoder, angegeben als Bitmuster. Jedem Decoder ist ein Bit zugeordnet (siehe Tabelle). Bit = 0: Modus „single shot“, der Decoder wird einmal ausgelesen. Bit = 1: Modus „continuous“, der Decoder 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 Decoders sofort. **P2_SSI_Start** ist hierzu nicht erforderlich. Mit **P2_SSI_Set_Delay** stellen Sie den Zeitabstand zwischen dem Einlesen von zwei Encoder-Werten ein.

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

[P2_SSI_Read](#), [P2_SSI_Read2](#), [P2_SSI_Set_Bits](#), [P2_SSI_Set_Clock](#), [P2_SSI_Set_Delay](#), [P2_SSI_Start](#), [P2_SSI_Status](#)

Gültig für

CNT-D Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc

Init:
P2_SSI_Set_Clock(1,200) 'CLK (Taktrate) = 250 kHz
P2_SSI_Set_Delay(1,1,250) 'Wartezeit Decoder 1: 5 µs
P2_SSI_Set_Delay(1,2,500) 'Wartezeit Decoder 2: 10 µs
P2_SSI_Set_Bits(1,1,23) 'Anzahl Bits = 23 (Decoder 1)
P2_SSI_Set_Bits(1,2,23) 'Anzahl Bits = 23 (Decoder 2)
P2_SSI_Mode(1,3) 'Continuous-Modus für beide Decoder

Event:
Par_1 = P2_SSI_Read(1,1) 'Positionswert Decoder 1 lesen
Par_2 = P2_SSI_Read(1,2) 'Positionswert Decoder 2 lesen
```


P2_SSI_Read gibt den zuletzt gespeicherten Zählerstand eines bestimmten SSI-Decoders auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_SSI_Read(module, dcd_r_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dcd_r_no	Nummer (1, 2) des SSI-Decoders, dessen Zählerstand auszulesen ist.	LONG
ret_val	Letzter Zählerstand des SSI-Decoders (= Abolutwert-Position des Encoders).	LONG

Bemerkungen

Ein Encoder-Wert wird dann gespeichert, wenn die durch **P2_SSI_SET_BITS** angegebene Anzahl von Bits eingelesen wurde.

Es wird immer diejenige Anzahl an Bits zurückgegeben, die mit der Anweisung **P2_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

[P2_SSI_Mode](#), [P2_SSI_Read2](#), [P2_SSI_Set_Bits](#), [P2_SSI_Set_Clock](#), [P2_SSI_Set_Delay](#), [P2_SSI_Start](#), [P2_SSI_Status](#)

Gültig für

CNT-D Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
Dim m, n, y As Long

Init:
P2_SSI_Set_Clock(1,50) 'CLK (Taktrate) = 1 MHz
P2_SSI_Set_Delay(1,1,250) 'Wartezeit Decoder: 5 µs
P2_SSI_Set_Bits(1,1,23) 'Anzahl Bits = 23 (Decoder 1)
P2_SSI_Mode(1,1) 'Continuous-Mode setzen (Decoder 1)

Event:
Par_1 = P2_SSI_Read(1,1) 'Positionswert (Decoder 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
```

P2_SSI_Read



P2_SSI_Read2

P2_SSI_Read2 gibt den zuletzt gespeicherten Zählerstand von beiden SSI-Decodern auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_SSI_Read2(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in das die Zählerstände geschrieben werden.	ARRAY LONG
index	Erstes Element in array[] , das beschrieben wird.	LONG

Bemerkungen

Ein Encoder-Wert wird dann gespeichert, wenn die durch **P2_SSI_Set_Bits** angegebene Anzahl von Bits eingelesen wurde.

Es wird immer diejenige Anzahl an Bits zurückgegeben, die mit der Anweisung **P2_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

[P2_SSI_Mode](#), [P2_SSI_Read](#), [P2_SSI_Set_Bits](#), [P2_SSI_Set_Clock](#), [P2_SSI_Set_Delay](#), [P2_SSI_Start](#), [P2_SSI_Status](#)

Gültig für

CNT-D Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Dim Data_1[2000] As Long

Init:
    P2_SSI_Set_Clock(1,50)      'CLK (clock rate) = 1 MHz
    P2_SSI_Set_Delay(1,1,250)  'waiting delay decoder 1: 5 µs
    P2_SSI_Set_Delay(1,2,1000) 'waiting delay decoder 2: 20 µs
    P2_SSI_Set_Bits(1,1,10)    '10 bits for decoder 1
    P2_SSI_Set_Bits(1,2,25)    '25 bits for decoder 2
    P2_SSI_Mode(1,3)           'Set continuous-mode (both decoders)
    Par_1 = 0

Event:
    Inc Par_1
    If (Par_1 > 1000) Then Par_1 = 1
    P2_SSI_Read2(1,Data_1,Par_1*2)'Read both position values
```



P2_SSI_Set_Bits stellt für einen SSI-Decoder 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

P2_SSI_Set_Bits(module, dcd_r_no, bit_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dcd_r_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-Decoders sollte mit der Anzahl der zu übertragenden Bits übereinstimmen.

Es wird immer diejenige Anzahl an Bits für einen Encoder-Wert erwartet, die mit **P2_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

[P2_SSI_Mode](#), [P2_SSI_Read](#), [P2_SSI_Read2](#), [P2_SSI_Set_Clock](#), [P2_SSI_Set_Delay](#), [P2_SSI_Start](#), [P2_SSI_Status](#)

Gültig für

CNT-D Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_SSI_Set_Clock(1,50)    'CLK (Taktrate) = 1 MHz
P2_SSI_Set_Delay(1,1,250)'waiting delay decoder 1: 5 µs
P2_SSI_Set_Delay(1,2,1000)'waiting delay decoder 2: 20 µs
P2_SSI_Mode(1,3)         'Continuous-Mode einstellen (für
                           'beide Decoder)

P2_SSI_Set_Bits(1,1,10)   '10 Bits für Decoder 1
P2_SSI_Set_Bits(1,2,25)   '25 Bits für Decoder 2
```

Event:

```
Par_1 = P2_SSI_Read(1,1) 'Positionswert (Decoder 1) auslesen
Par_2 = P2_SSI_Read(1,2) 'Positionswert (Decoder 2) auslesen
```

P2_SSI_Set_Bits



P2_SSI_Set_Clock

P2_SSI_Set_Clock stellt die Taktrate (ca. 12kHz bis 25MHz) auf dem angegebenen Modul ein, mit der der Decoder getaktet wird.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_SSI_Set_Clock(module,prescale)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
prescale	Teilerfaktor (1...4095) zur Einstellung der Taktrate nach der Formel: Taktrate = 50MHz / 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 100 verwendet, das entspricht einer Taktrate von 500kHz.

Bei Teilerfaktoren über 4095 werden die niederwertigsten 12 Bits als Teilerfaktor verwendet.

Die mögliche Taktfrequenz ist abhängig von Kabellänge, Kabeltyp und den jeweils verwendeten Send- und Empfangsbausteinen des Encoders bzw. Decoders. Grundsätzlich gilt als Regel: Je höher die Taktfrequenz, desto kürzer die mögliche Kabellänge.

Siehe auch

[P2_SSI_Mode](#), [P2_SSI_Read](#), [P2_SSI_Read2](#), [P2_SSI_Set_Bits](#), [P2_SSI_Set_Delay](#), [P2_SSI_Start](#), [P2_SSI_Status](#)

Gültig für

CNT-D Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_SSI_Set_Clock(1,10) 'CLK (Taktrate) = 5 MHz  
P2_SSI_Set_Delay(1,1,250) 'waiting delay decoder 1: 5 µs  
P2_SSI_Set_Delay(1,2,1000) 'waiting delay decoder 2: 20 µs  
P2_SSI_Mode(1,3) 'Continuous-Mode setzen  
'(für beide Decoder)  
P2_SSI_Set_Bits(1,1,10) 'Anzahl Bits = 10 (Decoder 1)  
P2_SSI_Set_Bits(1,2,25) 'Anzahl Bits = 25 (Decoder 2)
```

Event:

```
Par_1 = P2_SSI_Read(1,1) 'Positionswert (Decoder 1) auslesen  
Par_2 = P2_SSI_Read(1,2) 'Positionswert (Decoder 2) auslesen
```

P2_SSI_Set_Delay stellt für einen bestimmten SSI-Zähler auf dem angegebenen Modul den Zeitabstand zwischen dem Einlesen von zwei Encoder-Werten ein.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_SSI_Set_Delay(module, dcd_r_no, delay)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dcd_r_no	Nummer (1, 2) des SSI-Decoders, dessen Wartezeit einzustellen ist.	LONG
delay	Zeitabstand (1...65535) in Einheiten von 20ns; der einstellbare Bereich ist 20ns...1310,7µs.	LONG

Bemerkungen

Der Zeitabstand **delay** beginnt nach dem Einlesen eines Encoder-Werts und endet mit dem Einlesen des nächsten Encoder-Werts.

Nach dem Einschalten des Moduls wird als Voreinstellung der Wert 1250 verwendet, das entspricht 25µs.

Siehe auch

[P2_SSI_Mode](#), [P2_SSI_Read](#), [P2_SSI_Read2](#), [P2_SSI_Set_Bits](#), [P2_SSI_Set_Clock](#), [P2_SSI_Start](#), [P2_SSI_Status](#)

Gültig für

CNT-D Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_SSI_Set_Clock(1,50)    'CLK (Taktrate) = 1 MHz
P2_SSI_Set_Delay(1,1,400)'Zeitabstand 8µs für Decoder 1
P2_SSI_Set_Delay(1,2,200)'Zeitabstand 4µs für Decoder 2
P2_SSI_Set_Bits(1,1,10)   '10 Bits für Decoder 1
P2_SSI_Set_Bits(1,2,25)   '25 Bits für Decoder 2
P2_SSI_Mode(1,3)          'Continuous-Mode für beide Decoder
```

Event:

```
Par_1 = P2_SSI_Read(1,1) 'Positionswert (Decoder 1) auslesen
Par_2 = P2_SSI_Read(1,2) 'Positionswert (Decoder 2) auslesen
```

P2_SSI_Set_De- lay

P2_SSI_Start

P2_SSI_Start startet auf dem angegebenen Modul das Auslesen eines oder beider SSI-Decoder (nur im Modus single shot).

Syntax

```
#Include ADwinPro_All.Inc

P2_SSI_Start(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	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 **P2_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

[P2_SSI_Mode](#), [P2_SSI_Read](#), [P2_SSI_Read2](#), [P2_SSI_Set_Bits](#), [P2_SSI_Set_Clock](#), [P2_SSI_Set_Delay](#), [P2_SSI_Status](#)

Gültig für

CNT-D Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_SSI_Set_Clock(1,250) 'CLK (Taktrate) = 200 kHz
P2_SSI_Set_Delay(1,1,250) 'Wartezeit Decoder 1: 5 µs
P2_SSI_Set_Delay(1,2,1000) 'Wartezeit Decoder 2: 20 µs
P2_SSI_Mode(1,0) 'Single shot-Mode einstellen
                    '(beide Zähler)
P2_SSI_Set_Bits(1,1,23) 'Anzahl Bits = 23 (Decoder 1)
P2_SSI_Set_Bits(1,2,23) 'Anzahl Bits = 23 (Decoder 2)
```

Event:

```
P2_SSI_Start(1,3) 'Positionswert von Decoder 1 & 2 lesen
Do 'Für Decoder 1:
Until (P2_SSI_Status(1,1) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = P2_SSI_Read(1,1) 'Positionswert auslesen und anzeigen
Do 'Für Decoder 2:
Until (P2_SSI_Status(1,2) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = P2_SSI_Read(1,2) 'Positionswert auslesen und anzeigen
```



P2_SSI_Status liefert für einen bestimmten Decoder den aktuellen Lese-Status auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_SSI_Status(module, dcd_r_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dcd_r_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

[P2_SSI_Mode](#), [P2_SSI_Read](#), [P2_SSI_Read2](#), [P2_SSI_Set_Bits](#), [P2_SSI_Set_Clock](#), [P2_SSI_Set_Delay](#), [P2_SSI_Start](#)

Gültig für

CNT-D Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_SSI_Set_Clock(1,250) 'CLK (Taktrate) = 200 kHz
P2_SSI_Set_Delay(1,1,250) 'Wartezeit Decoder 1: 5 µs
P2_SSI_Set_Delay(1,2,1000) 'Wartezeit Decoder 2: 20 µs
P2_SSI_Mode(1,0) 'Single shot-Mode einstellen
                    '(beide Zähler)
P2_SSI_Set_Bits(1,1,23) 'Anzahl Bits = 23 (Decoder 1)
P2_SSI_Set_Bits(1,2,23) 'Anzahl Bits = 23 (Decoder 2)
```

Event:

```
P2_SSI_Start(1,3) 'Positionswert von Decoder 1 & 2 lesen
Do 'Für Decoder 1:
Until (P2_SSI_Status(1,1) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = P2_SSI_Read(1,1) 'Positionswert auslesen und anzeigen
Do 'Für Decoder 2:
Until (P2_SSI_Status(1,2) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = P2_SSI_Read(1,2) 'Positionswert auslesen und anzeigen
```

P2_SSI_Status

3.7 Pro II: PWM-Ausgänge

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit PWM-Ausgängen gelten:

- [P2_PWM_Enable](#) (Seite 189)
- [P2_PWM_Get_Status](#) (Seite 190)
- [P2_PWM_Init](#) (Seite 191)
- [P2_PWM_Latch](#) (Seite 193)
- [P2_PWM_Reset](#) (Seite 194)
- [P2_PWM_Standby_Value](#) (Seite 195)
- [P2_PWM_Write_Latch](#) (Seite 196)
- [P2_PWM_Write_Latch_Block](#) (Seite 197)

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 Modul die Adresse 1 eingestellt ist.



P2_PWM_Enable gibt einen oder mehrere PWM-Ausgänge zur Ausgabe frei oder sperrt sie.

Syntax

```
#Include ADwinPro_All.inc

P2_PWM_Enable(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster zur Auswahl der PWM-Ausgänge: Bit = 0: PWM-Ausgabe sperren. Bit = 1: PWM-Ausgabe freigeben.	LONG

Bit-Nr.	31...1 6	15	14	...	1	0
PWM-Ausgang	–	16	15	...	2	1

Bemerkungen

Wann die PWM-Ausgänge gesperrt werden – sofort oder nach dem nächsten Periodenende – hängt von der Einstellung ab, die mit **P2_PWM_Init** gemacht wurde (Parameter **mode**).

Siehe auch

[P2_PWM_Get_Status](#), [P2_PWM_Init](#), [P2_PWM_Latch](#), [P2_PWM_Reset](#), [P2_PWM_Standby_Value](#), [P2_PWM_Write_Latch_Block](#)

Gültig für

PWM-16(-I) Rev. E

Beispiel

siehe [P2_PWM_Init](#) (Seite 191)

P2_PWM_Enable

P2_PWM_Get_Status

P2_PWM_Get_Status liest den aktuellen Betriebsstatus für alle PWM-Ausgänge.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_PWM_Get_Status(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Statusbits für alle PWM-Ausgänge. Bit = 0: PWM-Ausgang ist gesperrt. Bit = 1: PWM-Ausgang ist freigegeben.	LONG

Bit-Nr.	31...1 6	15	14	...	1	0
PWM-Ausgang	–	16	15	...	2	1

Bemerkungen

- / -

Siehe auch

[P2_PWM_Enable](#), [P2_PWM_Init](#), [P2_PWM_Latch](#), [P2_PWM_Reset](#),
[P2_PWM_Standby_Value](#), [P2_PWM_Write_Latch_Block](#)

Gültig für

PWM-16(-I) Rev. E

Beispiel

- / -

P2_PWM_Init setzt die Voreinstellungen für den angegebenen PWM-Ausgang.

Syntax

```
#Include ADwinPro_All.inc

P2_PWM_Init(module, pwm_output, startdelay,
startvalue,
mode, count)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pwm_output	Nummer des PWM-Ausgabekanals (1...16).	LONG
startdelay	Startverzögerung in Einheiten von 10ns.	LONG
startvalue	Startpegel für die PWM-Ausgabe: 0: TTL-Pegel low. 1: TTL-Pegel high.	LONG
mode	Betriebsmodus des PWM-Ausgangs als Bitmuster; nur die Bits 0...2 sind relevant, andere Bits werden ignoriert. Bit 0: Übernahme einer neuen PW-Frequenz: • Bit =0: Übernahme bei Periodenende • Bit=1: Übernahme sofort. Bit 1: Anzahl der Pulse: • Bit =0: unendlich viele Pulse. • Bit=1: Anzahl der Pulse ist count . Bit 2: Anhalten bei Stopp-Befehl: • Bit =0: Anhalten bei Periodenende • Bit=1: Anhalten sofort.	LONG
count	Anzahl der Perioden (1...32768), die pro Ausgabszyklus ausgeführt werden.	LONG

Bemerkungen

- / -

Siehe auch

[P2_PWM_Enable](#), [P2_PWM_Get_Status](#), [P2_PWM_Latch](#), [P2_PWM_Reset](#), [P2_PWM_Standby_Value](#), [P2_PWM_Write_Latch_Block](#)

Gültig für

PWM-16(-I) Rev. E

P2_PWM_Init

Beispiel

```
#Include ADwinPro_All.inc
#Define module 4
#Define freq1 FPar_1
#Define freq2 FPar_2
#Define pw1 FPar_3
#Define pw2 FPar_4
Dim channel As Long

Init:
    freq1 = 1000           '1000 Hz
    freq2 = 2000           '2000 Hz
    pw1 = 50               '50 %
    pw2 = 70               '70 %
    P2_PWM_Reset(module,011b) 'stop channels 1 und 2

    For channel = 1 To 2
        P2_PWM_Init(module,channel,0,0,0,0)
    Next

    P2_PWM_Write_Latch(module,1,pw1,freq1)
    P2_PWM_Write_Latch(module,2,pw2,freq2)
    P2_PWM_Latch(module,11b)
    P2_PWM_Enable(module,011b) 'start output

Event:
    P2_PWM_Write_Latch(module,1,pw1,freq1)
    P2_PWM_Write_Latch(module,2,pw2,freq2)

    P2_PWM_Latch(module,11b)
```

P2_PWM_Latch aktiviert Frequenz und Tastverhältnis eines oder mehrerer PWM-Ausgänge für die PWM-Ausgabe.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_PWM_Latch(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster zur Auswahl der PWM-Ausgänge: Bit = 0: Kein Einfluss. Bit = 1: Latchen.	LONG

Bit-Nr.	31...1 6	15	14	...	1	0
PWM-Ausgang	–	16	15	...	2	1

Bemerkungen

Frequenz und Tastverhältnis werden mit **P2_PWM_Write_Latch** in das Latch-Register geschrieben. Erst mit **P2_PWM_Latch** werden die Werte aus dem Latch-Register ausgegeben.

Wann die Ausgabe mit den neuen Werten beginnt – sofort oder nach dem nächsten Periodenende – hängt von der Einstellung ab, die mit **P2_PWM_Init** gemacht wurde (Parameter [mode](#)).

Siehe auch

[P2_PWM_Enable](#), [P2_PWM_Get_Status](#), [P2_PWM_Init](#), [P2_PWM_Reset](#), [P2_PWM_Standby_Value](#), [P2_PWM_Write_Latch_Block](#)

Gültig für

PWM-16(-I) Rev. E

Beispiel

siehe [P2_PWM_Init](#) (Seite 191)

P2_PWM_Latch

P2_PWM_Reset

P2_PWM_Reset stoppt die Ausgabe auf einem oder mehreren Ausgängen sofort.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_PWM_Reset(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster zur Auswahl der PWM-Ausgänge: Bit = 0: Kein Einfluss Bit = 1: Ausgabe sofort stoppen	LONG

Bit-Nr.	31...1 6	15	14	...	1	0
PWM-Ausgang	–	16	15	...	2	1

Bemerkungen

Die Ausgabe wird auch dann sofort gestoppt, wenn mit **P2_PWM_Init** ein anderer Modus eingestellt ist.

Siehe auch

[P2_PWM_Enable](#), [P2_PWM_Get_Status](#), [P2_PWM_Init](#), [P2_PWM_Latch](#), [P2_PWM_Standby_Value](#), [P2_PWM_Write_Latch_Block](#)

Gültig für

PWM-16(-I) Rev. E

Beispiel

siehe [P2_PWM_Init](#) (Seite 191)

P2_PWM_Standby_Value setzt den Vorgabewert (TTL-Pegel) für einen PWM-Ausgang.

Syntax

```
#Include ADwinPro_All.inc

P2_PWM_Standby_Value (module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Vorgabewert für PWM-Ausgänge: Bit = 0: TTL-Pegel low Bit = 1: TTL-Pegel high	LONG

Bit-Nr.	31...1 6	15	14	...	1	0
PWM-Ausgang	–	16	15	...	2	1

Bemerkungen

Mit dem Befehl **P2_PWM_Standby_Value** können PWM-Ausgänge auch als einfache TTL-Ausgänge benutzt werden.

Wenn ein PWM-Ausgang nicht mit **P2_PWM_Enable** freigegeben ist, wird der Ausgang auf den Vorgabepegel aus **pattern** gesetzt. Der Vorgabepegel wird auch gesetzt, wenn der PWM-Ausgang stoppt. Nach dem Einschalten sind die Ausgänge zunächst auf TTL-Pegel low gesetzt.

Siehe auch

[P2_PWM_Enable](#), [P2_PWM_Get_Status](#), [P2_PWM_Init](#), [P2_PWM_Latch](#), [P2_PWM_Reset](#), [P2_PWM_Write_Latch_Block](#)

Gültig für

PWM-16(-I) Rev. E

Beispiel

- / -

P2_PWM_Standby_Value

P2_PWM_Write_Latch

P2_PWM_Write_Latch schreibt Frequenz und Tastverhältnis in das Latch-Register.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_PWM_Write_Latch(module, pwm_output, dutycycle, frequency)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>pwm_output</code>	Nummer des PWM-Ausgabekanals (1...16).	LONG
<code>dutycycle</code>	Tastverhältnis in Prozent zwischen 0.0 und 100.0 (die Werte 0.0 und 100.0 sind nicht zulässig).	FLOAT
<code>frequency</code>	Frequenz in Hertz: 0,025Hz ...500kHz.	FLOAT

Bemerkungen

Der Wert für `dutycycle` ist abhängig von der Einstellung des Parameters `startvalue` bei dem Befehl **P2_PWM_Init**:

- `startvalue` = 1: Geben Sie für `dutycycle` das Tastverhältnis an.
- `startvalue` = 0: Geben Sie für `dutycycle` das „inverse Tastverhältnis“ an: `dutycycle` = 100% - Tastverhältnis

Frequenz und Tastverhältnis werden mit **P2_PWM_Write_Latch** nur in das Latch-Register geschrieben. Erst mit **P2_PWM_Latch** werden die Werte für die PWM-Ausgabe aktiviert.

Die höchste Ausgangsfrequenz, bei der das Tastverhältnis noch in 1%-Schritten einstellbar ist, beträgt etwa 500kHz.

Wenn mehrere PWM-Ausgänge mit den gleichen Daten betrieben werden sollen, ist der Befehl **P2_PWM_Write_Latch_Block** schneller.

Siehe auch

[P2_PWM_Enable](#), [P2_PWM_Get_Status](#), [P2_PWM_Init](#), [P2_PWM_Latch](#), [P2_PWM_Reset](#), [P2_PWM_Standby_Value](#), [P2_PWM_Write_Latch_Block](#)

Gültig für

PWM-16(-I) Rev. E

Beispiel

siehe [P2_PWM_Init](#) (Seite 191)

P2_PWM_Write_Latch_Block schreibt Frequenz und Tastverhältnis für mehrere PWM-Ausgänge in das Latch-Register.

Syntax

```
#Include ADwinPro_All.inc

P2_PWM_Write_Latch_Block(module, dutycycle[],
    frequency[], channel_count)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dutycycle []	Tastverhältnis in Prozent zwischen 0.0 und 100.0 (die Werte 0.0 und 100.0 sind nicht zulässig).	FLOAT
frequency []	Frequenz in Hertz: 0,025Hz ...100MHz.	FLOAT
channel_ count	Anzahl (1...16) der PWM-Ausgänge, für die Ausgabedaten gesetzt werden.	LONG

Bemerkungen

Der Wert für **dutycycle** ist abhängig von der Einstellung des Parameters **startvalue** bei dem Befehl **P2_PWM_Init**:

- **startvalue** = 1: Geben Sie für **dutycycle** das Tastverhältnis an.
- **startvalue** = 0: Geben Sie für **dutycycle** das „inverse Tastverhältnis“ an: **dutycycle** = 100% - Tastverhältnis

Die Ausgabedaten gelten für die PWM-Ausgänge 1...**channel_count**.

Frequenz und Tastverhältnis werden mit **P2_PWM_Write_Latch_Block** nur in das Latch-Register geschrieben. Erst mit **P2_PWM_Latch** werden die Werte für die PWM-Ausgabe aktiviert.

Die höchste Ausgangsfrequenz, bei der das Tastverhältnis noch in 1%-Schritten einstellbar ist, beträgt 1 MHz.

Siehe auch

[P2_PWM_Enable](#), [P2_PWM_Get_Status](#), [P2_PWM_Init](#), [P2_PWM_Latch](#), [P2_PWM_Reset](#), [P2_PWM_Standby_Value](#)

Gültig für

PWM-16(-I) Rev. E

P2_PWM_Write_Latch_Block

Beispiel

```
#Include ADwinPro_All.inc
#Define module 4
#Define freq Data_1
#Define pw Data_2
Dim freq[16] As Float
Dim pw[16] As Float
Dim channel As Long

Init:
  For channel = 1 To 16
    freq[channel] = 1000 * channel 'channel 1: 1 kHz, channel
16: 16 KHz
    pw[channel] = 50 'all channels 50 %
  Next
  P2_PWM_Reset(module,0FFFFh)'stop all channels
  For channel = 1 To 16
    P2_PWM_Init(module,channel,0,0,0,0)
  Next
  P2_PWM_Write_Latch_Block(module, pw, freq, 3)
  P2_PWM_Latch(module,0FFFFh)
  P2_PWM_Enable(module,0FFFFh)'start output

Event:
  P2_PWM_Write_Latch_Block(module, pw, freq, 3)
  P2_PWM_Latch(module,11b)
```

3.8 Pro II: Temperaturmess-Module

Dieser Abschnitt beschreibt Befehle, die für Pro II-Module zur Temperaturmessung gelten:

- [P2_RTD_Channel_Config](#) (Seite 200)
- [P2_RTD_Config](#) (Seite 202)
- [P2_RTD_Convert](#) (Seite 203)
- [P2_RTD_Read](#) (Seite 204)
- [P2_RTD_Read8](#) (Seite 205)
- [P2_RTD_Start](#) (Seite 206)
- [P2_RTD_Status](#) (Seite 208)
- [P2_TC_Latch](#) (Seite 209)
- [P2_TC_Read_Latch](#) (Seite 210)
- [P2_TC_Read_Latch4](#) (Seite 212)
- [P2_TC_Read_Latch8](#) (Seite 214)
- [P2_TC_Set_Rate](#) (Seite 216)

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 Modul die Adresse 1 eingestellt ist.



P2_RTD_Channel_Config

P2_RTD_Channel_Config stellt den Temperatur-Messmodus für einen bestimmten Kanal auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_RTD_Channel_Config(module, channel, active, type,  
element, filter, sample_period)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1...8) des Messkanals.	LONG
active	Aktivierung des Messkanals: 0: Kanal wird deaktiviert. 1: Kanal wird aktiviert.	LONG
type	Messmethode (0...2): 0: 2-Leiter-Schaltung 1: 4-Leiter-Schaltung 2: 3-Leiter-Schaltung	LONG
element	Typ (0...2) des Thermoelements: 0: PT100 1: PT500 2: PT1000 3: Ni100	LONG
filter	Filterqualität (0...7); um periodische Störsignale zu filtern, werden mehrere Messungen zu einem Messwert gemittelt: 0: 1 Messung (keine Filterung). 1: 2 Messungen (= 2 ¹). 2: 4 Messungen (= 2 ²). 3: 8 Messungen (= 2 ³). 4: 16 Messungen (= 2 ⁴). 5: 32 Messungen (= 2 ⁵). 6: 64 Messungen (= 2 ⁶). 7: 128 Messungen (= 2 ⁷).	LONG
sample_period	Abtastintervall in Mikrosekunden (3...65535) zwischen 2 Messungen (nicht zwischen 2 Messwerten).	LONG

Bemerkungen

Nach dem Einschalten der Hardware sind alle Messkanäle deaktiviert. Aktive Messkanäle werden mit aufsteigender Kanalnummer im Messzyklus abgearbeitet.

Sie dürfen nur solche Kanäle aktivieren, an denen auch ein Temperatursensor angeschlossen ist. Anderenfalls können auf den anderen (an Sensoren angeschlossenen) Kanälen Störungen entstehen, die die Messwerte verfälschen.

Zu einem Messzyklus gehören alle aktivierten Messkanäle. Im Modus „continuous“ können Sie Messkanäle auch während eines Messzyklus aktivieren oder deaktivieren.

Die Messmethoden sind im Hardware-Handbuch erläutert.

Eine hohe Filterqualität **filter** verbessert die Genauigkeit des Messwerts, verlängert aber die Messdauer.

Mit einem geeigneten Wert für das Abtastintervall **sample_period**

können Sie den Filter für eine bestimmte Störfrequenz optimieren. Berechnen Sie dazu das Abtastintervall (in Mikrosekunden) wie folgt:

$$\text{sample_period} = 10^6 / (\text{Frequenz} \cdot 2^{\text{filter}})$$

Durch die Einstellung des Abtastintervalls werden alle Messungen für einen Messwert gleichmäßig über die Periodendauer der Störfrequenz verteilt, und die Störfrequenz wird aus dem Messwert herausgefiltert.

Die Messdauer T für einen Messwert (bei 2- und 4-Leiter-Messung) ist damit $T = \text{sample_period} \times 2^{\text{filter}}$.

Bei der 3-Leiter-Messung ist die Messdauer T doppelt so lang, denn hier müssen doppelt so viele Messungen durchgeführt werden.

Siehe auch

[P2_RTD_Config](#), [P2_RTD_Convert](#), [P2_RTD_Read](#), [P2_RTD_Read8](#),
[P2_RTD_Start](#), [P2_RTD_Status](#)

Gültig für

RTD-8 Rev. E

Beispiel

siehe [P2_RTD_Start](#)

P2_RTD_Config

P2_RTD_Config initialisiert die Temperaturmessung auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc  
P2_RTD_Config(module, mode, muxtime)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
mode	Betriebsmodus der Temperaturmessung: 0: Modus „single shot“, einfacher Messzyklus. 1: Modus „continuous“, regelmäßiger Messzyklus.	LONG
muxtime	Einschwingzeit beim Umschalten zwischen zwei Messkanälen: 0: Werkseinstellung (5000 = 100µs). 125...2 ³¹ : Zeitabstand in Einheiten von 20ns.	LONG

Bemerkungen

Sie konfigurieren den Betriebsmodus für jeden Temperatur-Messkanal separat mit **P2_RTD_Config_Channel**.

Zu einem Messzyklus gehören alle aktivierten Messkanäle. Im Modus „continuous“ können Sie Messkanäle auch während eines Messzyklus aktivieren oder deaktivieren.

Je länger die Messleitung zum Temperaturfühler ist, umso größer sollten Sie die Einschwingzeit wählen.

Die Dauer T_{Gesamt} eines Messzyklus ist die Summe aus den Messdauern der aktiven Messkanäle und der Einschwingzeit:

$$T_{\text{Gesamt}} = \sum_1^{\text{Kanäle}} (T_{\text{Kanal}} + \text{Einschwingzeit})$$

Siehe auch

[P2_RTD_Channel_Config](#), [P2_RTD_Convert](#), [P2_RTD_Read](#), [P2_RTD_Read8](#), [P2_RTD_Start](#), [P2_RTD_Status](#)

Gültig für

RTD-8 Rev. E

Beispiel

siehe [P2_RTD_Start](#)

P2_RTD_Convert berechnet aus dem Digitalwert eines Temperatur-Fühlers den zugehörigen Widerstand oder die Temperatur in Celsius oder Fahrenheit.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_RTD_Convert(dig_val, element, ret_type)
```

Parameter

dig_val	Digitalwert (24 Bit).	LONG
element	Typ (0...2) des Thermoelements: 0: PT100 1: PT500 2: PT1000 3: Ni100	LONG
ret_type	Typ des Rückgabewerts: 0: Widerstand in Ohm. 1: Temperatur in Grad Celsius. 2: Temperatur in Grad Fahrenheit.	LONG
ret_val	Widerstand in Ohm oder Temperatur in Grad Celsius oder in Grad Fahrenheit.	FLOAT

Bemerkungen

Für die Ermittlung der Temperaturwerte in °C und °F aus der Pt-Thermospannung wird die Grundwertreihe der IEC 751 (= EN 60751: 1990) verwendet, für Ni die IEC 43760. Der Messwert ist deswegen nur für Temperaturfühler richtig, die diesen Normen entsprechen.

Siehe auch

[P2_RTD_Channel_Config](#), [P2_RTD_Config](#), [P2_RTD_Read](#), [P2_RTD_Read8](#), [P2_RTD_Start](#), [P2_RTD_Status](#)

Gültig für

RTD-8 Rev. E

Beispiel

siehe [P2_RTD_Start](#)

P2_RTD_Convert

P2_RTD_Read

P2_RTD_Read gibt den aktuellen Temperaturmesswert eines bestimmten Kanals auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_RTD_Read(module, channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>channel</code>	Nummer (1...8) des Messkanals.	LONG
<code>ret_val</code>	Aktueller Temperaturmesswert (24 Bit) in Digits.	LONG

Bemerkungen

Im Modus „single shot“ darf ein Messwert erst gelesen werden, wenn der Messzyklus beendet ist (siehe **P2_RTD_Status**).

Siehe auch

[P2_RTD_Channel_Config](#), [P2_RTD_Config](#), [P2_RTD_Convert](#), [P2_RTD_Read8](#), [P2_RTD_Start](#), [P2_RTD_Status](#)

Gültig für

RTD-8 Rev. E

Beispiel

- / -

P2_RTD_Read8 gibt die aktuellen Temperaturmesswerte aller Kanäle auf dem angegebenen Modul in einem Feld zurück.

Syntax

```
#Include ADwinPro_All.inc

P2_RTD_Read8(module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY LONG
index	Indes des Felelements, ab dem die Messwerte gespeichert werden.	LONG

Bemerkungen

Es werden immer 8 Messwerte in dem Zielfeld gespeichert, auch wenn der Messzyklus aus weniger als 8 Messkanälen besteht. Die Messwerte werden mit aufsteigender Kanalnummer gespeichert.

Siehe auch

[P2_RTD_Channel_Config](#), [P2_RTD_Config](#), [P2_RTD_Convert](#), [P2_RTD_Read](#), [P2_RTD_Start](#), [P2_RTD_Status](#)

Gültig für

RTD-8 Rev. E

Beispiel

siehe [P2_RTD_Start](#)

P2_RTD_Read8

P2_RTD_Start

P2_RTD_Start startet den Temperatur-Messzyklus auf den angegebenen Modulen gleichzeitig.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_RTD_Start(module_pattern)
```

Parameter

module_
pattern Bitmuster für die Moduladressen, auf denen der Temperatur-Messzyklus starten soll: LONG
Bit = 0: Modul ignorieren.
Bit = 1: Temperaturmessung starten.

Bits in pattern	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

Bemerkungen

Bevor Sie den Temperatur-Messzyklus starten, müssen Sie den Betriebsmodus für die Module mit **P2_RTD_Config** und für die einzelnen Kanäle mit **P2_RTD_Config_Channel** festlegen.

Siehe auch

[P2_RTD_Channel_Config](#), [P2_RTD_Config](#), [P2_RTD_Convert](#), [P2_RTD_Read](#), [P2_RTD_Read8](#), [P2_RTD_Status](#)

Gültig für

RTD-8 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define module 2

Dim values24[8] As Long
Dim channel As Long
Dim i As Long
Dim run_state As Long
Dim status As Long

Init:
P2_RTD_Config(module, 0, 0) 'single shot mode
Rem use channels 1...6
For i = 1 To 6
    Rem do channel settings: 3 wire, PT100, 50 Hz filter
    P2_RTD_Channel_Config(module, i, 1, 2, 0, 7, 50)
Next
Processdelay=50000000
run_state = 0

Event:
SelectCase run_state
Case 0
    Rem start measurement cycle
    P2_RTD_start(Shift_Left(1, module-1))
    run_state = 1
Case 1
    Rem check for end of measurement cycle
    status = P2_RTD_status(module)
    If (status = 0) Then run_state = 2
Case 2
    Rem read measured values and prepare start of next cycle
    P2_RTD_read8(module, values24, 1) 'messwerte lesen
    For i = 1 To 6
        Rem convert measurement values
        fpar[i] = P2_RTD_convert(values24[i], 0, 1)
    Next
    run_state = 0
EndSelect
```

P2_RTD_Status

P2_RTD_Status gibt den Status eines einfachen Temperatur-Messzyklus auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_RTD_Status(module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>ret_val</code>	Status des Messzyklus: 0: Messzyklus ist beendet. 1: Messzyklus wird ausgeführt.	LONG

Bemerkungen

Der Befehl **P2_RTD_Status** ist nur sinnvoll für den Betriebsmodus „single shot“.

Siehe auch

[P2_RTD_Channel_Config](#), [P2_RTD_Config](#), [P2_RTD_Convert](#), [P2_RTD_Read](#), [P2_RTD_Read8](#), [P2_RTD_Start](#)

Gültig für

RTD-8 Rev. E

Beispiel

siehe [P2_RTD_Start](#)

P2_TC_Latch kopiert die an den Eingängen anliegenden Spannungswerte in die Latches.

Syntax

```
#Include ADwinPro_All.Inc

P2_TC_Latch(module)
```

Parameter

module Eingestellte Moduladresse (1...15).

LONG

Bemerkungen

Die Werte in den Latches werden erst beim Auslesen mit **...Read_Latch** in den gewünschten Rückgabewert (°C, °F, Thermospannung) umgerechnet.

Der Befehl **P2_Sync_All** hat auf dem Temperatur-Modul die gleiche Wirkung wie **P2_TC_Latch**.

Siehe auch

[P2_TC_Read_Latch](#), [P2_TC_Read_Latch4](#), [P2_TC_Read_Latch8](#),
[P2_TC_Set_Rate](#), [P2_Sync_All](#)

Gültig für

TC-8-ISO Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
Rem Set sampling rate to 27.5 Hz
P2_TC_Set_Rate(1,8)
```

Event:

```
Rem copy values to latches
P2_TC_Latch(1)
Rem Read temperature from channel 5, thermo couple K in °C
FPar_1 = P2_TC_Read_Latch(1,5,1,1)
```

P2_TC_Latch

P2_TC_Read_Latch

P2_TC_Read_Latch gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_TC_Read_Latch(module, channel,
                             tc_element, ret_type)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1...8) des zu lesenden Kanals.	LONG
tc_element	Typ des Thermoelements: -1: keine Konvertierung, also anliegende Thermospannung ohne Kaltstellenkompensation. 0: Thermoelement Typ J 1: Thermoelement Typ K 2: Thermoelement Typ N 3: Thermoelement Typ S 4: Thermoelement Typ T 5: Thermoelement Typ R 6: Thermoelement Typ E 7: Thermoelement Typ B	LONG
ret_type	Typ des Rückgabewerts ret_val : 0: Thermospannung in μV ; bei tc_element = -1 ohne Kaltstellenkompensation. 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
ret_val	Messwert des Kanals, abhängig von ret_type und tc_element .	FLOAT

Bemerkungen

Das Modul tastet die Kanäle regelmäßig ab (Abtastrate einstellen siehe **P2_TC_Set_Rate**). Die Anweisung **P2_TC_Read_Latch** gibt den jeweils zuletzt ermittelten Wert zurück.

Wenn Sie mehrere Kanäle mit der gleichen Konvertierung auslesen wollen, sind die Befehle **P2_TC_Read_Latch4** und **P2_TC_Read_Latch8** schneller.

Verwenden Sie für **tc_element** nach Möglichkeit eine Konstante. Wenn Sie eine Variable verwenden, benötigt der Befehl deutlich mehr Programmspeicher.

Wenn Sie **tc_element** = -1 angeben, wird die Thermospannung nicht korrigiert, d.h. das Modul führt weder eine Kaltstellenkompensation durch noch wird eine Thermoelement-Kennlinie berücksichtigt. Der Wertebereich für **ret_val** ist dann $-80000\mu\text{V} \dots 80000\mu\text{V}$.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen. Die Wertebereiche sind:

Typ	Temperaturbereich [$^{\circ}\text{C}$]	Temperaturbereich [$^{\circ}\text{F}$]	Thermospannung [μV]
B	250...1820	482...3329,6	291...13820

Typ	Temperaturbereich [°C]	Temperaturbereich [°F]	Thermospannung [µV]
E	-200...1000	-328...1832	-8825...76373
J	-210...1200	-346...2192	-8095...69553
K	-200...1372	-328...2501,6	-5891...54886
N	-200...1300	-328...2372	-3990...47513
R	-50...1768	-58...3214,4	-226...21101
S	-50...1768	-58...3214,4	-236...18693
T	-200...400	-454...752	-5603...20872

Siehe auch

[P2_TC_Latch](#), [P2_TC_Read_Latch4](#), [P2_TC_Read_Latch8](#), [P2_TC_Set_Rate](#)

Gültig für

TC-8-ISO Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
Rem Set sampling rate to 27.5 Hz
P2_TC_Set_Rate(1,8)
```

Event:

```
Rem copy values to latches
P2_TC_Latch(1)
Rem Read temperature from channel 5, thermo couple K in °C
FPar_1 = P2_TC_Read_Latch(1,5,1,1)
```

P2_TC_Read_Latch4

P2_TC_Read_Latch4 gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) der Kanäle 1...4 auf dem Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

P2_TC_Read_Latch4(module, tc_element, ret_type,
                  array[], array_start_index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tc_element	Typ des Thermoelements: -1: keine Konvertierung, also anliegende Thermospannung ohne Kaltstellenkompensation. 0: Thermoelement Typ J 1: Thermoelement Typ K 2: Thermoelement Typ N 3: Thermoelement Typ S 4: Thermoelement Typ T 5: Thermoelement Typ R 6: Thermoelement Typ E 7: Thermoelement Typ B	LONG
ret_type	Typ des Rückgabewerts ret_val : 0: Thermospannung in μV ; bei tc_element = -1 ohne Kaltstellenkompensation. 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
array[]	Zielfeld zum Speichern der Messwerte der Kanäle 1...4; Messwerte sind abhängig von ret_type und tc_element .	ARRAY FLOAT
array_start_index	Index des ersten Feldelements in array[] , das beschrieben wird.	LONG

Bemerkungen

Das Modul tastet die Kanäle regelmäßig ab (Abtastrate einstellen siehe **P2_TC_Set_Rate**). Die Anweisung **P2_TC_Read_Latch4** gibt die jeweils zuletzt ermittelten Werte der Kanäle 1...4 zurück.

Verwenden Sie für **tc_element** nach Möglichkeit eine Konstante. Wenn Sie eine Variable verwenden, benötigt der Befehl deutlich mehr Programmspeicher.

Wenn Sie **tc_element** = -1 angeben, wird die Thermospannung nicht korrigiert, d.h. das Modul führt weder eine Kaltstellenkompensation durch noch wird eine Thermoelement-Kennlinie berücksichtigt. Der Wertebereich für **ret_val** ist dann $-80000\mu\text{V}$... $80000\mu\text{V}$.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen. Die Wertebereiche sind:

Typ	Temperaturbereich[$^{\circ}\text{C}$]	Temperaturbereich[$^{\circ}\text{F}$]	Thermospannung[μV]
B	250...1820	482...3329,6	291...13820
E	-200...1000	-328...1832	-8825...76373

Typ	Temperaturbereich[°C]	Temperaturbereich[°F]	Thermospannung[μV]
J	-210...1200	-346...2192	-8095...69553
K	-200...1372	-328...2501,6	-5891...54886
N	-200...1300	-328...2372	-3990...47513
R	-50...1768	-58...3214,4	-226...21101
S	-50...1768	-58...3214,4	-236...18693
T	-200...400	-454...752	-5603...20872

Siehe auch

[P2_TC_Latch](#), [P2_TC_Read_Latch](#), [P2_TC_Read_Latch8](#), [P2_TC_Set_Rate](#)

Gültig für

TC-8-ISO Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
Dim cnt As Long
Dim values[1000] As Float
```

Init:

```
Rem Set sampling rate to 27.5 Hz
P2_TC_Set_Rate(1,8)
cnt = 1
```

Event:

```
Rem copy values to latches
P2_TC_Latch(1)
Rem Read temperature from channels 1..4, thermo couple J in °F
P2_TC_Read_Latch4(1,0,2,values,cnt)
Rem increase counter
cnt = cnt + 4 : If (cnt > 1000) Then cnt = 1
```

P2_TC_Read_Latch8

P2_TC_Read_Latch8 gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) der Kanäle 1...8 auf dem Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

P2_TC_Read_Latch8(module, tc_element, ret_type,
                  array[], array_start_index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tc_element	Typ des Thermoelements: -1: keine Konvertierung, also anliegende Thermospannung ohne Kaltstellenkompensation. 0: Thermoelement Typ J 1: Thermoelement Typ K 2: Thermoelement Typ N 3: Thermoelement Typ S 4: Thermoelement Typ T 5: Thermoelement Typ R 6: Thermoelement Typ E 7: Thermoelement Typ B	LONG
ret_type	Typ des Rückgabewerts ret_val : 0: Thermospannung in μV ; bei tc_element = -1 ohne Kaltstellenkompensation. 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
array[]	Zielfeld zum Speichern der Messwerte der Kanäle 1...4; Messwerte sind abhängig von ret_type und tc_element .	ARRAY FLOAT
array_start_index	Index des ersten Feldelements in array[] , das beschrieben wird.	LONG

Bemerkungen

Das Modul tastet die Kanäle regelmäßig ab (Abtastrate einstellen siehe **P2_TC_Set_Rate**). Die Anweisung **P2_TC_Read_Latch8** gibt die jeweils zuletzt ermittelten Werte zurück.

Verwenden Sie für **tc_element** nach Möglichkeit eine Konstante. Wenn Sie eine Variable verwenden, benötigt der Befehl deutlich mehr Programmspeicher.

Wenn Sie **tc_element** = -1 angeben, wird die Thermospannung nicht korrigiert, d.h. das Modul führt weder eine Kaltstellenkompensation durch noch wird eine Thermoelement-Kennlinie berücksichtigt. Der Wertebereich für **ret_val** ist dann $-80000\mu\text{V}$... $80000\mu\text{V}$.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen. Die Wertebereiche sind:

Typ	Temperaturbereich [$^{\circ}\text{C}$]	Temperaturbereich [$^{\circ}\text{F}$]	Thermospannung [μV]
B	250...1820	482...3329,6	291...13820
E	-200...1000	-328...1832	-8825...76373

Typ	Temperaturbereich [°C]	Temperaturbereich [°F]	Thermospannung [µV]
J	-210...1200	-346...2192	-8095...69553
K	-200...1372	-328...2501,6	-5891...54886
N	-200...1300	-328...2372	-3990...47513
R	-50...1768	-58...3214,4	-226...21101
S	-50...1768	-58...3214,4	-236...18693
T	-200...400	-454...752	-5603...20872

Siehe auch

[P2_TC_Latch](#), [P2_TC_Read_Latch](#), [P2_TC_Read_Latch4](#), [P2_TC_Set_Rate](#)

Gültig für

TC-8-ISO Rev. E

Beispiel

```
#Include ADwinPro_All.inc
Dim cnt As Long
Dim values[1000] As Float
```

Init:

```
Rem Set sampling rate to 27.5 Hz
P2_TC_Set_Rate(1,8)
cnt = 1
```

Event:

```
Rem copy values to latches
P2_TC_Latch(1)
Rem Read temperature from channels 1..8, thermo couple J in °F
P2_TC_Read_Latch8(1,0,2,values,cnt)
Rem increase counter
cnt = cnt + 8 : If (cnt > 1000) Then cnt = 1
```

P2_TC_Set_Rate

P2_TC_Set_Rate stellt die Abtastrate für das angegebene Modul ein.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_TC_Set_Rate(module, rate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

[P2_TC_Latch](#), [P2_TC_Read_Latch](#), [P2_TC_Read_Latch4](#), [P2_TC_Read_Latch8](#)

Gültig für

TC-8-ISO Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
  
Init:  
    Rem Set sampling rate to 27.5 Hz  
    P2_TC_Set_Rate(1,8)
```

3.9 Pro II: CAN-Bus

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit CAN-Bus gelten:

- [CAN_Msg](#) (Seite 218)
- [P2_CAN_Interrupt_Source](#) (Seite 220)
- [P2_CAN_Set_LED](#) (Seite 222)
- [P2_En_Interrupt](#) (Seite 223)
- [P2_En_Receive](#) (Seite 225)
- [P2_En_Transmit](#) (Seite 226)
- [P2_Get_CAN_Reg](#) (Seite 227)
- [P2_Init_CAN](#) (Seite 228)
- [P2_Read_Msg](#) (Seite 229)
- [P2_Read_Msg_Con](#) (Seite 231)
- [P2_Set_CAN_Baudrate](#) (Seite 233)
- [P2_Set_CAN_Reg](#) (Seite 234)
- [P2_Transmit](#) (Seite 235)

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 Modul die Adresse 1 eingestellt ist.



CAN_Msg

CAN_Msg ist ein eindimensionales Feld bestehend aus 9 Elementen, in dem die Message-Objekte (Nachrichten) des CAN-Busses beim Senden und Empfangen gespeichert sind oder werden.

Syntax

```
#INCLUDE ADwinPro_All.inc
```

```
CAN_Msg[n] = value
```

oder

```
ret_val = CAN_Msg[n]
```

Parameter

n	Nummer eines Feldelements (1... 9)	LONG
value	Berechnungs-Ausdruck, dessen Wert (0...256) in das Message-Objekt geschrieben 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 Werte in das Feld **CAN_Msg** [] ein, bevor Sie diese mit **P2_Transmit** übertragen.

Siehe auch

[P2_En_Receive](#), [P2_En_Transmit](#), [P2_Read_Msg](#), [P2_Transmit](#)

Gültig für

CAN-2 Rev. E

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 P2_Read_Msg)

```
#INCLUDE ADwinPro_All.inc
#DEFINE pi 3.14159265
DIM i AS LONG
```

Init:

```
P2_Init_CAN(1,1)          'CAN-Controller initialisieren
```

```
REM Initialisiere das Message-Objekt 6
REM zum Senden von CAN-Nachrichten mit dem Identifier 40
P2_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:

```
P2_Transmit(1,1,6)        'Message-Objekt 6 senden
```

P2_CAN_Interrupt_Source

P2_CAN_Interrupt_Source gibt zurück, welche CAN-Kanäle einen Interrupt ausgelöst haben.

Syntax

```
#INCLUDE ADwinPro_All.inc  
  
ret_val = P2_CAN_Interrupt_Source(module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>ret_val</code>	Bitmuster, das die Interrupt-Quelle angibt.	LONG

Bit-Nr.	31...2	1	0
CAN-Kanal	–	2	1

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn mit **P2_En_Interrupt** das Erzeugen von Event-Signalen (Interrupts) konfiguriert ist.

P2_CAN_Interrupt_Source arbeitet deutlich schneller als das Lesen des Interrupt-Registers auf einem CAN-Controller.

Nach dem Erzeugen eines Event-Signals müssen Sie die Nachricht des auslösenden Message-Objekts mit **P2_Read_Msg** lesen, damit der das Objekt wieder ein neues Event-Signal erzeugen kann. In der Zwischenzeit ignoriert der CAN-Controller eintreffende Nachrichten für dieses Message-Objekt.

Siehe auch

[P2_En_Interrupt](#), [P2_Init_CAN](#), [P2_Read_Msg](#)

Gültig für

CAN-2 Rev. E

Beispiel

```
#Include ADwinPro_All.INC
```

Init:

```
P2_Init_CAN(1,1)           'initialize channel 1
P2_En_Receive(1,1,3,1,0)   'configure msg objects 3 and 15
P2_En_Receive(1,1,15,385,0) 'for read
P2_En_Interrupt(1,1,3)     'configure msg objects 3 and 15
P2_En_Interrupt(1,1,15)    'for interrupt
P2_Event_Enable(1,1)       'enable event interrupt
```

Event:

```
Par_13 = P2_CAN_Interrupt_Source(1) 'check for interrupt
If (Par_13 And 01b = 1) Then
    Par_14 = CAN_Interrupt_Msg(1,1) 'get interrupting msg object
    Rem get msg object = enable new interrupt
    Par_15 = P2_Read_Msg(1,1,CAN_Interrupt_Msg(1,1))
EndIf
```

Function CAN_Interrupt_Msg(module,channel) As Long

```
REM read interrupt register and change value to objekt no.
CAN_Interrupt_Msg = P2_Get_CAN_Reg(module,channel,5fh)
If (CAN_Interrupt_Msg = 2) Then
    CAN_Interrupt_Msg = 15
Else
    CAN_Interrupt_Msg = CAN_Interrupt_Msg - 2
EndIf
```

EndFunction

Der Wert im Interrupt-Register entspricht einem der Message-Objekte nach folgendem Schema:

Wert	2	3	4	...	16
Nummer Message-Objekt	15	1	2	...	14

P2_CAN_Set_LED

P2_CAN_Set_LED schaltet die Zusatz-LED für einen CAN-Kanal auf dem Modul ein (mit Farbe) oder aus.

Syntax

```
#INCLUDE ADwinPro_All.inc  
  
P2_CAN_Set_LED(module, channel, led_col)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>channel</code>	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
<code>led_col</code>	Status und Farbe der Zusatz-LED: 0: LED aus. 1: LED ein, Farbe rot. 2: LED ein, Farbe grün. 3: LED ein, Farbe orange.	LONG

Bemerkungen

- / -

Siehe auch

[P2_Set_LED](#)

Gültig für

CAN-2 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc  
Init:  
    P2_Init_CAN(1,1)           'CAN-Controller initialisieren  
    P2_CAN_Set_LED(1,1,3)      'Setze LED 1 auf orange
```

P2_En_Interrupt konfiguriert ein bestimmtes Message-Objekt des angegebenen Moduls, so dass bei Eintreffen einer Nachricht ein Event-Signal (Interrupt) erzeugt wird.

Syntax

```
#INCLUDE ADwinPro_All.inc
```

```
P2_En_Interrupt (module, channel, msg_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

Bemerkungen

Ein erzeugtes Event-Signal wird nur dann an das Prozessormodul geleitet, wenn das Event-Signal mit **P2_EVENT_ENABLE** 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.

Nach dem Erzeugen eines Event-Signals müssen Sie die Nachricht des auslösenden Message-Objekts mit **P2_Read_Msg** lesen, damit der das Objekt wieder ein neues Event-Signal erzeugen kann. In der Zwischenzeit ignoriert der CAN-Controller eintreffende Nachrichten für dieses Message-Objekt.

Siehe auch

[P2_CAN_Interrupt_Source](#), [P2_En_Receive](#), [P2_Event_Enable](#), [P2_Event_Read](#), [P2_Get_CAN_Reg](#), [P2_Init_CAN](#)

Gültig für

CAN-2 Rev. E

P2_En_Interrupt



Beispiel

```
#Include ADwinPro_All.INC
```

Init:

```
P2_Init_CAN(1,1)           'initialize channel 1
P2_En_Receive(1,1,3,1,0)   'configure msg objects 3 and 15
P2_En_Receive(1,1,15,385,0) 'for read
P2_En_Interrupt(1,1,3)     'configure msg objects 3 and 15
P2_En_Interrupt(1,1,15)    'for interrupt
P2_Event_Enable(1,1)       'enable event interrupt
```

Event:

```
REM read interrupt register and change value to objekt no.
```

```
Par_13 = P2_Get_CAN_Reg(1,1,5fh)
```

```
If (Par_13 = 2) Then
```

```
    Par_13 = 15
```

```
Else
```

```
    Par_13 = Par_13 - 2
```

```
EndIf
```

```
Rem get msg object = enable new interrupt
```

```
Par_15 = P2_Read_Msg(1,1,Par_13)
```

Der Wert im Interrupt-Register entspricht einem der Message-Objekte nach folgendem Schema:

Wert	2	3	4	...	16
Nummer Message-Objekt	15	1	2	...	14

P2_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
```

```
P2_En_Receive(module, channel, msg_no, id, id_extend)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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](#), [P2_En_Transmit](#), [P2_Init_CAN](#), [P2_Read_Msg](#), [P2_Transmit](#)

Gültig für

CAN-2 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
```

Init:

```
REM Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
P2_Init_CAN(1,1)
REM Message-Objekt 1 freigeben für den Empfang von
REM CAN-Nachrichten mit dem 11 Bit-Identifier 200
P2_En_Receive(1,1,1,200,0)
```

P2_En_Receive

P2_En_Transmit

P2_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
```

```
P2_En_Transmit(module, channel, msg_no, id, id_extend)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>channel</code>	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
<code>msg_no</code>	Nummer (1...14) des Message-Objektes im CAN-Controller.	LONG
<code>id</code>	Identifier der Nachrichten, die in diesem Message-Objekt gesendet werden sollen ($0 \dots 2^{11}$ oder $0 \dots 2^{29}$).	LONG
<code>id_extend</code>	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](#), [P2_En_Receive](#), [P2_Init_CAN](#), [P2_Read_Msg](#), [P2_Transmit](#)

Gültig für

CAN-2 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
```

```
Init:
```

```
REM Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
```

```
P2_Init_CAN(1,1)
```

```
REM Message-Objekt 6 freigeben für das Senden von
```

```
REM CAN-Nachrichten mit dem 11 Bit-Identifier 40
```

```
P2_En_Transmit(1,1,6,40,0)
```

P2_Get_CAN_Reg gibt den Inhalt eines bestimmten Registers auf einem CAN-Controller des angegebenen Moduls zurück.

Syntax

```
#INCLUDE ADwinPro_All.inc

ret_val = P2_Get_CAN_Reg(module, channel, regno)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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. Beispiele sind:

- Adresse **00h**: Kontroll-Register
- Adresse **01h**: Status-Register
- Adresse **5fh**: Interrupt-Register

Siehe auch

[P2_En_Interrupt](#), [P2_Init_CAN](#), [P2_Set_CAN_Baudrate](#), [P2_Set_CAN_Reg](#)

Gültig für

CAN-2 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
Init:
    REM Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
    P2_Init_CAN(1,1)
    REM Das Kontroll-Register des CAN-Controller 1, Modul 1 auslesen
    Par_1 = P2_Get_CAN_Reg(1,1,0)
```

P2_Get_CAN_Reg

P2_Init_CAN

P2_Init_CAN initialisiert einen der CAN-Controller auf dem angegebenen Modul und bringt ihn in einen definierten Anfangszustand.

Syntax

```
#INCLUDE ADwinPro_All.inc  
P2_Init_CAN(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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 125kBit/s und muss deswegen auf jeden Fall mit **P2_Set_CAN_Baudrate** neu eingestellt werden.

Siehe auch

[P2_En_Receive](#), [P2_En_Transmit](#), [P2_Get_CAN_Reg](#), [P2_Set_CAN_Baudrate](#), [P2_Set_CAN_Reg](#)

Gültig für

CAN-2 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc  
Init:  
    REM Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1  
    P2_Init_CAN(1,1)
```


P2_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 = P2_Read_Msg(module, channel, msg_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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	≥-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](#), [P2_En_Receive](#), [P2_En_Transmit](#), [P2_Init_CAN](#), [P2_Transmit](#)

Gültig für

CAN-2 Rev. E

P2_Read_Msg

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 P2_Transmit).
#include ADwinPro_All.inc
DIM n AS LONG

Init:
  Par_1 = 0
  P2_Init_CAN(1,1)           'CAN-Controller 1 initialisieren
  P2_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 = P2_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
```

P2_Read_Msg_Con gibt zurück, ob eine neue Nachricht in einem Message-Objekt eines der CAN-Controller auf dem angegebenen Modul empfangen wurde.

Falls ja, wird die Nachricht in **CAN_Msg** gespeichert und der Identifier der Nachricht zurückgegeben.

Syntax

```
#INCLUDE ADwinPro_All.inc

ret_val = P2_Read_Msg_Con(module, channel, msg_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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	≥-1: Eine neue Nachricht ist eingegangen, der Wert ist der Identifier des Message-Objektes. -1: keine neue Nachricht vorhanden.	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](#), [P2_En_Receive](#), [P2_En_Transmit](#), [P2_Read_Msg](#)

Gültig für

CAN-2 Rev. E

P2_Read_Msg_Con

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 P2_Transmit).
#include ADwinPro_All.inc
DIM n AS LONG

Init:
  Par_1 = 0
  P2_Init_CAN(1,1)           'CAN-Controller 1 initialisieren
  P2_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 = P2_Read_Msg_Con(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
```

P2_Set_CAN_Baudrate stellt die angegebene Baudrate auf einem der Controller auf dem angegebenen Modul ein.

Syntax

```
#INCLUDE ADwinPro_All.inc

ret_val = P2_Set_CAN_Baudrate(module, channel, rate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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 (1MBit/s) überschrieben wird.

Siehe auch

[P2_Get_CAN_Reg](#), [P2_Init_CAN](#), [P2_Set_CAN_Reg](#)

Gültig für

CAN-2 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
DIM status as long
Init:
    P2_Init_CAN(1,1)          'Initialisierung des CAN-Controllers
    status = P2_Set_CAN_Baudrate(1,1,125000)'Baudrate = 125 kBit/s
```

P2_Set_CAN_Baudrate



P2_Set_CAN_Reg

P2_Set_CAN_Reg schreibt einen Wert in ein Register des gewählten CAN-Controllers auf dem angegebenen Modul.

Syntax

```
#INCLUDE ADwinPro_All.inc
```

```
P2_Set_CAN_Reg(module, channel, regno, value)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>channel</code>	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
<code>regno</code>	Register-Nummer (0...255) des CAN-Controllers	LONG
<code>value</code>	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

[P2_Init_CAN](#), [P2_Set_CAN_Baudrate](#), [P2_Get_CAN_Reg](#)

Gültig für

CAN-2 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc
```

```
Init:
```

```
P2_Init_CAN(1,1)           'Initialisierung des CAN-Controllers  
P2_Set_CAN_Reg(1,1,0,1)    'Setze Control-Register auf den Wert 1
```

P2_Transmit liest die Daten aus dem Feld **CAN_Msg** und sendet die Daten als Nachricht.

Die Nachricht wird gesendet, sobald das angegebene Message-Objekt in einem der CAN-Controller auf dem eingestellten Modul Zugriffsrecht auf den CAN-Bus hat.

Syntax

```
#INCLUDE ADwinPro_All.inc

P2_Transmit ( module , channel , msg_no )
```

Parameter

module	Eingestellte Moduladresse (1...15).	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.
- 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](#), [P2_En_Receive](#), [P2_En_Transmit](#), [P2_Read_Msg](#)

Gültig für

CAN-2 Rev. E

P2_Transmit

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 [P2_Read_Msg](#))

```
#INCLUDE ADwinPro_All.inc
#DEFINE pi 3.14159265
DIM i AS LONG

Init:
  P2_Init_CAN(1,1)          'CAN-Controller initialisieren

  REM Initialisiere das Message-Objekt 6
  REM zum Senden von CAN-Nachrichten mit dem Identifier 40
  P2_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:
  P2_Transmit(1,1,6)        'Message-Objekt 6 senden
```


3.10 Pro II: RSxxx

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit RSxxx-Schnittstellen gelten:

- [P2_Check_Shift_Reg](#) (Seite 238)
- [P2_Get_RS](#) (Seite 239)
- [P2_Read_FIFO](#) (Seite 240)
- [P2_RS_Init](#) (Seite 241)
- [P2_RS_Reset](#) (Seite 243)
- [P2_RS485_Send](#) (Seite 244)
- [P2_RS_Set_LED](#) (Seite 245)
- [P2_Set_RS](#) (Seite 246)
- [P2_Write_FIFO](#) (Seite 247)

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 Modul die Adresse 1 eingestellt ist.



P2_Check_Shift_Reg

P2_Check_Shift_Reg gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO des Kanals (auf dem angegebenen Modul) geschrieben wurden.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_Check_Shift_Reg(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Kanal, dessen Sende-Status gelesen werden soll (1, 2 oder 1...4).	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

[P2_Get_RS](#), [P2_Read_FIFO](#), [P2_RS_Init](#), [P2_RS_Reset](#), [P2_RS485_Send](#), [P2_Set_RS](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc  
  
Event:  
Rem ...  
Rem Prüft, ob Schnittstelle 1 noch Daten zu senden hat  
Par_1 = P2_Check_Shift_Reg(1,1)  
Rem ...
```

P2_Get_RS liest den Inhalt eines bestimmten Controller-Registers auf dem angegebenen Modul aus.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_Get_RS(module, reg_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

[P2_Check_Shift_Reg](#), [P2_Read_FIFO](#), [P2_RS_Init](#), [P2_RS_Reset](#),
[P2_RS485_Send](#), [P2_Set_RS](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

- / -

P2_Get_RS

P2_Read_FIFO

P2_Read_FIFO liest einen Wert aus dem Eingangs-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_Read_FIFO(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

[P2_Check_Shift_Reg](#), [P2_Get_RS](#), [P2_RS_Init](#), [P2_RS_Reset](#), [P2_RS485_Send](#), [P2_Set_RS](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_RS_Reset(1)  
P2_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 = P2_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 ab Seite 141.

P2_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
```

```
P2_RS_Init(module, channel, baud_rate, parity, bits,  
stop, handshake)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Kanal, der initialisiert werden soll (1, 2 oder 1...4).	LONG
baud_rate	Übertragungsgeschwindigkeit in Baud: RS232: 35 ... 115200 Baud. 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_bits	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 das Übertragungsprotokoll RS232 oder RS485 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 **P2_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 ... 2304000 Bit/s. Entsprechend der Spezifikation ist die RS232-Schnittstelle auf 115200 Bit/s beschränkt. Die folgende Liste zeigt einige übliche Baudraten.

P2_RS_Init



Übliche Baudraten [Bit/s]		
2304000	57600	2400
1152000	38400	1200
460800	19200	600
230400	9600	300
115200	4800	

Siehe auch

[P2_Check_Shift_Reg](#), [P2_Get_RS](#), [P2_Read_FIFO](#), [P2_RS_Reset](#),
[P2_RS485_Send](#), [P2_Set_RS](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

```
Init:
```

```
P2_RS_Reset(1)           'RS-Modul zurücksetzen  
P2_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 ab Seite 141.

P2_RS_Reset führt auf dem angegebenen Modul einen Hardware-Reset durch und löscht dabei die Einstellungen für alle Kanäle.

Syntax

```
#Include ADwinPro_All.Inc

P2_RS_Reset(module)
```

Parameter

module Eingestellte Moduladresse (1...15). 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 **P2_RS_Init** folgen, um den Controller zu initialisieren und die gewünschten Schnittstellen-Parameter einzustellen.

Siehe auch

[P2_Check_Shift_Reg](#), [P2_Get_RS](#), [P2_Read_FIFO](#), [P2_RS_Init](#), [P2_RS485_Send](#), [P2_Set_RS](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

```
Init:
```

```
  P2_RS_Reset(1)           'RS-Modul zurücksetzen
  P2_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 ab Seite 141.

P2_RS_Reset

P2_RS485_Send

P2_RS485_Send legt die Übertragungsrichtung für einen bestimmten Kanal des angegebenen Moduls fest.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_RS485_Send(module, channel, dir)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

[P2_Check_Shift_Reg](#), [P2_Get_RS](#), [P2_Read_FIFO](#), [P2_RS_Init](#), [P2_RS_Reset](#), [P2_Set_RS](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

Siehe Beispiel „RS485: Empfangen und senden“ auf Seite 252.

P2_RS_Set_LED schaltet die Zusatz-LED für eine RSxxx-Schnittstelle auf dem Modul ein (mit Farbe) oder aus.

Syntax

```
#INCLUDE ADwinPro_All.inc

P2_RS_Set_LED(module,channel,led_col)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) der RSxxx-Schnittstelle.	LONG
led_col	Status und Farbe der Zusatz-LED: 0: LED aus. 1: LED ein, Farbe rot. 2: LED ein, Farbe grün. 3: LED ein, Farbe orange.	LONG

Bemerkungen

- / -

Siehe auch

[P2_Set_LED](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc

Init:
    P2_RS_Set_LED(1,1,3)      'Setze LED 1 auf orange
```

P2_RS_Set_LED

P2_Set_RS

P2_Set_RS schreibt einen Wert in ein bestimmtes Register des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_Set_RS(module,register,value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
register	Nummer des zu beschreibenen 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

[P2_Check_Shift_Reg](#), [P2_Get_RS](#), [P2_Read_FIFO](#), [P2_RS_Init](#), [P2_RS_Reset](#), [P2_RS485_Send](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

- / -

P2_Write_FIFO schreibt einen Wert in den Sende-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_Write_FIFO(module,channel,value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	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

[P2_Check_Shift_Reg](#), [P2_Get_RS](#), [P2_Read_FIFO](#), [P2_RS_Init](#), [P2_RS_Reset](#), [P2_RS485_Send](#), [P2_Set_RS](#)

Gültig für

RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

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

Init:

```
P2_RS_Reset(1)
P2_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 = P2_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 ab Seite 141.

P2_Write_FIFO

3.11 Pro II: LIN-Bus-Schnittstelle

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit LIN-Bus-Schnittstellen gelten:

- [P2_LIN_Init](#) (Seite 249)
- [P2_LIN_Init_Write](#) (Seite 251)
- [P2_LIN_Init_Apply](#) (Seite 252)
- [P2_LIN_Reset](#) (Seite 253)
- [P2_LIN_Get_Version](#) (Seite 254)
- [P2_LIN_Read_Dat](#) (Seite 255)
- [P2_LIN_Msg_Write](#) (Seite 257)
- [P2_LIN_Msg_Transmit](#) (Seite 259)
- [P2_LIN_Set_LED](#) (Seite 260)

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 Modul die Adresse 1 eingestellt ist.



P2_LIN_Init initialisiert die Datenübertragung zwischen ADwin CPU und der LIN-Schnittstelle auf einem bestimmten Modul.

Syntax

```
#Include ADwinPro_All.Inc

REM define LIN settings array
Dim lin_datatable[150] As Long

ret_val = P2_LIN_Init(module, lin_datatable[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
lin_datatable	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und LIN-Modul aufnimmt.	ARRAY LONG
ret_val	Initialisierungsstatus: 0: Initialisierung erfolgreich. 1: Fehler: kein Pro II-Modul an dieser Adresse. 2: Fehler: keine LIN-Schnittstelle auf dem Modul.	LONG

Bemerkungen

P2_LIN_Init muss vor der Datenübertragung zwischen ADwin CPU und LIN-Modul ausgeführt werden. Der Befehl sollte im Abschnitt **Init:** stehen.

Bei der Initialisierung muss für jedes Modul ein Feld **lin_datatable[]** mit 150 Elementen angelegt werden.

Gültig für

LIN-2 Rev. E

Siehe auch

[P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#), [P2_LIN_Get_Version](#), [P2_LIN_Read_Dat](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

P2_LIN_Init

Beispiel

```
#Include ADwinPro_All.Inc

#Define mod_adr 4
Dim lin_datatable[150] As Long
Dim Data_1[20] As Long
Dim Data_2[20] As Long
Dim state As Long

Init:
    Processdelay = 30000000 '10 Hz
    'Initialize communication ADwin CPU - LIN module
    Par_1 = P2_LIN_Init(mod_adr, lin_datatable)
    If (Par_1 <> 0) Then Exit 'error
    Rem Interface 1, 9600 baud, LIN master
    P2_LIN_Init_Write(lin_datatable, 1, 9600, 1, 0)
    Rem Interface 2, 9600 baud, LIN slave
    P2_LIN_Init_Write(lin_datatable, 2, 9600, 0, 0)
    Rem message box 1 for receive on interface 2, msg id 1
    P2_LIN_Msg_Write(lin_datatable, 2, 1, 1, Data_2, 8, 0)
    P2_LIN_Init_Apply(lin_datatable)
    state = 1

Event:
    SelectCase state
    Case 1 'msg transmit
        Data_1[1] = 1
        Data_1[2] = 2
        Data_1[3] = 3
        Data_1[4] = 4
        Data_1[5] = 5
        Data_1[6] = 6
        Data_1[7] = 7
        Data_1[8] = 8
        Rem message box 1 for write on interface 1, msg id 1
        P2_LIN_Msg_Write(lin_datatable, 1, 1, 1, Data_1, 8, 1)
        Rem send header and message (interface 1 = LIN master)
        P2_LIN_Msg_Transmit(lin_datatable, 1, 1) 'msg tx
        state = 2
    Case 2 'check for msg receive, msg id 1
        P2_LIN_Read_Dat(lin_datatable, 2, 1, Data_2)
        If (Data_2[20] = 1) Then 'new msg rx
            Par_11 = Data_2[3] 'ID
            Par_12 = Data_2[4] 'Byte 1
            Par_13 = Data_2[5]
            Par_14 = Data_2[6]
            Par_15 = Data_2[7]
            Par_16 = Data_2[8]
            Par_17 = Data_2[9]
            Par_18 = Data_2[10]
            Par_19 = Data_2[11] 'Byte 8
            Par_20 = Data_2[12] 'checksum
            Par_21 = Data_2[13] 'length
            Inc Par_10
            state = 1 'new Msg tx
        EndIf
    EndSelect
```

P2_LIN_Init_Write legt Baudrate und Betriebsmodus für eine bestimmte LIN-Schnittstelle fest.

Syntax

```
#Include ADwinPro_All.Inc

P2_LIN_Init_Write(lin_datatable[], channel,
    baudrate, mode, chs_enh)
```

Parameter

lin_datatable	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und LIN-Modul enthält..	ARRAY LONG
channel	Nummer (1...2) der LIN-Schnittstelle.	LONG
baudrate	Baudrate (2400...19200), mit der die Schnittstelle betrieben wird.	LONG
mode	Betriebsmodus der LIN-Schnittstelle: 0: Betrieb als LIN Slave-Teilnehmer. 1: Betrieb als LIN Master-Teilnehmer.	LONG
chs_enh	Version der Prüfsumme: 0: classic. 1. enhanced.	LONG

Bemerkungen

Die Einstellung muss für jede verwendete LIN-Schnittstelle separat vorgenommen werden. Die Einstellungsdaten müssen anschließend mit **P2_LIN_Init_Apply** aktiviert werden, damit sie auf dem LIN-Bus wirksam werden.

Wenn **P2_LIN_Init_Write** nicht ausgeführt wird, arbeiten alle Kanäle mit folgender Standard-Einstellung:

- Baudrate 9600 Baud
- Betrieb als Slave
- Prüfsummenversion „classic“.

Gültig für

LIN-2 Rev. E

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#), [P2_LIN_Get_Version](#), [P2_LIN_Read_Dat](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Beispiel

siehe [P2_LIN_Init](#)

P2_LIN_Init_Write

P2_LIN_Init_Apply

P2_LIN_Init_Apply aktiviert die mit **P2_LIN_Init_Write** eingestellten Betriebsdaten für alle LIN-Schnittstellen.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_LIN_Init_Apply(lin_datatable[])
```

Parameter

lin_ Feld, das Einstellungen für die Datenübertragung **ARRAY**
datatable zwischen ADwin CPU und LIN-Modul enthält.. **LONG**
[]

Bemerkungen

P2_LIN_Init_Apply verändert die Einstellungsdaten der LIN-Schnittstellen nicht.

Gültig für

LIN-2 Rev. E

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Reset](#), [P2_LIN_Get_Version](#),
[P2_LIN_Read_Dat](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Beispiel

siehe [P2_LIN_Init](#)

P2_LIN_Reset setzt alle LIN-Kanäle zurück, und zwar entweder alle Einstellungen (Einschaltzustand) oder nur die LIN-internen Zähler.

Syntax

```
#Include ADwinPro_All.Inc

P2_LIN_Reset(lin_datatable[], reset_mode)
```

Parameter

lin_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	zwischen ADwin CPU und LIN-Modul enthält..	LONG
[]		
reset_	Einstellungen, die Kanäle zurückgesetzt werden:	LONG
mode	1: alle Einstellungen auf Einschaltzustand. 2: nur LIN-Zähler auf 0 rücksetzen	

Bemerkungen

Beim Rücksetzen auf den Einschaltzustand werden folgende Einstellungen für alle LIN-Kanäle gesetzt:

- Baudrate 9600 Baud
- Betrieb als Slave
- interne Zähler (Nachrichten, Timeout) auf 0.

Gültig für

LIN-2 Rev. E

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Get_Version](#), [P2_LIN_Read_Dat](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Beispiel

- / -

P2_LIN_Reset

P2_LIN_Get_Version

P2_LIN_Get_Version gibt die Versionsnummer der LIN-Schnittstelle zurück.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_LIN_Get_Version(lin_datatable[])
```

Parameter

lin_	Feld, das Einstellungen für die Datenübertragung	ARRAY
datatable	zwischen ADwin CPU und LIN-Modul enthält..	LONG
[]		
ret_val	Versionsnummer (0...9999) der LIN-Schnittstelle.	LONG

Bemerkungen

Die Versionsnummer wird nur benötigt, wenn Sie Fragen zur Programmierung des LIN-Bus an unseren Support haben.

Gültig für

LIN-2 Rev. E

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#),
[P2_LIN_Read_Dat](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Beispiel

- / -

P2_LIN_Read_Dat liest die Daten einer Messagebox oder den Status einer LIN-Schnittstelle und schreibt sie in ein Feld.

Syntax

```
#Include ADwinPro_All.Inc

P2_LIN_Read_Dat(lin_datatable[], channel, membox,
                rd_dat[])
```

Parameter

lin_datatable	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und LIN-Modul enthält..	ARRAY LONG
channel	Nummer (1...2) der LIN-Schnittstelle.	LONG
membox	Kennzahl zur Auswahl einer Messagebox oder der Schnittstellenstatus: 1...64: Nummer der LIN-Messagebox, deren Daten gelesen werden. 65: Schnittstellenstatus lesen.	LONG
rd_dat	Zielfeld, in dem die Daten gespeichert werden.	ARRAY LONG

Bemerkungen

Wenn Sie eine ungültige Kennzahl **membox** angeben, kann das Modul in einen instabilen Zustand geraten. In diesem Fall müssen Sie das Pro-System aus- und wieder einschalten.

Eine Messagebox / der Schnittstellenstatus besteht aus 20 Elementen. Diese Daten werden in den Feldelementen **rd_dat[1]** ... **rd_dat[20]** gespeichert. Die folgende Tabelle zeigt die Bedeutung für eine Messagebox und für den Schnittstellenstatus:

Element	Messagebox (membox =1...64)	Schnittstellenstatus (membox =65)
1	Schnittstellennummer (1...2)	Schnittstellennr. (1...2)
2	Nummer (1...64) der Messagebox	Nummer (65) der Messagebox
3	Identifizier (0...63) der Messagebox	Zuletzt übertragene Kennzahl (0...63).
4	Datenbytes 1...8	Anzahl der übertragenen Nachrichten seit dem Start des Moduls.
5...11		—
12	Prüfsumme für Datenbytes	—
13	Anzahl gültiger Datenbytes	—
14	Sendestatus der Messagebox: 0: receive 1: send	—
15	Zeit für LIN-Header in µs.	—
16	Zeit für LIN-Antwort in µs.	—
17	Gesamtzeit einer LIN-Nachricht in µs (= 15+16).	—

Element	Messagebox (membox=1...64)	Schnittstellenstatus (membox=65)
18	Pausenzeit in μ s zwischen 2 Datenbytes. Standard: 0.	–
19	Anzahl aufgetretener Timeout-Fehler für diese Nachricht.	–
20	1: neue Nachricht wurde empfangen. -1: keine neue Nachricht -2: Nachricht wird gerade empfangen. -3: Nachricht mit Timeout-Fehler	–

Gültig für

LIN-2 Rev. E

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#),
[P2_LIN_Get_Version](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Beispiel

siehe [P2_LIN_Init](#)

P2_LIN_Msg_Write konfiguriert eine Messagebox in einer LIN-Schnittstelle zum Senden oder Empfangen.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_LIN_Msg_Write(lin_datatable[], channel, membox,  
                msg_id, msg_dat[], msg_len, msg_send)
```

Parameter

lin_datatable []	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und LIN-Modul enthält..	ARRAY LONG
channel	Nummer (1...2) der LIN-Schnittstelle.	LONG
membox	Nummer (1...64) der konfigurierten LIN-Messagebox.	LONG
msg_id	Identifizier (0...63) der Messagebox.	LONG
msg_dat []	Quellfeld, aus dem Datenbytes in die Messagebox übertragen werden.	LONG
msg_len	Anzahl (1...8) der zu übertragenden Datenbytes aus msg_dat [].	LONG
msg_send	Sendestatus der Messagebox: 0: receive (Empfangen) 1: send (Senden)	LONG

Bemerkungen

Das Feld **msg_dat** [] muss auf mind. 8 Elemente dimensioniert sein. Bei einer Messagebox mit dem Sendestatus „Empfangen“ werden die Daten des Felds **msg_dat** [] nicht verwendet.

Nach dem Konfigurieren einer Messagebox ist diese sofort auf dem LIN-Bus aktiv, d.h. es können Daten empfangen oder gesendet werden.

Wenn Sie bei einer Messagebox mit dem Sendestatus „send“ die zu übertragenden Datenbytes ändern wollen, verwenden Sie ebenfalls **P2_LIN_Msg_Write**.

Die Messagebox eines Master-Teilnehmers verhält sich anders als die eines Slave-Teilnehmers:

- **Master-Teilnehmer, Senden:** Der Master sendet sowohl den Header (siehe **P2_LIN_Msg_Transmit**) als auch gleich anschließend das Datenpaket der Messagebox.
- **Master-Teilnehmer, Empfangen:** Der Master sendet den Header (siehe **P2_LIN_Msg_Transmit**) auf den LIN-Bus und wartet auf die Antwort des passenden Slaves. Das empfangene Datenpaket wird in die Messagebox eingetragen.
- **Slave-Teilnehmer, Senden:** Der Slave wartet, bis der Master den Header mit dem zur Messagebox passenden Identifizier sendet. Erst dann sendet der Slave-Teilnehmer das Datenpaket.
- **Slave-Teilnehmer, Empfangen:** Der Slave wartet, bis der Master den Header mit dem zur Messagebox passenden Identifizier sendet, empfängt anschließend das Datenpaket und trägt es in die Messagebox ein.

Gültig für

LIN-2 Rev. E

P2_LIN_Msg_Write

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#),
[P2_LIN_Get_Version](#), [P2_LIN_Read_Dat](#), [P2_LIN_Msg_Transmit](#)

Beispiel

siehe [P2_LIN_Init](#)

P2_LIN_Msg_Transmit sendet einen Header auf den LIN-Bus. Nur gültig im Betriebsmodus LIN Master.

Syntax

```
#Include ADwinPro_All.Inc

P2_LIN_Msg_Transmit(lin_datatable[], channel,
                    membox)
```

Parameter

lin_datatable	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und LIN-Modul enthält..	ARRAY LONG
channel	Nummer (1...2) der LIN-Schnittstelle.	LONG
membox	Nummer (1...64) der LIN-Messagebox, die übertragen werden soll.	LONG

Bemerkungen

P2_LIN_Msg_Transmit hat nur im Betriebsmodus LIN-Master eine Funktion (siehe **P2_LIN_Init_Writ**), weil nur der LIN Master einen Header senden kann.

Durch das Senden eines Headers auf dem LIN-Bus reagieren diejenigen Teilnehmer am LIN-Bus (das kann auch der Master selbst sein), die eine Messagebox mit dem Identifier **membox** verwalten, indem sie ein Datenpaket senden oder ein auf dem Bus anstehendes Datenpaket empfangen.

Gültig für

LIN-2 Rev. E

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#), [P2_LIN_Get_Version](#), [P2_LIN_Read_Dat](#), [P2_LIN_Msg_Write](#)

Beispiel

siehe [P2_LIN_Init](#)

P2_LIN_Msg_Transmit

P2_LIN_Set_LED

P2_LIN_Set_LED schaltet die Zusatz-LED für eine LIN-Schnittstelle auf dem Modul ein (mit Farbe) oder aus.

Syntax

```
#INCLUDE ADwinPro_All.inc  
  
P2_LIN_Set_LED(module, channel, led_col)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) der LIN-Schnittstelle.	LONG
led_col	Status und Farbe der Zusatz-LED: 0: LED aus. 1: LED ein, Farbe rot. 2: LED ein, Farbe grün. 3: LED ein, Farbe orange.	LONG

Bemerkungen

- / -

Siehe auch

[P2_Set_LED](#)

Gültig für

LIN-2 Rev. E

Beispiel

```
#INCLUDE ADwinPro_All.inc  
Dim lin_datatable[150] As Long  
Dim ret_val As Long  
  
Init:  
    Rem LIN-Controller initialisieren  
    ret_val = P2_LIN_Init(1, lin_datatable)  
    P2_LIN_Set_LED(1, 1, 3)      'Setze LED 1 auf orange
```


3.12 Pro II: Profibus-Schnittstelle

Dieser Abschnitt enthält Befehle zum Ansprechen der Profibus-Schnittstellen auf *ADwin-Pro II*.

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit LIN-Bus-Schnittstellen gelten:

- [P2_Init_Profibus](#) (Seite 262)
- [P2_Run_Profibus](#) (Seite 264)

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

P2_Init_Profibus

P2_Init_Profibus initialisiert den Profibus-Slave.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Init_Profibus(module, dev_adr,
    in_mod_cnt, in_mod_type, out_mod_cnt,
    out_mod_type, work_arr[], info[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dev_adr	Slave Knotenadresse (1...125) auf dem Profibus.	LONG
in_mod_cnt	Anzahl (0...76) der Eingangs-Datenbereiche im Profibus-Slave. Die max. Anzahl hängt von der Kennzahl in_mod_type ab.	LONG
in_mod_type	Kennzahl (1...3, 16) für die Länge der Eingangs-Datenbereiche: 1: 1 Byte; max. Wert für in_mod_cnt : 76. 2: 2 Byte; max. Wert für in_mod_cnt : 38. 3: 4 Byte; max. Wert für in_mod_cnt : 19. 16: 8 Byte; max. Wert für in_mod_cnt : 9.	LONG
out_mod_cnt	Anzahl (0...76) der Ausgangs-Datenbereiche im Profibus-Slave. Die max. Anzahl hängt von der Kennzahl out_mod_type ab.	LONG
out_mod_type	Kennzahl (1...3, 16) für die Länge der Ausgangs-Datenbereiche: 1: 1 Byte; max. Wert für out_mod_type : 76. 2: 2 Byte; max. Wert für out_mod_type : 38. 3: 4 Byte; max. Wert für out_mod_type : 19. 16: 8 Byte; max. Wert für out_mod_type : 9.	LONG
work_arr[]	Feld, das Daten für den Betrieb des Profibus-Slave aufnimmt. Das Feld muss mind. 200 Elemente haben.	ARRAY LONG
info[]	Feld, das Daten über den Profibus-Slave enthält. Das Feld muss mind. 10 Elemente haben. Die Elemente info[1] und info[2] enthalten den Produktionstyp des Profibus-Slave: info[1]=1, info[2]=4	ARRAY LONG
ret_val	Status der Initialisierung: 0: kein Fehler. ≠0: Fehler; bitte melden Sie sich beim Support von Jäger Messtechnik.	LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem Profibus-Slave ausgeführt werden.

P2_Init_Profibus soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit (etwa 2-3 Sekunden) dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und ADwin-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.



Stationsadresse, Anzahl und Größe der Datenbereiche müssen die gleichen sein wie bei der Projektierung des Profibus. Die Modullänge wird bei der Projektierung auch in Worten angegeben: 1 Wort = 2 Byte.

Gültig für

Profi-SL Rev. E

Siehe auch

[P2_Run_Profibus](#)

Beispiel

```
#Include ADwinPro_All.INC
#Define module 5          'module address
#Define node 2            'slave node address
#Define info Data_1       'info array
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[76] As Long At DM_Local
Dim in_arr[76] As Long At DM_Local
Dim conf_arr[200] As Long At DM_Local
Dim info[10] As Long At DM_Local
Dim i As Long
Dim error As Long

Init:
    Processdelay = 3000000    'set to 100 Hz
    For i = 1 To 10          'initialize info array
        info[i] = 0
    Next i
    Rem initialize profibus interface: 38 input data areas of 2 byte
    Rem and 76 output data bytes of 1 Byte
    error = P2_Init_Profibus(module,node,38,2,76,1,conf_arr,info)
    If (error <> 0) Then      'initialization error
        Par_1 = error
        Exit
    EndIf

Event:
    Rem set data in out_arr[] to be transferred
    For i = 1 To 76
        out_arr[i] = (out_arr[i] + i) And 0FFh
    Next i

    Rem send and read data (output areas: 76; input areas: 38)
    error = P2_Run_Profibus(module,out_arr,76,in_arr,38,conf_arr)
    error = error And 7h
    Par_2 = error

    Rem here the received data in in_arr[] can be processed
```

P2_Run_Profibus

P2_Run_Profibus tauscht Daten mit dem Profibus-Slave aus.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Run_Profibus(module, out_pd_arr[],
    out_pd_arr_len, in_pd_arr[], in_pd_arr_len,
    work_arr[])
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>out_pd_arr[]</code>	Feld, aus dem der Profibus-Slave Daten liest und auf den der Profibus schreibt.	ARRAY LONG
<code>out_pd_arr_len</code>	Anzahl der Ausgangsbereiche (1...76), deren Datenbytes aus dem Feld <code>out_pd_arr[]</code> gelesen werden. Die Anzahl darf nicht größer sein als in <code>out_mod_cnt</code> bei P2_INIT_PROFIBUS angegeben wurde.	LONG
<code>in_pd_arr[]</code>	Feld, in das der Profibus-Slave Daten schreibt, die vom Profibus gelesen werden.	ARRAY LONG
<code>in_pd_arr_len</code>	Anzahl der Eingangsbereiche (1...76), deren Datenbytes im Feld <code>in_pd_arr[]</code> zurückgegeben werden. Die Anzahl darf nicht größer sein als in <code>in_mod_cnt</code> bei P2_INIT_PROFIBUS angegeben wurde.	LONG
<code>work_arr[]</code>	Feld, das Daten für den Betrieb des Profibus-Slave enthält, siehe P2_INIT_PROFIBUS .	ARRAY LONG
<code>ret_val</code>	Bitmuster, das den Betriebszustand des Profibus-Slave angibt. Von Bedeutung sind die Bits 0...2: 100b : Slave ist aktiv und arbeitet korrekt. 010b : Profibus nicht aktiv, Slave im Wartezustand. 110b, 111b : Fehler.	LONG

Bemerkungen

Run_Profibus soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioren Prozess würde die Kommunikation zwischen PC und ADwin-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Jedes Feldelement in `out_pd_arr[]` und `in_pd_arr[]` enthält nur 1 Datenbyte (Bits 0...7). Datenbereiche aus mehreren Bytes werden in entsprechend vielen, aufeinander folgenden Feldelementen abgelegt. Beispiel: 5 Datenbereiche mit je 4 Byte Länge werden in $5 \times 4 = 20$ Feldelementen gespeichert.

Gültig für

Profi-SL Rev. E

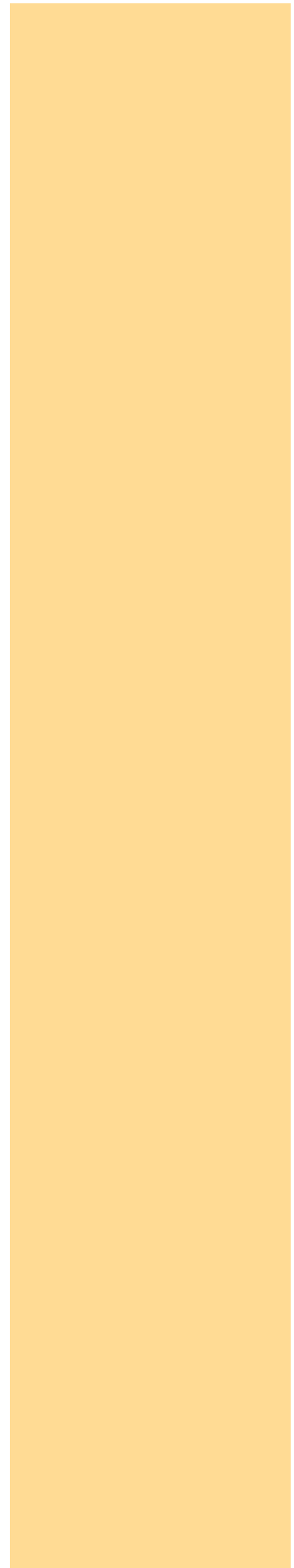
Siehe auch

[P2_Init_Profibus](#)



Beispiel

siehe [P2_Init_Profibus](#)



3.13 Pro II: EtherCAT-Schnittstelle

Dieser Abschnitt enthält Befehle zum Ansprechen der EtherCAT-Schnittstellen auf *ADwin-Pro II*.

- [P2_ECAT_Get_Version](#) (Seite 267)
- [P2_ECAT_Get_State](#) (Seite 268)
- [P2_ECAT_Init](#) (Seite 269)
- [P2_ECAT_Read_Data_16L](#) (Seite 271)
- [P2_ECAT_Write_Data_16L](#) (Seite 272)

P2_ECAT_Get_Version gibt die Version der EtherCAT-Schnittstelle zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_ECAT_Get_Version(ecat_datatable[])
```

Parameter

<code>ecat_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und EtherCAT-Modul enthält.	ARRAY LONG
<code>ret_val</code>	Versionsnummer der EtherCAT-Schnittstelle, zu lesen in hexadezimaler Schreibweise.	LONG

Bemerkungen

Die Versionsnummer wird nur benötigt, wenn Sie Fragen zur Programmierung des EtherCAT-Bus an unseren Support haben.

Die Versionsnummer (in hexadezimaler Schreibweise) ist fünfstellig, beispielsweise 10000h; die erste Stelle ist die Hauptversionsnummer.

Gültig für

EtherCAT-SL Rev. E

Siehe auch

[P2_ECAT_Init](#)

Beispiel

siehe [P2_ECAT_Init](#)

P2_ECAT_Get_Version

P2_ECAT_Get_State

P2_ECAT_Get_State gibt den Status der EtherCAT-Schnittstelle zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_ECAT_Get_State(ecat_datatable[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ecat_datatable []	Feld, das Einstellungen für die Datenübertragung zwischen <i>ADwin</i> CPU und EtherCAT-Modul enthält.	ARRAY LONG
ret_val	Betriebszustand der EtherCAT-Schnittstelle: 1: Betriebszustand Init. 2: Betriebszustand PreOp. 3: Betriebszustand Boot. 4: Betriebszustand SafeOp. 8: Betriebszustand Op.	LONG

Bemerkungen

Der Betriebszustand Boot wird in *ADbasic* nicht unterstützt.

Gültig für

EtherCAT-SL Rev. E

Siehe auch

[P2_ECAT_Init](#)

Beispiel

siehe [P2_ECAT_Init](#)

P2_ECAT_Init initialisiert den EtherCAT-Slave.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_ECAT_Init(module, ecat_datatable[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ecat_datatable []	Feld, das Einstellungen für die Datenübertragung zwischen <i>ADwin</i> CPU und EtherCAT-Modul aufnimmt.	ARRAY LONG
ret_val	Status der Initialisierung: 0: kein Fehler. 1: ungültiges Modul.	LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem EtherCAT-Slave ausgeführt werden.

Gültig für

EtherCAT-SL Rev. E

Siehe auch

[P2_ECAT_Get_Version](#)

P2_ECAT_Init

Beispiel

```
#Include ADwinPro_All.INC

#Define ECAT_MODULE_ADDRESS 5
#Define ecat_inputs Data_1
#Define ecat_outputs Data_2

Dim ecat_inputs[16] As Long
Dim ecat_outputs[16] As Long
Dim ecat_comtable[150] As Long
Dim i As Long
Dim ret As Long

Init:
    Processdelay = 300000 ' 1kHz
    Rem initialize data transfer T11 <-> TiCo
    Par_1 = P2_ECAT_Init(ECAT_MODULE_ADDRESS, ecat_comtable)
    Par_2 = P2_ECAT_Get_Version(ecat_comtable) '10000h

    For i = 1 To 16
        ecat_inputs[i] = 0
    Next
    For i = 1 To 16
        ecat_outputs[i] = i
    Next
    Par_11 = 0
    Par_12 = 0

Event:
    ret = P2_ECAT_Get_State(ecat_comtable)

    If (ret = 8) Then 'operational mode
        ret = P2_ECAT_Write_Data_16L(ecat_comtable, ecat_outputs)
        If (ret = 0) Then 'writing data was o.k.
            Inc Par_11 'increase write counter
        EndIf

        ret = P2_ECAT_Read_Data_16L(ecat_comtable, ecat_inputs)
        If (ret = 0) Then 'reading data was o.k.
            Inc Par_12 'increase read counter
        EndIf
    EndIf
```

P2_ECAT_Read_Data_16L liest 16 Long-Werte vom EtherCAT-Slave und gibt sie in einem Feld zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_ECAT_Read_Data_16L(ecat_datatable[],
    ecat_inputs[])
```

Parameter

<code>ecat_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und EtherCAT-Modul enthält.	ARRAY LONG
<code>ecat_inputs[]</code>	Feld, in das der EtherCAT-Slave Daten schreibt.	ARRAY LONG
<code>ret_val</code>	Lese-Status: 0: Lesen war erfolgreich. ≠0: Fehler beim Lesen der Daten.	LONG

Bemerkungen

- / -

Gültig für

EtherCAT-SL Rev. E

Siehe auch

[P2_ECAT_Init](#)

Beispiel

siehe [P2_ECAT_Init](#)

P2_ECAT_Read_Data_16L

P2_ECAT_Write_Data_16L

P2_ECAT_Write_Data_16L schreibt 16 Long-Werte aus einem Feld zum EtherCAT-Slave.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_ECAT_Write_Data_16L(ecat_datatable[],  
    ecat_outputs[])
```

Parameter

ecat_datatable []	Feld, das Einstellungen für die Datenübertragung zwischen <i>ADwin</i> CPU und EtherCAT-Modul enthält.	ARRAY LONG
ecat_outputs []	Feld, aus dem der EtherCAT-Slave Daten liest und auf den EtherCAT-Bus schreibt.	ARRAY LONG
ret_val	Schreib-Status: 0: Schreiben war erfolgreich. ≠0: Fehler beim Schreiben der Daten.	LONG

Bemerkungen

- / -

Gültig für

EtherCAT-SL Rev. E

Siehe auch

[P2_ECAT_Init](#)

Beispiel

siehe [P2_ECAT_Init](#)

3.14 Pro II: Flexray

Dieser Abschnitt beschreibt Befehle, die für Pro II-Module mit Flexray-Schnittstellen gelten:

- [P2_FlexRay_Get_Version](#) (Seite 274)
- [P2_FlexRay_Init](#) (Seite 275)
- [P2_FlexRay_Read_Word](#) (Seite 277)
- [P2_FlexRay_Reset](#) (Seite 278)
- [P2_FlexRay_Set_LED](#) (Seite 279)
- [P2_FlexRay_Write_Word](#) (Seite 280)

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

P2_FlexRay_Get_Version

P2_FlexRay_Get_Version gibt die Versionsnummer der FlexRay-Schnittstelle zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_FlexRay_Get_Version(  
    fr_datatable[], status)
```

Parameter

fr_	Feld mit Einstellungen für den Zugriff der ADwin	ARRAY
datatable	CPU auf das FlexRay-Modul.	LONG
[]		
status	Status des Zugriffs auf das FlexRay-Modul: 0: Zugriff war erfolgreich. 1: Fehler: FlexRay-Controller war beschäftigt. 2: Fehler: FlexRay-Controller hat nicht rechtzeitig reagiert.	LONG
ret_val	Versionsnummer der FlexRay-Firmware.	LONG

Bemerkungen

Die Versionsnummer wird nur benötigt, wenn Sie Fragen zur Programmierung des FlexRay-Moduls an unseren Support haben.

Je 4 hexadezimale Ziffern stehen für die Versionsnummern des High-Level- und des Low-Level-Treibers. Beispielsweise steht **01030205h** für die Versionen 1.3 (high level) und 2.5 (low level).

Siehe auch

[P2_FlexRay_Init](#)

Gültig für

FlexRay-2 Rev. E

Beispiel

- / -

P2_FlexRay_Init initialisiert die Datenübertragung zwischen *ADwin* CPU und einer FlexRay-Schnittstelle auf einem bestimmten Modul.

Syntax

```
#Include ADwinPro_All.inc

REM communication settings array of FlexRay module
Dim fr_datatable[150] As Long

P2_FlexRay_Init(module, fr_datatable[], status)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
fr_ datatable []	Feld für Einstellungen für den Zugriff der <i>ADwin</i> CPU auf das FlexRay-Modul.	ARRAY LONG
status	Status des Zugriffs auf das FlexRay-Modul: 0: Zugriff war erfolgreich. 1: Fehler: Kein Pro II-Modul an dieser Adresse. 2: Fehler: Keine FlexRay-Schnittstelle auf dem Modul.	LONG

Bemerkungen

Vor der Initialisierung muss für jedes Modul ein Feld **fr_datatable[]** mit 150 Elementen angelegt werden.

P2_FlexRay_Init muss vor der Datenübertragung zwischen *ADwin* CPU und FlexRay-Modul ausgeführt werden. Der Befehl sollte im Abschnitt **Init:** stehen.

Siehe auch

[P2_FlexRay_Read_Word](#), [P2_FlexRay_Reset](#), [P2_FlexRay_Set_LED](#), [P2_FlexRay_Write_Word](#)

Gültig für

FlexRay-2 Rev. E

P2_FlexRay_Init

Beispiel

```
#Include ADwinPro_All.inc
Dim fr_datatable[150] As Long
Dim value, status As Long

Init:
    Rem initialize communication to the FlexRay controller
    P2_FlexRay_Init(1, fr_datatable, status)
    If (status <> 0) Then Exit

Event:
    Rem read address 210h from controller 1
    value = P2_FlexRay_Read_Word(fr_datatable,1,210h,status)
    If (status <> 0) Then End
    If (value = 15) Then
        Rem read address 220h from controller 1
        value = P2_FlexRay_Read_Word(fr_datatable,1,220h,status)
    Else
        Rem write value to address 192h of controller 1
        P2_FlexRay_Write_Word(fr_datatable,1,192h,value,status)
    EndIf

Finish:
    If (status <> 0) Then
        Rem set Par_1 to error number
        Par_1 = status
    EndIf
```


P2_FlexRay_Read_Word gibt einen 16 Bit-Wert aus einem FlexRay-Controller auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_FlexRay_Read_Word(fr_datatable[],
                               controller, address, status)
```

Parameter

fr_	Feld mit Einstellungen für den Zugriff der ADwin	ARRAY
datatable	CPU auf das FlexRay-Modul.	LONG
[]		
controller	Nummer (1, 2) des FlexRay-Controllers.	LONG
r		
address	Adresse (0...1FFeh) auf dem FlexRay-Controller, deren Wert gelesen wird. Geben Sie die Adresse im 2-Byte-Alignment an.	LONG
status	Status des Zugriffs auf das FlexRay-Modul: 0: Zugriff war erfolgreich. 1: Fehler: FlexRay-Controller war beschäftigt. 2: Fehler: FlexRay-Controller hat nicht rechtzeitig reagiert.	LONG
ret_val	Inhalt (16 Bit-Wert) der Adresse im FlexRay-Controller.	LONG

Bemerkungen

- / -

Siehe auch

[P2_FlexRay_Init](#), [P2_FlexRay_Reset](#), [P2_FlexRay_Write_Word](#)

Gültig für

FlexRay-2 Rev. E

Beispiel

siehe [P2_FlexRay_Init](#)

P2_FlexRay_Read_Word

P2_FlexRay_Reset

P2_FlexRay_Reset setzt einen FlexRay-Controller auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_FlexRay_Reset(fr_datatable[], controller, status)
```

Parameter

fr_	Feld mit Einstellungen für den Zugriff der ADwin	ARRAY
datatable	CPU auf das FlexRay-Modul.	LONG
[]		
controller	Nummer (1, 2) des FlexRay-Controllers.	LONG
r		
status	Status des Zugriffs auf das FlexRay-Modul: 0: Zugriff war erfolgreich. 1: Fehler: FlexRay-Controller war beschäftigt. 2: Fehler: FlexRay-Controller hat nicht rechtzeitig reagiert.	LONG

Bemerkungen

- / -

Siehe auch

[P2_FlexRay_Init](#), [P2_FlexRay_Read_Word](#), [P2_FlexRay_Write_Word](#)

Gültig für

FlexRay-2 Rev. E

Beispiel

siehe [P2_FlexRay_Init](#)

P2_FlexRay_Set_LED schaltet eine Kanal-LED eines FlexRay-Controllers auf dem angegebenen Modul ein oder aus.

Syntax

```
#Include ADwinPro_All.inc

P2_FlexRay_Set_LED(fr_datatable[], controller,
                  channel, value, status)
```

Parameter

fr_	Feld mit Einstellungen für den Zugriff der ADwin	ARRAY
datatable	CPU auf das FlexRay-Modul.	LONG
controller	Nummer (1, 2) des FlexRay-Controllers.	LONG
channel	Nummer (1, 2) des FlexRay-Kanals. 1: Kanal A. 2: Kanal B.	LONG
value	Status der LED: 0: LED aus. 1: LED ein.	LONG
status	Status des Zugriffs auf das FlexRay-Modul: 0: Zugriff war erfolgreich. 1: Fehler: FlexRay-Controller war beschäftigt. 2: Fehler: FlexRay-Controller hat nicht rechtzeitig reagiert.	LONG

Bemerkungen

- / -

Siehe auch

[P2_FlexRay_Init](#)

Gültig für

FlexRay-2 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
Dim fr_datatable[150] As Long
Dim status As Long

Init:
Rem FlexRay-Controller initialisieren
P2_FlexRay_Init(1, fr_datatable, status)
Rem LED für Kanal 2, Controller 1 einschalten
P2_FlexRay_Set_LED(fr_datatable, 1, 2, 1, status)
```

P2_FlexRay_Set_LED

P2_FlexRay_Write_Word

P2_FlexRay_Write_Word schreibt einen 16 Bit-Wert an eine Adresse in einem FlexRay-Controller des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_FlexRay_Write_Word(fr_datatable[],  
                      controller, address, value, status)
```

Parameter

fr_	Feld mit Einstellungen für den Zugriff der ADwin	ARRAY
datatable	CPU auf das FlexRay-Modul.	LONG
[]		
controller	Nummer (1, 2) des FlexRay-Controllers.	LONG
r		
address	Adresse (0...1FFeh) auf dem FlexRay-Controller, in die ein Wert geschrieben wird. Geben Sie die Adresse im 2-Byte-Alignment an.	LONG
value	16 Bit-Wert, der in die Adresse im FlexRay-Controller geschrieben wird.	LONG
status	Status des Zugriffs auf das FlexRay-Modul: 0: Zugriff war erfolgreich. 1: Fehler: FlexRay-Controller war beschäftigt. 2: Fehler: FlexRay-Controller hat nicht rechtzeitig reagiert.	LONG

Bemerkungen

- / -

Siehe auch

[P2_FlexRay_Init](#), [P2_FlexRay_Read_Word](#), [P2_FlexRay_Reset](#)

Gültig für

FlexRay-2 Rev. E

Beispiel

siehe [P2_FlexRay_Init](#)

4 Programmbeispiele

4.1 Kontinuierliche Messwertwandlung

Die Module Pro II AIn-F-4/14 Rev. E und Pro II AIn-F-4/14 Rev. E ermöglichen eine sehr schnelle, kontinuierliche Messwert-Wandlung. Parallel zur Wandlung müssen jedoch die Daten gelesen (und ggf. verarbeitet) werden.

Im folgenden finden Sie Beispiele für kontinuierliche Datenwandlung.

1 Kanal wandeln

Benötigt wird ein Modul Pro II AIn-F-x/14 Rev. E mit Moduladresse 1 und ein Analogsignal am Eingang 1 des Moduls.

Das Programm <PROII-F-x-14-CONT-1CH.BAS> führt auf dem Eingangskanal 1 eine kontinuierliche Burst-Messreihe mit einer Taktrate von 25MHz aus. Der Speicherbereich für die Messwerte fasst 20000 Werte.

Während der laufenden Messreihe werden Messwerte in das Feld **Data_1** eingelesen (ein FIFO-Feld ist nicht möglich). Der Parallelbetrieb erfordert eine Abstimmung von Wandeln und Auslesen. Hierzu wird der Speicherbereich in 4 Bereiche zu 5000 Messwerten eingeteilt, und es wird nur der Bereich ausgelesen, der gerade fertig beschrieben wurde.

Ebenso ist eine Abstimmung zwischen der Wandlungsrate und der Leserate erforderlich. Der Prozesszyklus wird mit **Processdelay** = 20000 (= 15kHz) so gewählt, dass die Leserate im Mittel ein Mehrfaches der Wandlungsrate 25MHz beträgt:

$$15\text{kHz} \cdot 5000 = \text{Leserate} 75\text{MHz} > \text{Wandlungsrate } 25\text{MHz}$$

Beachten Sie bitte bei einer Veränderung des Programmbeispiels: Wenn die Bearbeitungszeit des Abschnitts **Event**: länger wird, beispielsweise durch eine Verarbeitung der Messwerte, genügt die Leserate vielleicht nicht mehr zum Lesen der gewandelten Messwerte. In diesem Fall gehen Messwerte ver-



loren, weil sie schneller überschrieben als gelesen werden, und Sie müssen die Abstimmung von Wandlungsrate und Leserate neu durchführen.

```
#Include ADwinPro_All.inc 'include file
#Define module 1          'module no.
#Define d1 Data_1         'holds values of channel 1
#Define mem_idx Par_1     'mem position of last written value
#Define max_val 20000     'no. of values
#Define seg1 max_val/8     'end of segment 1
#Define seg2 max_val/4     'end of segment 2
#Define seg3 max_val/8*3   'end of segment 3
#Define blk max_val/4     'read block size

Dim d1[max_val] as long   'destination array
Dim pattern as long       'bit pattern to address one module
Dim segment as long       'segment that is currently written

Init:
Rem 1 channel continuous, mem for max_val values, 25 MHz
P2_Burst_Init(module,1,0,max_val,2,010b)
pattern = Shift_Left(1,module-1)'address this module only
P2_Burst_Start(pattern)
segment = 1                'start with memory segment 1
Processdelay = 20000      'cycle time 66.6 µs -> 15 kHz

Event:
mem_idx = P2_Burst_Read_Index(module) 'get current mem index
If (segment = 1) then      'read 1. segment
    If ((mem_idx > seg1) And (mem_idx < seg3)) then
        Rem memory index is in segments 2 or 3: read segment 1
        P2_Burst_Read_Unpacked1(module,blk,0,Data_1,1,3)
        segment = 2
    endif
endif

If (segment = 2) then      'read 2. segment
    If (mem_idx > seg2) then
        Rem memory index is in segments 3 or 4: read segment 2
        P2_Burst_Read_Unpacked1(module,blk,seg1,Data_1,blk+1,3)
        segment = 3
    endif
endif

If (segment = 3) then      'read 3. segment
    If ((mem_idx > seg3) Or (mem_idx < seg1)) then
        Rem memory index is in segments 4 or 1: read segment 3
        P2_Burst_Read_Unpacked1(module,blk,seg2,Data_1,blk*2+1,3)
        segment = 4
    Endif
endif

If (segment = 4) then      'read 4. segment
    If (mem_idx < seg2) then
        Rem memory index is in segments 1 or 2: read segment 4
        P2_Burst_Read_Unpacked1(module,blk,seg3,Data_1,blk*3+1,3)
        segment = 1
    endif
endif
```

Befehlsübersichten

A.1 Alphabetische Befehlsübersicht

A

P2_ADC · 34
P2_ADC24 · 35
P2_ADCF · 88
P2_ADCF24 · 89
P2_ADCF_Mode · 90
P2_ADCF_Read_Limit · 93
P2_ADCF_Read_Min_Max4 · 96
P2_ADCF_Read_Min_Max8 · 98
P2_ADCF_Reset_Min_Max · 95
P2_ADCF_Set_Limit · 94
P2_ADC_Read_Limit · 36
P2_ADC_Set_Limit · 38

B

P2_Burst_CRead_Unpacked1 · 57
P2_Burst_CRead_Unpacked2 · 59
P2_Burst_CRead_Unpacked4 · 61
P2_Burst_CRead_Unpacked8 · 63
P2_Burst_Init · 65
P2_Burst_Read · 70
P2_Burst_Read_Index · 68
P2_Burst_Read_Unpacked1 · 72
P2_Burst_Read_Unpacked2 · 74
P2_Burst_Read_Unpacked4 · 76
P2_Burst_Read_Unpacked8 · 78
P2_Burst_Reset · 80
P2_Burst_Start · 82
P2_Burst_Status · 83
P2_Burst_Stop · 85

C

P2_CAN_Interrupt_Source · 220
CAN_Msg · 218
P2_CAN_Set_LED · 222
P2_Check_LED · 4
P2_Check_Shift_Reg · 238
P2_Cnt_Clear · 162
P2_Cnt_Enable · 163
P2_Cnt_Get_PW · 166
P2_Cnt_Get_PW_HL · 167
P2_Cnt_Get_Status · 165
P2_Cnt_Latch · 168
P2_Cnt_Mode · 169
P2_Cnt_PW_Enable · 164
P2_Cnt_PW_Latch · 171
P2_Cnt_Read · 172
P2_Cnt_Read4 · 173
P2_Cnt_Read_Int_Register · 174
P2_Cnt_Read_Latch · 176
P2_Cnt_Read_Latch4 · 177
P2_Cnt_Sync_Latch · 178
CPU_Digin (T11) · 11
CPU_Digout · 12

CPU_Dig_IO_Config · 13
CPU_Event_Config · 14

D

P2_DAC · 116
P2_DAC4 · 117
P2_DAC4_Packed · 118
P2_DAC8 · 120
P2_DAC8_Packed · 121
P2_Digin_Edge · 136
P2_Digin_FIFO_Clear · 137
P2_Digin_FIFO_Enable · 138
P2_Digin_FIFO_Full · 140
P2_Digin_FIFO_Read · 141
P2_Digin_Fifo_Read_Fast · 143
P2_Digin_FIFO_Read_Timer · 145
P2_Digin_Long · 146
P2_Digout · 147
P2_Digout_Bits · 148
P2_Digout_FIFO_Clear · 149
P2_Digout_FIFO_Empty · 150
P2_Digout_FIFO_Enable · 151
P2_Digout_FIFO_Read_Timer · 152
P2_Digout_FIFO_Start · 153
P2_Digout_FIFO_Write · 154
P2_Digout_Long · 156
P2_Digout_Reset · 157
P2_Digout_Set · 158
P2_DigProg · 159
P2_Dig_FIFO_Mode · 131
P2_Dig_Latch · 133
P2_Dig_Read_Latch · 134
P2_Dig_Write_Latch · 135

E

P2_En_Interrupt · 223
P2_En_Receive · 225
P2_En_Transmit · 226
P2_Event2_Config · 8
P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10

F

P2_FlexRay_Get_Version · 274
P2_FlexRay_Init · 275
P2_FlexRay_Read_Word · 277
P2_FlexRay_Reset · 278
P2_FlexRay_Set_LED · 279
P2_FlexRay_Write_Word · 280

G

P2_Get_CAN_Reg · 227
P2_Get_Digout_Long · 160

P2_MIO_Get_Digout_Long · 31
P2_Get_RS · 239

I

P2_Init_CAN · 228

L

P2_LIN_Get_Version · 254
P2_LIN_Init · 249
P2_LIN_Init_Apply · 252
P2_LIN_Init_Write · 251
P2_LIN_Msg_Transmit · 259
P2_LIN_Msg_Write · 257
P2_LIN_Read_Dat · 255
P2_LIN_Reset · 253
P2_LIN_Set_LED · 260

M

P2_MIO_Digin_Long · 27
P2_MIO_Digout · 28
P2_MIO_Digout_Long · 29
P2_MIO_DigProg · 30
P2_MIO_Dig_Latch · 24
P2_MIO_Dig_Read_Latch · 25
P2_MIO_Dig_Write_Latch · 26

P

P2_ECAT_Get_State · 268
P2_ECAT_Get_Version · 267
P2_ECAT_Init · 269
P2_ECAT_Read_Data_16L · 271
P2_ECAT_Write_Data_16L · 272
P2_Init_Profibus · 262
P2_PWM_Latch · 193
P2_PWM_Reset · 194
P2_PWM_Standby_Value · 195
P2_PWM_Write_Latch · 196
P2_PWM_Write_Latch_Block · 197
P2_Run_Profibus · 264
P2_TC_Latch · 209
P2_TC_Read_Latch · 210
P2_TC_Read_Latch4 · 212
P2_TC_Read_Latch8 · 214
P2_TC_Set_Rate · 216
P2_PWM_Enable · 189
P2_PWM_Get_Status · 190
P2_PWM_Init · 191
P2_PWM_Latch · 193
P2_PWM_Reset · 194
P2_PWM_Standby_Value · 195
P2_PWM_Write_Latch · 196
P2_PWM_Write_Latch_Block · 197

R

P2_Read_ADC · 39
P2_Read_ADC24 · 40
P2_Read_ADCF · 100
P2_Read_ADCF24 · 101

P2_Read_ADCF32 · 108
P2_Read_ADCF4 · 102
P2_Read_ADCF4_24B · 103
P2_Read_ADCF4_Packed · 106
P2_Read_ADCF8 · 104
P2_Read_ADCF8_24B · 105
P2_Read_ADCF8_Packed · 107
P2_Read_ADCF_SConv · 109
P2_Read_ADCF_SConv24 · 110
P2_Read_ADCF_SConv32 · 111
P2_Read_ADC_SConv · 41
P2_Read_ADC_SConv24 · 42
P2_Read_FIFO · 240
P2_Read_Msg · 229
P2_Read_Msg_Con · 231
P2_RS485_Send · 244
P2_RS_Init · 241
P2_RS_Reset · 243
P2_RS_Set_LED · 245
P2_RTD_Channel_Config · 200
P2_RTD_Config · 202
P2_RTD_Convert · 203
P2_RTD_Read · 204
P2_RTD_Read8 · 205
P2_RTD_Start · 206
P2_RTD_Status · 208

S

P2_Seq_Init · 44
P2_Seq_Read · 47
P2_Seq_Read24 · 48
P2_Seq_Read_Packed · 50
P2_Seq_Start · 51
P2_Seq_Wait · 52
P2_Set_Average_Filter · 87
P2_Set_CAN_Baudrate · 233
P2_Set_CAN_Reg · 234
P2_Set_Gain · 112
P2_Set_LED · 5
P2_Set_Mux · 53
P2_Set_RS · 246
P2_SE_Diff · 43
P2_SSI_Mode · 180
P2_SSI_Read · 181
P2_SSI_Read2 · 182
P2_SSI_Set_Bits · 183
P2_SSI_Set_Clock · 184
P2_SSI_Set_Delay · 185
P2_SSI_Start · 186
P2_SSI_Status · 187
P2_Start_Conv · 54
P2_Start_ConvF · 113
P2_Start_DAC · 123
P2_Sync_All · 15
P2_Sync_Enable · 17
P2_Sync_Mode · 19
P2_Sync_Stat · 21

T

[P2_Transmit · 235](#)

W

[P2_Wait_EOC · 55](#)

[P2_Wait_EOCF · 114](#)

[P2_Wait_Mux · 56](#)

[P2_Write_DAC · 124](#)

[P2_Write_DAC32 · 129](#)

[P2_Write_DAC4 · 125](#)

[P2_Write_DAC4_Packed · 126](#)

[P2_Write_DAC8 · 127](#)

[P2_Write_DAC8_Packed · 128](#)

[P2_Write_Fifo · 247](#)

A.2 Befehlsübersicht nach Modulen

Sie finden die Befehlsübersichten der Module auf den folgenden Seiten. Die Module sind alphabetisch nach Namen sortiert:

Modulname	Seite
AIn-16/18-8B Rev. E	A-4
AIn-32/18 Rev. E	A-4
AIn-8/18 Rev. E	A-5
AIn-8/18-8B Rev. E	A-5
AIn-F-4/14 Rev. E	A-5
AIn-F-4/16 Rev. E	A-6
AIn-F-4/18 Rev. E	A-6
AIn-F-8/14 Rev. E	A-7
AIn-F-8/16 Rev. E	A-7
AIn-F-8/18 Rev. E	A-8
AOut-4/16 Rev. E	A-8
AOut-8/16 Rev. E	A-8
CAN-2 Rev. E	A-8
CNT-D Rev. E	A-9
CNT-I Rev. E	A-9
CNT-T Rev. E	A-9
CPU-T11	A-9
DIO-32 Rev. E	A-9
DIO-32-TiCo Rev. E	A-10
EtherCAT-SL Rev. E	A-10
FlexRay-2 Rev. E	A-10
LIN-2 Rev. E	A-10
MIO-4 Rev. E	A-10
MIO-4-ET1 Rev. E	A-11
OPT-16 Rev. E	A-11
OPT-32 Rev. E	A-11
Profi-SL Rev. E	A-11
PWM-16(-I) Rev. E	A-12
REL-16 Rev. E	A-12
RSxxx-2 Rev. E	A-12
RSxxx-4 Rev. E	A-12
RTD-8 Rev. E	A-12
TC-8-ISO Rev. E	A-12
TRA-16 Rev. E	A-12

AIn-16/18-8B Rev. E

- A:** P2_ADC · 34
P2_ADC24 · 35
P2_ADC_Read_Limit · 36
P2_ADC_Set_Limit · 38
- C:** P2_Check_LED · 4
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- R:** P2_Read_ADC · 39
P2_Read_ADC24 · 40
P2_Read_ADC_SConv · 41
P2_Read_ADC_SConv24 · 42
- S:** P2_Seq_Init · 44
P2_Seq_Read · 47
P2_Seq_Read24 · 48
P2_Seq_Read_Packed · 50
P2_Seq_Start · 51
P2_Seq_Wait · 52
P2_Set_LED · 5
P2_Set_Mux · 53
P2_SE_Diff · 43
P2_Start_Conv · 54
P2_Sync_All · 15
- W:** P2_Wait_EOC · 55
P2_Wait_Mux · 56

AIn-32/18 Rev. E

- A:** P2_ADC · 34
P2_ADC24 · 35
P2_ADC_Read_Limit · 36
P2_ADC_Set_Limit · 38
- C:** P2_Check_LED · 4
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- R:** P2_Read_ADC · 39
P2_Read_ADC24 · 40
P2_Read_ADC_SConv · 41
P2_Read_ADC_SConv24 · 42
- S:** P2_Seq_Init · 44
P2_Seq_Read · 47
P2_Seq_Read24 · 48
P2_Seq_Read_Packed · 50
P2_Seq_Start · 51
P2_Seq_Wait · 52
P2_Set_LED · 5
P2_Set_Mux · 53
P2_SE_Diff · 43
P2_Start_Conv · 54
P2_Sync_All · 15
- W:** P2_Wait_EOC · 55
P2_Wait_Mux · 56

Aln-8/18 Rev. E

- A:** P2_ADC · 34
P2_ADC24 · 35
P2_ADC_Read_Limit · 36
P2_ADC_Set_Limit · 38
- C:** P2_Check_LED · 4
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- R:** P2_Read_ADC · 39
P2_Read_ADC24 · 40
P2_Read_ADC_SConv · 41
P2_Read_ADC_SConv24 · 42
- S:** P2_Seq_Init · 44
P2_Seq_Read · 47
P2_Seq_Read24 · 48
P2_Seq_Read_Packed · 50
P2_Seq_Start · 51
P2_Seq_Wait · 52
P2_Set_LED · 5
P2_Set_Mux · 53
P2_Start_Conv · 54
P2_Sync_All · 15
- W:** P2_Wait_EOC · 55
P2_Wait_Mux · 56

Aln-8/18-8B Rev. E

- A:** P2_ADC · 34
P2_ADC24 · 35
P2_ADC_Read_Limit · 36
P2_ADC_Set_Limit · 38
- C:** P2_Check_LED · 4
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- R:** P2_Read_ADC · 39
P2_Read_ADC24 · 40
P2_Read_ADC_SConv · 41
P2_Read_ADC_SConv24 · 42
- S:** P2_Seq_Init · 44
P2_Seq_Read · 47
P2_Seq_Read24 · 48
P2_Seq_Read_Packed · 50
P2_Seq_Start · 51
P2_Seq_Wait · 52
P2_Set_LED · 5
P2_Set_Mux · 53
P2_Start_Conv · 54
P2_Sync_All · 15
- W:** P2_Wait_EOC · 55
P2_Wait_Mux · 56

Aln-F-4/14 Rev. E

- A:** P2_ADCF · 88
P2_ADCF_Read_Limit · 93
P2_ADCF_Set_Limit · 94
- B:** P2_Burst_CRead_Unpacked1 · 57
P2_Burst_CRead_Unpacked2 · 59
P2_Burst_CRead_Unpacked4 · 61
P2_Burst_Init · 65
P2_Burst_Read · 70
P2_Burst_Read_Index · 68
P2_Burst_Read_Unpacked1 · 72
P2_Burst_Read_Unpacked2 · 74
P2_Burst_Read_Unpacked4 · 76
P2_Burst_Reset · 80
P2_Burst_Start · 82
P2_Burst_Status · 83
P2_Burst_Stop · 85
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 8
P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- R:** P2_Read_ADCF · 100
P2_Read_ADCF32 · 108
P2_Read_ADCF4 · 102
P2_Read_ADCF4_Packed · 106
- S:** P2_Set_Average_Filter · 87
P2_Set_LED · 5
P2_Sync_All · 15
P2_Sync_Enable · 17
P2_Sync_Mode · 19
P2_Sync_Stat · 21

Aln-F-4/16 Rev. E

- A:** P2_ADCF · 88
 P2_ADCF_Mode · 90
 P2_ADCF_Read_Limit · 93
 P2_ADCF_Read_Min_Max4 · 96
 P2_ADCF_Read_Min_Max8 · 98
 P2_ADCF_Reset_Min_Max · 95
 P2_ADCF_Set_Limit · 94
- B:** P2_Burst_CRead_Unpacked1 · 57
 P2_Burst_CRead_Unpacked2 · 59
 P2_Burst_CRead_Unpacked4 · 61
 P2_Burst_Init · 65
 P2_Burst_Read · 70
 P2_Burst_Read_Index · 68
 P2_Burst_Read_Unpacked1 · 72
 P2_Burst_Read_Unpacked2 · 74
 P2_Burst_Read_Unpacked4 · 76
 P2_Burst_Reset · 80
 P2_Burst_Start · 82
 P2_Burst_Status · 83
 P2_Burst_Stop · 85
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 8
 P2_Event_Config · 7
 P2_Event_Enable · 6
 P2_Event_Read · 10
- R:** P2_Read_ADCF · 100
 P2_Read_ADCF32 · 108
 P2_Read_ADCF4 · 102
 P2_Read_ADCF4_Packed · 106
 P2_Read_ADCF_SConv · 109
 P2_Read_ADCF_SConv32 · 111
- S:** P2_Set_Average_Filter · 87
 P2_Set_Gain · 112
 P2_Set_LED · 5
 P2_Start_ConvF · 113
 P2_Sync_All · 15
 P2_Sync_Enable · 17
 P2_Sync_Mode · 19
 P2_Sync_Stat · 21
- W:** P2_Wait_EOCF · 114

Aln-F-4/18 Rev. E

- A:** P2_ADCF · 88
 P2_ADCF24 · 89
 P2_ADCF_Mode · 90
 P2_ADCF_Read_Limit · 93
 P2_ADCF_Set_Limit · 94
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 8
 P2_Event_Config · 7
 P2_Event_Enable · 6
 P2_Event_Read · 10
- R:** P2_Read_ADCF · 100
 P2_Read_ADCF24 · 101
 P2_Read_ADCF32 · 108
 P2_Read_ADCF4 · 102
 P2_Read_ADCF4_24B · 103
 P2_Read_ADCF4_Packed · 106
 P2_Read_ADCF_SConv · 109
 P2_Read_ADCF_SConv24 · 110
 P2_Read_ADCF_SConv32 · 111
- S:** P2_Set_LED · 5
 P2_Start_ConvF · 113
 P2_Sync_All · 15
 P2_Sync_Enable · 17
 P2_Sync_Mode · 19
 P2_Sync_Stat · 21
- W:** P2_Wait_EOCF · 114

Aln-F-8/14 Rev. E

- A:** [P2_ADCF · 88](#)
[P2_ADCF_Read_Limit · 93](#)
[P2_ADCF_Set_Limit · 94](#)
- B:** [P2_Burst_CRead_Unpacked1 · 57](#)
[P2_Burst_CRead_Unpacked2 · 59](#)
[P2_Burst_CRead_Unpacked4 · 61](#)
[P2_Burst_CRead_Unpacked8 · 63](#)
[P2_Burst_Init · 65](#)
[P2_Burst_Read · 70](#)
[P2_Burst_Read_Index · 68](#)
[P2_Burst_Read_Unpacked1 · 72](#)
[P2_Burst_Read_Unpacked2 · 74](#)
[P2_Burst_Read_Unpacked4 · 76](#)
[P2_Burst_Read_Unpacked8 · 78](#)
[P2_Burst_Reset · 80](#)
[P2_Burst_Start · 82](#)
[P2_Burst_Status · 83](#)
[P2_Burst_Stop · 85](#)
- C:** [P2_Check_LED · 4](#)
- E:** [P2_Event2_Config · 8](#)
[P2_Event_Config · 7](#)
[P2_Event_Enable · 6](#)
[P2_Event_Read · 10](#)
- R:** [P2_Read_ADCF · 100](#)
[P2_Read_ADCF32 · 108](#)
[P2_Read_ADCF4 · 102](#)
[P2_Read_ADCF4_Packed · 106](#)
[P2_Read_ADCF8 · 104](#)
[P2_Read_ADCF8_Packed · 107](#)
- S:** [P2_Set_Average_Filter · 87](#)
[P2_Set_LED · 5](#)
[P2_Sync_All · 15](#)
[P2_Sync_Enable · 17](#)
[P2_Sync_Mode · 19](#)
[P2_Sync_Stat · 21](#)

Aln-F-8/16 Rev. E

- A:** [P2_ADCF · 88](#)
[P2_ADCF_Mode · 90](#)
[P2_ADCF_Read_Limit · 93](#)
[P2_ADCF_Read_Min_Max4 · 96](#)
[P2_ADCF_Read_Min_Max8 · 98](#)
[P2_ADCF_Reset_Min_Max · 95](#)
[P2_ADCF_Set_Limit · 94](#)
- B:** [P2_Burst_CRead_Unpacked1 · 57](#)
[P2_Burst_CRead_Unpacked2 · 59](#)
[P2_Burst_CRead_Unpacked4 · 61](#)
[P2_Burst_CRead_Unpacked8 · 63](#)
[P2_Burst_Init · 65](#)
[P2_Burst_Read · 70](#)
[P2_Burst_Read_Index · 68](#)
[P2_Burst_Read_Unpacked1 · 72](#)
[P2_Burst_Read_Unpacked2 · 74](#)
[P2_Burst_Read_Unpacked4 · 76](#)
[P2_Burst_Read_Unpacked8 · 78](#)
[P2_Burst_Reset · 80](#)
[P2_Burst_Start · 82](#)
[P2_Burst_Status · 83](#)
[P2_Burst_Stop · 85](#)
- C:** [P2_Check_LED · 4](#)
- E:** [P2_Event2_Config · 8](#)
[P2_Event_Config · 7](#)
[P2_Event_Enable · 6](#)
[P2_Event_Read · 10](#)
- R:** [P2_Read_ADCF · 100](#)
[P2_Read_ADCF32 · 108](#)
[P2_Read_ADCF4 · 102](#)
[P2_Read_ADCF4_Packed · 106](#)
[P2_Read_ADCF8 · 104](#)
[P2_Read_ADCF8_Packed · 107](#)
[P2_Read_ADCF_SConv · 109](#)
[P2_Read_ADCF_SConv32 · 111](#)
- S:** [P2_Set_Average_Filter · 87](#)
[P2_Set_Gain · 112](#)
[P2_Set_LED · 5](#)
[P2_Start_ConvF · 113](#)
[P2_Sync_All · 15](#)
[P2_Sync_Enable · 17](#)
[P2_Sync_Mode · 19](#)
[P2_Sync_Stat · 21](#)
- W:** [P2_Wait_EOCF · 114](#)

AIn-F-8/18 Rev. E

- A:** P2_ADCF · 88
P2_ADCF24 · 89
P2_ADCF_Mode · 90
P2_ADCF_Read_Limit · 93
P2_ADCF_Set_Limit · 94
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 8
P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- R:** P2_Read_ADCF · 100
P2_Read_ADCF24 · 101
P2_Read_ADCF32 · 108
P2_Read_ADCF4 · 102
P2_Read_ADCF4_24B · 103
P2_Read_ADCF4_Packed · 106
P2_Read_ADCF8 · 104
P2_Read_ADCF8_24B · 105
P2_Read_ADCF8_Packed · 107
P2_Read_ADCF_SConv · 109
P2_Read_ADCF_SConv24 · 110
P2_Read_ADCF_SConv32 · 111
- S:** P2_Set_LED · 5
P2_Start_ConvF · 113
P2_Sync_All · 15
P2_Sync_Enable · 17
P2_Sync_Mode · 19
P2_Sync_Stat · 21
- W:** P2_Wait_EOCF · 114

AOut-4/16 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_DAC · 116
P2_DAC4 · 117
P2_DAC4_Packed · 118
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- S:** P2_Set_LED · 5
P2_Start_DAC · 123
P2_Sync_All · 15
P2_Sync_Enable · 17
P2_Sync_Stat · 21
- W:** P2_Write_DAC · 124
P2_Write_DAC32 · 129
P2_Write_DAC4 · 125
P2_Write_DAC4_Packed · 126

AOut-8/16 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_DAC · 116
P2_DAC4 · 117
P2_DAC4_Packed · 118
P2_DAC8 · 120
P2_DAC8_Packed · 121
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- S:** P2_Set_LED · 5
P2_Start_DAC · 123
P2_Sync_All · 15
P2_Sync_Enable · 17
P2_Sync_Stat · 21
- W:** P2_Write_DAC · 124
P2_Write_DAC32 · 129
P2_Write_DAC4 · 125
P2_Write_DAC4_Packed · 126
P2_Write_DAC8 · 127
P2_Write_DAC8_Packed · 128

CAN-2 Rev. E

- C:** CAN_Msg · 218
P2_CAN_Set_LED · 222
P2_Check_LED · 4
- E:** P2_CAN_Interrupt_Source · 220
P2_En_Interrupt · 223
P2_En_Receive · 225
P2_En_Transmit · 226
P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- G:** P2_Get_CAN_Reg · 227
- I:** P2_Init_CAN · 228
- R:** P2_Read_Msg · 229
P2_Read_Msg_Con · 231
- S:** P2_Set_CAN_Baudrate · 233
P2_Set_CAN_Reg · 234
P2_Set_LED · 5
- T:** P2_Transmit · 235

CNT-D Rev. E

- C:** P2_Check_LED · 4
P2_Cnt_Clear · 162
P2_Cnt_Enable · 163
P2_Cnt_Get_PW · 166
P2_Cnt_Get_PW_HL · 167
P2_Cnt_Get_Status · 165
P2_Cnt_Latch · 168
P2_Cnt_Mode · 169
P2_Cnt_PW_Enable · 164
P2_Cnt_PW_Latch · 171
P2_Cnt_Read · 172
P2_Cnt_Read4 · 173
P2_Cnt_Read_Int_Register · 174
P2_Cnt_Read_Latch · 176
P2_Cnt_Read_Latch4 · 177
P2_Cnt_Sync_Latch · 178
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- S:** P2_Set_LED · 5
P2_SSI_Mode · 180
P2_SSI_Read · 181
P2_SSI_Read2 · 182
P2_SSI_Set_Bits · 183
P2_SSI_Set_Clock · 184
P2_SSI_Set_Delay · 185
P2_SSI_Start · 186
P2_SSI_Status · 187
P2_Sync_All · 15

CNT-I Rev. E

- C:** P2_Check_LED · 4
P2_Cnt_Clear · 162
P2_Cnt_Enable · 163
P2_Cnt_Get_PW · 166
P2_Cnt_Get_PW_HL · 167
P2_Cnt_Get_Status · 165
P2_Cnt_Latch · 168
P2_Cnt_Mode · 169
P2_Cnt_PW_Enable · 164
P2_Cnt_PW_Latch · 171
P2_Cnt_Read · 172
P2_Cnt_Read4 · 173
P2_Cnt_Read_Int_Register · 174
P2_Cnt_Read_Latch · 176
P2_Cnt_Read_Latch4 · 177
P2_Cnt_Sync_Latch · 178
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- S:** P2_Set_LED · 5
P2_Sync_All · 15

CNT-T Rev. E

- C:** P2_Check_LED · 4
P2_Cnt_Clear · 162
P2_Cnt_Enable · 163
P2_Cnt_Get_PW · 166
P2_Cnt_Get_PW_HL · 167
P2_Cnt_Get_Status · 165
P2_Cnt_Latch · 168
P2_Cnt_Mode · 169
P2_Cnt_PW_Enable · 164
P2_Cnt_PW_Latch · 171
P2_Cnt_Read · 172
P2_Cnt_Read4 · 173
P2_Cnt_Read_Int_Register · 174
P2_Cnt_Read_Latch · 176
P2_Cnt_Read_Latch4 · 177
P2_Cnt_Sync_Latch · 178
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- S:** P2_Set_LED · 5
P2_Sync_All · 15

CPU-T11

- C:** CPU_Digin · 11
CPU_Digout · 12
CPU_Dig_IO_Config · 13
CPU_Event_Config · 14
P2_Check_LED · 4
- S:** P2_Set_LED · 5

DIO-32 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 136
P2_Digin_FIFO_Clear · 137
P2_Digin_FIFO_Enable · 138
P2_Digin_FIFO_Full · 140
P2_Digin_FIFO_Read · 141
P2_Digin_Fifo_Read_Fast · 143
P2_Digin_FIFO_Read_Timer · 145
P2_Digin_Long · 146
P2_Digout · 147
P2_Digout_Bits · 148
P2_Digout_Long · 156
P2_Digout_Reset · 157
P2_Digout_Set · 158
P2_DigProg · 159
P2_Dig_Latch · 133
P2_Dig_Read_Latch · 134
P2_Dig_Write_Latch · 135
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- G:** P2_Get_Digout_Long · 160
- S:** P2_Set_LED · 5
P2_Sync_All · 15

DIO-32-TiCo Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 136
 - P2_Digin_FIFO_Clear · 137
 - P2_Digin_FIFO_Enable · 138
 - P2_Digin_FIFO_Full · 140
 - P2_Digin_FIFO_Read · 141
 - P2_Digin_Fifo_Read_Fast · 143
 - P2_Digin_FIFO_Read_Timer · 145
 - P2_Digin_Long · 146
 - P2_Digout · 147
 - P2_Digout_Bits · 148
 - P2_Digout_FIFO_Clear (Rev. E03) · 149
 - P2_Digout_FIFO_Empty (Rev. E03) · 150
 - P2_Digout_FIFO_Enable (Rev. E03) · 151
 - P2_Digout_FIFO_Read_Timer (Rev. E03) · 152
 - P2_Digout_FIFO_Start (Rev. E03) · 153
 - P2_Digout_FIFO_Write (Rev. E03) · 154
 - P2_Digout_Long · 156
 - P2_Digout_Reset · 157
 - P2_Digout_Set · 158
 - P2_DigProg · 159
 - P2_Dig_FIFO_Mode (Rev. E03) · 131
 - P2_Dig_Latch · 133
 - P2_Dig_Read_Latch · 134
 - P2_Dig_Write_Latch · 135
- E:** P2_Event_Config · 7
 - P2_Event_Enable · 6
 - P2_Event_Read · 10
- G:** P2_Get_Digout_Long · 160
- S:** P2_Set_LED · 5
 - P2_Sync_All · 15

EtherCAT-SL Rev. E

- C:** P2_Check_LED · 4
- I:** P2_ECAT_Init · 269
- R:** P2_ECAT_Get_State · 268
 - P2_ECAT_Get_Version · 267
 - P2_ECAT_Read_Data_16L · 271
 - P2_ECAT_Write_Data_16L · 272
- S:** P2_Set_LED · 5

FlexRay-2 Rev. E

- C:** P2_Check_LED · 4
- F:** P2_FlexRay_Get_Version · 274
 - P2_FlexRay_Init · 275
 - P2_FlexRay_Read_Word · 277
 - P2_FlexRay_Reset · 278
 - P2_FlexRay_Set_LED · 279
 - P2_FlexRay_Write_Word · 280
- S:** P2_Set_LED · 5

LIN-2 Rev. E

- C:** P2_Check_LED · 4
- L:** P2_LIN_Get_Version · 254
 - P2_LIN_Init · 249
 - P2_LIN_Init_Apply · 252
 - P2_LIN_Init_Write · 251
 - P2_LIN_Msg_Transmit · 259
 - P2_LIN_Msg_Write · 257
 - P2_LIN_Read_Dat · 255
 - P2_LIN_Reset · 253
 - P2_LIN_Set_LED · 260
- S:** P2_Set_LED · 5

MIO-4 Rev. E

- A:** P2_ADC · 34
 - P2_ADC24 · 35
 - P2_ADC_Read_Limit · 36
 - P2_ADC_Set_Limit · 38
- C:** P2_Check_LED · 4
- D:** P2_DAC · 116
 - P2_DAC4 · 117
 - P2_DAC4_Packed · 118
- E:** P2_Event_Config · 7
 - P2_Event_Enable · 6
 - P2_Event_Read · 10
- M:** P2_MIO_Digin_Long · 27
 - P2_MIO_Digout · 28
 - P2_MIO_Digout_Long · 29
 - P2_MIO_DigProg · 30
 - P2_MIO_Dig_Latch · 24
 - P2_MIO_Dig_Read_Latch · 25
 - P2_MIO_Dig_Write_Latch · 26
 - P2_MIO_Get_Digout_Long · 31
- R:** P2_Read_ADC · 39
 - P2_Read_ADC24 · 40
 - P2_Read_ADC_SConv · 41
 - P2_Read_ADC_SConv24 · 42
- S:** P2_Seq_Init · 44
 - P2_Seq_Read · 47
 - P2_Seq_Read24 · 48
 - P2_Seq_Read_Packed · 50
 - P2_Seq_Start · 51
 - P2_Seq_Wait · 52
 - P2_Set_LED · 5
 - P2_Set_Mux · 53
 - P2_SE_Diff · 43
 - P2_Start_Conv · 54
 - P2_Start_DAC · 123
 - P2_Sync_All · 15
 - P2_Sync_Enable · 17
 - P2_Sync_Stat · 21
- W:** P2_Wait_EOC · 55
 - P2_Wait_Mux · 56
 - P2_Write_DAC · 124
 - P2_Write_DAC32 · 129
 - P2_Write_DAC4 · 125
 - P2_Write_DAC4_Packed · 126

MIO-4-ET1 Rev. E

- A:** P2_ADC · 34
P2_ADC24 · 35
P2_ADC_Read_Limit · 36
P2_ADC_Set_Limit · 38
- C:** P2_Check_LED · 4
P2_Cnt_Clear · 162
P2_Cnt_Enable · 163
P2_Cnt_Get_PW · 166
P2_Cnt_Get_PW_HL · 167
P2_Cnt_Get_Status · 165
P2_Cnt_Latch · 168
P2_Cnt_Mode · 169
P2_Cnt_PW_Enable · 164
P2_Cnt_PW_Latch · 171
P2_Cnt_Read · 172
P2_Cnt_Read_Int_Register · 174
P2_Cnt_Read_Latch · 176
P2_Cnt_Sync_Latch · 178
- D:** P2_DAC · 116
P2_DAC4 · 117
P2_DAC4_Packed · 118
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- M:** P2_MIO_Digin_Long · 27
P2_MIO_Digout · 28
P2_MIO_Digout_Long · 29
P2_MIO_DigProg · 30
P2_MIO_Dig_Latch · 24
P2_MIO_Dig_Read_Latch · 25
P2_MIO_Dig_Write_Latch · 26
P2_MIO_Get_Digout_Long · 31
- R:** P2_Read_ADC · 39
P2_Read_ADC24 · 40
P2_Read_ADC_SConv · 41
P2_Read_ADC_SConv24 · 42
- S:** P2_Seq_Init · 44
P2_Seq_Read · 47
P2_Seq_Read24 · 48
P2_Seq_Read_Packed · 50
P2_Seq_Start · 51
P2_Seq_Wait · 52
P2_Set_LED · 5
P2_Set_Mux · 53
P2_SE_Diff · 43
P2_SSI_Mode · 180
P2_SSI_Read · 181
P2_SSI_Set_Bits · 183
P2_SSI_Set_Clock · 184
P2_SSI_Set_Delay · 185
P2_SSI_Start · 186
P2_SSI_Status · 187
P2_Start_Conv · 54
P2_Start_DAC · 123
P2_Sync_All · 15
P2_Sync_Enable · 17
P2_Sync_Stat · 21

- W:** P2_Wait_EOC · 55
P2_Wait_Mux · 56
P2_Write_DAC · 124
P2_Write_DAC32 · 129
P2_Write_DAC4 · 125
P2_Write_DAC4_Packed · 126

OPT-16 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 136
P2_Digin_FIFO_Clear · 137
P2_Digin_FIFO_Enable · 138
P2_Digin_FIFO_Full · 140
P2_Digin_FIFO_Read · 141
P2_Digin_Fifo_Read_Fast · 143
P2_Digin_FIFO_Read_Timer · 145
P2_Digin_Long · 146
P2_Dig_Latch · 133
P2_Dig_Read_Latch · 134
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- S:** P2_Set_LED · 5
P2_Sync_All · 15

OPT-32 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 136
P2_Digin_FIFO_Clear · 137
P2_Digin_FIFO_Enable · 138
P2_Digin_FIFO_Full · 140
P2_Digin_FIFO_Read · 141
P2_Digin_Fifo_Read_Fast · 143
P2_Digin_FIFO_Read_Timer · 145
P2_Digin_Long · 146
P2_Dig_Latch · 133
P2_Dig_Read_Latch · 134
- E:** P2_Event_Config · 7
P2_Event_Enable · 6
P2_Event_Read · 10
- S:** P2_Set_LED · 5
P2_Sync_All · 15

Profi-SL Rev. E

- C:** P2_Check_LED · 4
- I:** P2_Init_Profibus · 262
- R:** P2_Run_Profibus · 264
- S:** P2_Set_LED · 5

PWM-16(-I) Rev. E

- C:** [P2_Check_LED · 4](#)
- E:** [P2_Event_Config · 7](#)
[P2_Event_Enable · 6](#)
[P2_Event_Read · 10](#)
- P:** [P2_PWM_Enable · 189](#)
[P2_PWM_Get_Status · 190](#)
[P2_PWM_Init · 191](#)
[P2_PWM_Latch · 193](#)
[P2_PWM_Reset · 194](#)
[P2_PWM_Standby_Value · 195](#)
[P2_PWM_Write_Latch · 196](#)
[P2_PWM_Write_Latch_Block · 197](#)
- S:** [P2_Set_LED · 5](#)
[P2_Sync_All · 15](#)
[P2_Sync_Enable · 17](#)
[P2_Sync_Stat · 21](#)

REL-16 Rev. E

- C:** [P2_Check_LED · 4](#)
- D:** [P2_Digout · 147](#)
[P2_Digout_Bits · 148](#)
[P2_Digout_Long · 156](#)
[P2_Digout_Reset · 157](#)
[P2_Digout_Set · 158](#)
[P2_Dig_Latch · 133](#)
[P2_Dig_Write_Latch · 135](#)
- E:** [P2_Event_Config · 7](#)
[P2_Event_Enable · 6](#)
[P2_Event_Read · 10](#)
- G:** [P2_Get_Digout_Long · 160](#)
- S:** [P2_Set_LED · 5](#)
[P2_Sync_All · 15](#)

RSxxx-2 Rev. E

- C:** [P2_Check_LED · 4](#)
[P2_Check_Shift_Reg · 238](#)
- E:** [P2_Event_Config · 7](#)
[P2_Event_Enable · 6](#)
[P2_Event_Read · 10](#)
- G:** [P2_Get_RS · 239](#)
- R:** [P2_ · 245](#)
[P2_Read_FIFO · 240](#)
[P2_RS485_Send · 244](#)
[P2_RS_Init · 241](#)
[P2_RS_Reset · 243](#)
- S:** [P2_Set_LED · 5](#)
[P2_Set_RS · 246](#)
- W:** [P2_Write_Fifo · 247](#)

RSxxx-4 Rev. E

- C:** [P2_Check_LED · 4](#)
[P2_Check_Shift_Reg · 238](#)
- E:** [P2_Event_Config · 7](#)
[P2_Event_Enable · 6](#)
[P2_Event_Read · 10](#)
- G:** [P2_Get_RS · 239](#)
- R:** [P2_D · 245](#)
[P2_Read_FIFO · 240](#)
[P2_RS485_Send · 244](#)
[P2_RS_Init · 241](#)
[P2_RS_Reset · 243](#)
- S:** [P2_Set_LED · 5](#)
[P2_Set_RS · 246](#)
- W:** [P2_Write_Fifo · 247](#)

RTD-8 Rev. E

- C:** [P2_Check_LED · 4](#)
- R:** [P2_RTD_Channel_Config · 200](#)
[P2_RTD_Config · 202](#)
[P2_RTD_Convert · 203](#)
[P2_RTD_Read · 204](#)
[P2_RTD_Read8 · 205](#)
[P2_RTD_Start · 206](#)
[P2_RTD_Status · 208](#)
- S:** [P2_Set_LED · 5](#)

TC-8-ISO Rev. E

- C:** [P2_Check_LED · 4](#)
- S:** [P2_Set_LED · 5](#)
[P2_Sync_All · 15](#)
- T:** [P2_TC_Latch · 209](#)
[P2_TC_Read_Latch · 210](#)
[P2_TC_Read_Latch4 · 212](#)
[P2_TC_Read_Latch8 · 214](#)
[P2_TC_Set_Rate · 216](#)

TRA-16 Rev. E

- C:** [P2_Check_LED · 4](#)
- D:** [P2_Digout · 147](#)
[P2_Digout_Bits · 148](#)
[P2_Digout_Long · 156](#)
[P2_Digout_Reset · 157](#)
[P2_Digout_Set · 158](#)
[P2_Dig_Latch · 133](#)
[P2_Dig_Write_Latch · 135](#)
- E:** [P2_Event_Config · 7](#)
[P2_Event_Enable · 6](#)
[P2_Event_Read · 10](#)
- G:** [P2_Get_Digout_Long · 160](#)
- S:** [P2_Set_LED · 5](#)
[P2_Sync_All · 15](#)

A.3 Thematische Befehlsübersicht

Die Befehle sind in die folgenden Themengruppen aufgeteilt. Innerhalb der Themengruppen sind die Befehle alphabetisch sortiert.

– Analoge Ausgänge:	Seite A-13
– Analoge Eingänge (Fast-ADC):	Seite A-13
– Analoge Eingänge (Multiplexer):	Seite A-15
– CAN-Bus:	Seite A-15
– Digitale Ein-/Ausgänge:	Seite A-16
– EtherCAT:	Seite A-17
– FlexRay:	Seite A-17
– LIN bus:	Seite A-17
– Multi-I/O:	Seite A-17
– Profibus:	Seite A-18
– PWM-Ausgänge:	Seite A-18
– RSxxx:	Seite A-18
– SSI-Decoder:	Seite A-18
– System:	Seite A-18
– Temperatur-Eingänge:	Seite A-19
– Zähler:	Seite A-19

Analoge Ausgänge

P2_DAC	gibt auf einem Kanal des angegebenen Moduls eine (analoge) Spannung aus, die dem angegebenen Digitalwert entspricht.
P2_DAC4	gibt 4 Digitalwerte aus einem Feld auf die DAC 1...4 des angegebenen Moduls als (analoge) Spannung aus.
P2_DAC4_Packed	gibt 4 gepackte Digitalwerte aus einem Feld auf die DAC 1...4 des angegebenen Moduls als (analoge) Spannung aus.
P2_DAC8	gibt 8 Digitalwerte aus einem Feld auf die DAC 1...8 des angegebenen Moduls als (analoge) Spannung aus.
P2_DAC8_Packed	gibt die Digitalwerte aus einem Feld auf den DAC 1...8 des angegebenen Moduls als (analoge) Spannung aus.
P2_Start_DAC	startet die Wandlung bzw. Ausgabe aller DAC des angegebenen D/A-Moduls.
P2_Write_DAC	schreibt einen Digitalwert in das Ausgaberegister eines bestimmten DAC des angegebenen Moduls.
P2_Write_DAC32	kopiert aus einem 32 Bit-Wert zwei 16 Bit-Werte in die Ausgaberegister eines DAC-Paars des angegebenen Moduls.
P2_Write_DAC4	schreibt 4 Digitalwerte aus einem Feld in die Ausgaberegister der DAC 1...4 des angegebenen Moduls.
P2_Write_DAC4_Packed	schreibt 4 Digitalwerte aus einem Feld in die Ausgaberegister der DAC 1...4 des angegebenen Moduls.
P2_Write_DAC8	schreibt 8 Digitalwerte aus einem Feld in die Ausgaberegister der DAC 1...8 des angegebenen Moduls.
P2_Write_DAC8_Packed	schreibt 8 Digitalwerte aus einem Feld in die Ausgaberegister der DAC 1...8 des angegebenen Moduls.

Analoge Eingänge (Fast-ADC)

P2_ADCF	führt eine komplette Messung auf einem Fast-ADC durch. Der Rückgabewert hat 16 Bit Auf-
---------	---

	lösung.
P2_ADCF24	führt eine komplette Messung auf einem Fast-ADC durch. Der Rückgabewert hat 24 Bit Auflösung.
P2_ADCF_Mode	stellt den Arbeitsmodus für alle Kanäle der angegebenen Module ein.
P2_ADCF_Read_Limit	liest die Flags für Grenzwertüber- und unterschreitungen auf allen F-ADC des angegebenen Moduls aus.
P2_ADCF_Read_Min_Max4	gibt die Minimal- und Maximalwerte von 4 F-ADC des angegebenen Moduls in einem Feld zurück.
P2_ADCF_Read_Min_Max8	gibt die Minimal- und Maximalwerte von 8 F-ADC des angegebenen Moduls in einem Feld zurück.
P2_ADCF_Reset_Min_Max	setzt die Minimal- und Maximalwerte bestimmter Kanäle auf dem angegebenen Modul zurück.
P2_ADCF_Set_Limit	setzt den oberen und unteren Grenzwert für einen F-ADC des angegebenen Moduls.
P2_Burst_CRead_Unpacked1	kopiert eine Anzahl der zuletzt gemessenen Messwerte eines Kanals aus dem Speicher des angegebenen Moduls in ein Feld.
P2_Burst_CRead_Unpacked2	kopiert eine Anzahl der zuletzt gemessenen Messwerte zweier Kanäle aus dem Speicher des angegebenen Moduls in 2 Felder.
P2_Burst_CRead_Unpacked4	kopiert eine Anzahl der zuletzt gemessenen Messwerte von 4 Kanälen aus dem Speicher des angegebenen Moduls in 4 Felder.
P2_Burst_CRead_Unpacked8	kopiert eine Anzahl der zuletzt gemessenen Messwerte von 8 Kanälen aus dem Speicher des angegebenen Moduls in 8 Felder.
P2_Burst_Init	legt die Kennwerte für eine Burst-Messreihe auf dem angegebenen Modul fest.
P2_Burst_Read	kopiert 32 Bit-Werte aus dem Speicher des angegebenen Moduls in ein bestimmtes Feld.
P2_Burst_Read_Index	gibt die Adresse im Modulspeicher zurück, an der zuletzt Messwerte abgelegt wurden.
P2_Burst_Read_Unpacked1	kopiert die Messwerte eines Kanals aus dem Speicher des angegebenen Moduls in ein Feld.
P2_Burst_Read_Unpacked2	kopiert die Messwerte von 2 Kanälen aus dem Speicher des angegebenen Moduls in 2 Felder.
P2_Burst_Read_Unpacked4	kopiert die Messwerte von 4 Kanälen aus dem Speicher des angegebenen Moduls in 4 Felder.
P2_Burst_Read_Unpacked8	kopiert die Messwerte von 8 Kanälen aus dem Speicher des angegebenen Moduls in 8 Felder.
P2_Burst_Reset	setzt den Datenzeiger der Burst-Messreihe auf allen angegebenen Modulen zurück.
P2_Burst_Start	startet eine Burst-Messreihe auf allen angegebenen Modulen gleichzeitig.
P2_Burst_Status	ermittelt die Anzahl der noch durchzuführenden Burst-Messungen auf dem angegebenen Modul.
P2_Burst_Stop	unterbricht eine laufende Burst-Messreihe auf allen angegebenen Modulen gleichzeitig.
P2_Read_ADCF	liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus. Das Ergebnis hat 16 Bit Auflösung.
P2_Read_ADCF24	liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus. Das Ergebnis hat 24 Bit Auflösung.
P2_Read_ADCF32	liest die Wandlungsergebnisse aus 2 aufeinander folgenden F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.
P2_Read_ADCF4	liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus.
P2_Read_ADCF4_24B	liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus. Die Ergebnisse haben eine Auflösung von 24 Bit.
P2_Read_ADCF4_Packed	liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus.
P2_Read_ADCF8	liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus.
P2_Read_ADCF8_24B	liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus. Die Ergebnisse haben eine Auflösung von 24 Bit.
P2_Read_ADCF8_Packed	liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus.
P2_Read_ADCF_SConv	liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung.
P2_Read_ADCF_SConv24	liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus und startet

	sofort eine neue Konvertierung.
P2_Read_ADCF_SConv32	liest die Wandlungsergebnisse aus 2 F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.
P2_Set_Average_Filter	legt fest, ob und aus wievielen Messwerten das angegebene Modul einen Mittelwert berechnet.
P2_Set_Gain	setzt für einen Kanal des angegebenen Moduls die Betriebsart fest und damit auch den Verstärkungsfaktor und den Messbereich.
P2_Start_ConvF	startet die Wandlung auf einem oder mehreren F-ADC des angegebenen Moduls.
P2_Wait_EOCF	wartet, bis die Wandlung auf allen angegebenen F-ADC des Moduls abgeschlossen ist.

Analoge Eingänge (Multiplexer)

P2_ADC	führt eine komplette Messung auf einem ADC des angegebenen Moduls durch. Der Rückgabewert hat 16 Bit Auflösung.
P2_ADC24	führt eine komplette Messung auf einem ADC des angegebenen Moduls durch. Der Rückgabewert ist (linksbündig) auf 24 Bit formatiert.
P2_ADC_Read_Limit	liest die Flags für Grenzwertüber- und unterschreitungen auf 16 Eingangskanälen des angegebenen Moduls aus.
P2_ADC_Set_Limit	setzt den oberen und unteren Grenzwert für einen analogen Eingang des angegebenen Moduls.
P2_Read_ADC	liest das Wandlungsergebnis des angegebenen Moduls aus. Das Ergebnis hat 16 Bit Auflösung.
P2_Read_ADC24	liest das Wandlungsergebnis des angegebenen Moduls aus. Der Rückgabewert ist (linksbündig) auf 24 Bit formatiert.
P2_Read_ADC_SConv	liest das Wandlungsergebnis des angegebenen Moduls aus und startet sofort eine neue Konvertierung.
P2_Read_ADC_SConv24	liest das Wandlungsergebnis des angegebenen Moduls aus und startet sofort eine neue Konvertierung.
P2_Seq_Init	initialisiert das angegebene Modul für den Betrieb mit Ablaufsteuerung.
P2_Seq_Read	kopiert eine bestimmte Anzahl an Messwerten (16 Bit) von dem angegebenen Modul in ein Ziel-Feld.
P2_Seq_Read24	kopiert eine bestimmte Anzahl an Messwerten (24 Bit) von dem angegebenen Modul in ein Ziel-Feld.
P2_Seq_Read_Packed	kopiert eine gerade Anzahl an Messwerten (16 Bit) paarweise von dem angegebenen Modul in ein Ziel-Feld.
P2_Seq_Start	startet die Ablaufsteuerung auf allen angegebenen Modulen gleichzeitig.
P2_Seq_Wait	wartet, bis die Ablaufsteuerung auf dem angegebenen Modul alle Kanäle der Messgruppen gewandelt und gespeichert hat.
P2_Set_Mux	stellt den Multiplexer des angegebenen Moduls auf einen bestimmten Eingang und eine bestimmte Verstärkung ein.
P2_SE_Diff	stellt die Betriebsart single ended oder differentiell für alle Analog-Eingänge auf dem angegebenen Modul ein.
P2_Start_Conv	startet die Wandlung auf dem angegebenen Modul.
P2_Wait_EOC	wartet, bis die Wandlung auf dem angegebenen Modul abgeschlossen ist.
P2_Wait_Mux	wartet, bis das Einschwingen des Multiplexers auf dem angegebenen Modul abgeschlossen ist.

CAN-Bus

CAN_Msg	ist ein eindimensionales Feld bestehend aus 9 Elementen, in dem die Message-Objekte (Nachrichten) des CAN-Busses beim Senden und Empfangen gespeichert sind oder werden.
P2_CAN_Interrupt_Source	gibt zurück, welche CAN-Kanäle einen Interrupt ausgelöst haben.
P2_CAN_Set_LED	schaltet die Zusatz-LED für einen CAN-Kanal auf dem Modul ein (mit Farbe) oder aus.
P2_En_Interrupt	konfiguriert ein bestimmtes Message-Objekt des angegebenen Moduls, so dass bei Eintref-

	fen einer Nachricht ein Event-Signal (Interrupt) erzeugt wird.
P2_En_Receive	gibt ein Message-Objekt für den Empfang von Nachrichten auf dem angegebenen Modul frei.
P2_En_Transmit	gibt ein Message-Objekt für das Senden von Nachrichten auf dem angegebenen Modul frei.
P2_Get_CAN_Reg	gibt den Inhalt eines bestimmten Registers auf einem CAN-Controller des angegebenen Moduls zurück.
P2_Init_CAN	initialisiert einen der CAN-Controller auf dem angegebenen Modul und bringt ihn in einen definierten Anfangszustand.
P2_Read_Msg	gibt zurück, ob eine neue Nachricht in einem Message-Objekt eines der CAN-Controller auf dem angegebenen Modul empfangen wurde.
P2_Read_Msg_Con	gibt zurück, ob eine neue Nachricht in einem Message-Objekt eines der CAN-Controller auf dem angegebenen Modul empfangen wurde.
P2_Set_CAN_Baudrate	stellt die angegebene Baudrate auf einem der Controller auf dem angegebenen Modul ein.
P2_Set_CAN_Reg	schreibt einen Wert in ein Register des gewählten CAN-Controllers auf dem angegebenen Modul.
P2_Transmit	liest die Daten aus dem Feld CAN_Msg und sendet die Daten als Nachricht.

Digitale Ein-/Ausgänge

P2_Digin_Edge	gibt zurück, ob an den Digitaleingängen des angegebenen Moduls eine positive oder negative Flanke aufgetreten ist.
P2_Digin_FIFO_Clear	löscht den FIFO der Flankenüberwachung auf dem angegebenen Modul.
P2_Digin_FIFO_Enable	legt fest, an welchen Eingangskanälen des angegebenen Moduls die Flanken überwacht werden.
P2_Digin_FIFO_Full	gibt die Anzahl der gespeicherten Wertepaare im FIFO der Flankenüberwachung zurück.
P2_Digin_FIFO_Read	liest die Wertepaare aus dem FIFO der Flankenüberwachung und schreibt sie in in 2 Felder.
P2_Digin_Fifo_Read_Fast	P2_Digin_FIFO_Read liest die Wertepaare aus dem FIFO der Flankenüberwachung und schreibt sie in in 2 Felder.
P2_Digin_FIFO_Read_Timer	gibt den aktuellen Stand des 100MHz-Zählers auf dem angegebenen Modul zurück.
P2_Digin_Long	gibt den Zustand der Eingänge (Bits 31...00) des angegebenen Moduls als Bitmuster zurück.
P2_Digout	setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.
P2_Digout_Bits	setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel High oder den Pegel Low.
P2_Digout_FIFO_Clear	stoppt die Flankenausgabe und löscht den FIFO der Flankenausgabe auf dem angegebenen Modul.
P2_Digout_FIFO_Empty	gibt die Anzahl der freien Wertepaare im FIFO der Flankenausgabe zurück.
P2_Digout_FIFO_Enable	legt fest, an welchen Ausgangskanälen des angegebenen Moduls Flanken ausgegeben werden.
P2_Digout_FIFO_Read_Timer	gibt den aktuellen Stand des 100MHz-Zählers auf dem angegebenen Modul zurück.
P2_Digout_FIFO_Start	startet die Ausgabe der Flankenausgabe auf dem angegebenen Modul.
P2_Digout_FIFO_Write	schreibt Wertepaare in den FIFO der Flankenausgabe.
P2_Digout_Long	setzt oder löscht durch den übergebenen 32 Bit-Wert alle Ausgänge des angegebenen Moduls.
P2_Digout_Reset	setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel Low.
P2_Digout_Set	setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel High.
P2_DigProg	programmiert die digitalen Kanäle 0...31 des angegebenen Moduls in Gruppen zu je 8 als Ein- oder Ausgang.
P2_Dig_FIFO_Mode	stellt den Betriebsmodus des FIFO auf dem angegebenen Modul ein als Eingang mit Flankenüberwachung oder als Ausgang mit Flankenausgabe.
P2_Dig_Latch	überträgt auf dem angegebenen Modul Digital-Informationen von den Eingängen in die Eingangs-Latches und von den Ausgangs-Latches zu den Ausgängen.
P2_Dig_Read_Latch	liefert die Bitwerte aus dem Latch-Register für digitale Eingänge auf dem angegebenen Modul.
P2_Dig_Write_Latch	schreibt einen 32 Bit-Wert in das Latch-Register für digitale Ausgänge auf dem angegebenen Modul.
P2_Get_Digout_Long	gibt den Inhalt des Ausgangs-Latches (Register für digitale Ausgänge) auf dem angegebenen

Modul zurück.

EtherCAT

- P2_ECAC_Get_State P2_ECAC_Get_Version gibt die Version der EtherCAT-Schnittstelle zurück.
- P2_ECAC_Get_Version gibt die Version der EtherCAT-Schnittstelle zurück.
- P2_ECAC_Init initialisiert den EtherCAT-Slave.
- P2_ECAC_Read_Data_16L P2_ECAC_Get_Version gibt die Version der EtherCAT-Schnittstelle zurück.
- P2_ECAC_Write_Data_16L P2_ECAC_Get_Version gibt die Version der EtherCAT-Schnittstelle zurück.

FlexRay

- P2_FlexRay_Get_Version gibt die Versionsnummer der FlexRay-Schnittstelle zurück.
- P2_FlexRay_Init initialisiert die Datenübertragung zwischen ADwin CPU und einer FlexRay-Schnittstelle auf einem bestimmten Modul.
- P2_FlexRay_Read_Word gibt einen 16 Bit-Wert aus einem FlexRay-Controller auf dem angegebenen Modul zurück.
- P2_FlexRay_Reset setzt einen FlexRay-Controller auf dem angegebenen Modul zurück.
- P2_FlexRay_Set_LED schaltet eine Kanal-LED eines FlexRay-Controllers auf dem angegebenen Modul ein oder aus.
- P2_FlexRay_Write_Word schreibt einen 16 Bit-Wert an eine Adresse in einem FlexRay-Controller des angegebenen Moduls.

LIN bus

- P2_LIN_Get_Version gibt die Versionsnummer der LIN-Schnittstelle zurück.
- P2_LIN_Init initialisiert die Datenübertragung zwischen ADwin CPU und der LIN-Schnittstelle auf einem bestimmten Modul.
- P2_LIN_Init_Apply aktiviert die mit P2_LIN_Init_Write eingestellten Betriebsdaten für alle LIN-Schnittstellen.
- P2_LIN_Init_Write legt Baudrate und Betriebsmodus für eine bestimmte LIN-Schnittstelle fest.
- P2_LIN_Msg_Transmit sendet einen Header auf den LIN-Bus. Nur gültig im Betriebsmodus LIN Master.
- P2_LIN_Msg_Write konfiguriert eine Messagebox in einer LIN-Schnittstelle zum Senden oder Empfangen.
- P2_LIN_Read_Dat liest die Daten einer Messagebox oder den Status einer LIN-Schnittstelle und schreibt sie in ein Feld.
- P2_LIN_Reset setzt alle LIN-Kanäle zurück, und zwar entweder alle Einstellungen (Einschaltzustand) oder nur die LIN-internen Zähler.
- P2_LIN_Set_LED schaltet die Zusatz-LED für eine LIN-Schnittstelle auf dem Modul ein (mit Farbe) oder aus.

Multi-I/O

- P2_MIO_Digin_Long gibt den Zustand der Eingänge (Bits 7...0) des angegebenen Moduls als Bitmuster zurück.
- P2_MIO_Digout setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.
- P2_MIO_Digout_Long setzt oder löscht alle Ausgänge des angegebenen Moduls.
- P2_MIO_DigProg programmiert die digitalen Kanäle 0...7 des angegebenen Moduls in Gruppen zu je 4 als Ein- oder Ausgang.
- P2_MIO_Dig_Latch überträgt auf dem angegebenen Modul Digital-Informationen von den Eingängen in die Eingangs-Latches und von den Ausgangs-Latches zu den Ausgängen.
- P2_MIO_Dig_Read_Latch liefert die Bitwerte aus dem Latch-Register für digitale Eingänge auf dem angegebenen Modul.
- P2_MIO_Dig_Write_Latch schreibt einen 32 Bit-Wert in das Latch-Register für digitale Ausgänge auf dem angegebenen Modul.
- P2_MIO_Get_Digout_Long gibt den Inhalt des Ausgangs-Latches (Register für digitale Ausgänge) auf dem angegebenen Modul zurück.

Profibus

P2_Init_Profibus	initialisiert den Profibus-Slave.
P2_Run_Profibus	tauscht Daten mit dem Profibus-Slave aus.

PWM-Ausgänge

P2_PWM_Enable	gibt einen oder mehrere PWM-Ausgänge zur Ausgabe frei oder sperrt sie.
P2_PWM_Get_Status	liest den aktuellen Betriebsstatus für alle PWM-Ausgänge.
P2_PWM_Init	setzt die Voreinstellungen für den angegebenen PWM-Ausgang.
P2_PWM_Latch	aktiviert Frequenz und Tastverhältnis eines oder mehrerer PWM-Ausgänge für die PWM-Ausgabe.
P2_PWM_Reset	stoppt die Ausgabe auf einem oder mehreren Ausgängen sofort.
P2_PWM_Standby_Value	setzt den Vorgabewert (TTL-Pegel) für einen PWM-Ausgang.
P2_PWM_Write_Latch	schreibt Frequenz und Tastverhältnis in das Latch-Register.
P2_PWM_Write_Latch_Block	schreibt Frequenz und Tastverhältnis für mehrere PWM-Ausgänge in das Latch-Register.

RSxxx

P2_Check_Shift_Reg	gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO des Kanals (auf dem angegebenen Modul) geschrieben wurden.
P2_Get_RS	liest den Inhalt eines bestimmten Controller-Registers auf dem angegebenen Modul aus.
P2_Read_FIFO	liest einen Wert aus dem Eingangs-FIFO eines bestimmten Kanals auf dem angegebenen Modul.
P2_RS485_Send	legt die Übertragungsrichtung für einen bestimmten Kanal des angegebenen Moduls fest.
P2_RS_Init	initialisiert einen bestimmten Kanal auf dem angegebenen Modul.
P2_RS_Reset	führt auf dem angegebenen Modul einen Hardware-Reset durch und löscht dabei die Einstellungen für alle Kanäle.
P2_RS_Set_LED	schaltet die Zusatz-LED für eine RSxxx-Schnittstelle auf dem Modul ein (mit Farbe) oder aus.
P2_Set_RS	schreibt einen Wert in ein bestimmtes Register des angegebenen Moduls.
P2_Write_FIFO	schreibt einen Wert in den Sende-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

SSI-Decoder

P2_SSI_Mode	stellt den Modus aller SSI-Decoder auf dem angegebenen Modul ein, entweder „single shot“ (einzelne lesen) und „continuous“ (kontinuierlich lesen)..
P2_SSI_Read	gibt den zuletzt gespeicherten Zählerstand eines bestimmten SSI-Decoders auf dem angegebenen Modul zurück.
P2_SSI_Read2	gibt den zuletzt gespeicherten Zählerstand von beiden SSI-Decodern auf dem angegebenen Modul zurück.
P2_SSI_Set_Bits	stellt für einen SSI-Decoder auf dem angegebenen Modul die Anzahl der zu Bits ein, die einen vollständigen Encoder-Wert bilden.
P2_SSI_Set_Clock	stellt die Taktrate (ca. 12kHz bis 25MHz) auf dem angegebenen Modul ein, mit der der Decoder getaktet wird.
P2_SSI_Set_Delay	stellt für einen bestimmten SSI-Zähler auf dem angegebenen Modul den Zeitabstand zwischen dem Einlesen von zwei Encoder-Werten ein.
P2_SSI_Start	startet auf dem angegebenen Modul das Auslesen eines oder beider SSI-Decoder (nur im Modus single shot).
P2_SSI_Status	liefert für einen bestimmten Decoder den aktuellen Lese-Status auf dem angegebenen Modul zurück.

System

P2_Check_LED	gibt den Status der LED (oben auf der Frontplatte) auf dem angegebenen Modul zurück.
CPU_Digin (T11)	Nur Prozessor T11. CPU_Digin gibt zurück, ob seit dem letzten Befehlsaufruf eine Flanke an

	einem DIG I/O-Eingang des Prozessormoduls aufgetreten ist.
CPU_Digout	setzt einen DIG I/O-Ausgang des Prozessormoduls auf den angegebenen TTL-Pegel.
CPU_Dig_IO_Config	konfiguriert alle DIG I/O-Kanäle des Prozessormoduls.
CPU_Event_Config	konfiguriert den EVENT IN-Kanal des Prozessormoduls.
P2_Event2_Config	konfiguriert die Vorverarbeitung der Event-Signale auf dem angegebenen Modul.
P2_Event_Config	konfiguriert den externen Event-Eingang des angegebenen Moduls.
P2_Event_Enable	sperrt oder aktiviert den externen Event-Eingang auf dem angegebenen Modul.
P2_Event_Read	gibt den aktuellen TTL-Pegel an den Event-Eingängen des angegebenen Moduls zurück.
P2_Set_LED	schaltet die LED (oben auf der Frontplatte) auf dem angegebenen Modul ein oder aus.
P2_Sync_All	startet auf den angegebenen Modulen synchron eine bestimmte Aktion.
P2_Sync_Enable	aktiviert oder deaktiviert die gewählten Eingänge, Ausgänge oder Funktionsgruppen auf dem angegebenen Modul für die Synchron-Option.
P2_Sync_Mode	aktiviert oder deaktiviert die Synchronisation (von Messwert-Wandlungen) mit anderen Modulen als Master oder Slave.
P2_Sync_Stat	gibt die Einstellung der Synchron-Option auf dem angegebenen Modul zurück.

Temperatur-Eingänge

P2_RTD_Channel_Config	stellt den Temperatur-Messmodus für einen bestimmten Kanal auf dem angegebenen Modul ein.
P2_RTD_Config	initialisiert die Temperaturmessung auf dem angegebenen Modul.
P2_RTD_Convert	berechnet aus dem Digitalwert eines Temperatur-Fühlers den zugehörigen Widerstand oder die Temperatur in Celsius oder Fahrenheit.
P2_RTD_Read	gibt den aktuellen Temperaturmesswert eines bestimmten Kanals auf dem angegebenen Modul zurück.
P2_RTD_Read8	gibt die aktuellen Temperaturmesswerte aller Kanäle auf dem angegebenen Modul in einem Feld zurück.
P2_RTD_Start	startet den Temperatur-Messzyklus auf den angegebenen Modulen gleichzeitig.
P2_RTD_Status	gibt den Status eines einfachen Temperatur-Messzyklus auf dem angegebenen Modul zurück.
P2_TC_Latch	kopiert die an den Eingängen anliegenden Spannungswerte in die Latches.
P2_TC_Read_Latch	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.
P2_TC_Read_Latch4	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) der Kanäle 1...4 auf dem Modul zurück.
P2_TC_Read_Latch8	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) der Kanäle 1...8 auf dem Modul zurück.
P2_TC_Set_Rate	stellt die Abtastrate für das angegebene Modul ein.

Zähler

P2_Cnt_Clear	setzt einen oder mehrere Zähler auf Null, gemäß dem Bitmuster in pattern.
P2_Cnt_Enable	hält die gewählten Zähler an oder gibt sie frei, um eingehende Impulse zu zählen.
P2_Cnt_Get_PW	gibt Frequenz und Tastverhältnis eines PWM-Zählers zurück.
P2_Cnt_Get_PW_HL	gibt die Eintastzeit und die Austastzeit eines PWM-Zählers zurück.
P2_Cnt_Get_Status	gibt den Inhalt des Statusregisters für einen Zähler zurück.
P2_Cnt_Latch	überträgt den aktuellen Zählerstand eines oder mehrerer Zähler in das zugehörige Latch, je nach Bitmuster in pattern.
P2_Cnt_Mode	definiert die Betriebsart eines Zählers.
P2_Cnt_PW_Enable	hält die gewählten PWM-Zähler an oder gibt sie frei, um eingehende Impulse zu zählen.
P2_Cnt_PW_Latch	kopiert den Inhalt eines oder mehrerer PWM-Zähler in einen Zwischenspeicher.
P2_Cnt_Read	überträgt einen aktuellen Zählerstand in das zugehörige Latch und gibt ihn als Rückgabewert

	zurück.
P2_Cnt_Read4	überträgt alle 4 Zählerstände in die zugehörigen Latches A und gibt sie in einem Feld zurück.
P2_Cnt_Read_Int_Register	gibt den Inhalt eines Zählerregisters zurück.
P2_Cnt_Read_Latch	gibt den Wert aus dem Latch eines Zählers als Rückgabewert zurück.
P2_Cnt_Read_Latch4	gibt die Werte aus den Latches A aller 4 Zähler in einem Feld zurück.
P2_Cnt_Sync_Latch	kopiert die Inhalte der gewählten Zähler und PWM-Zähler in Zwischenspeicher.