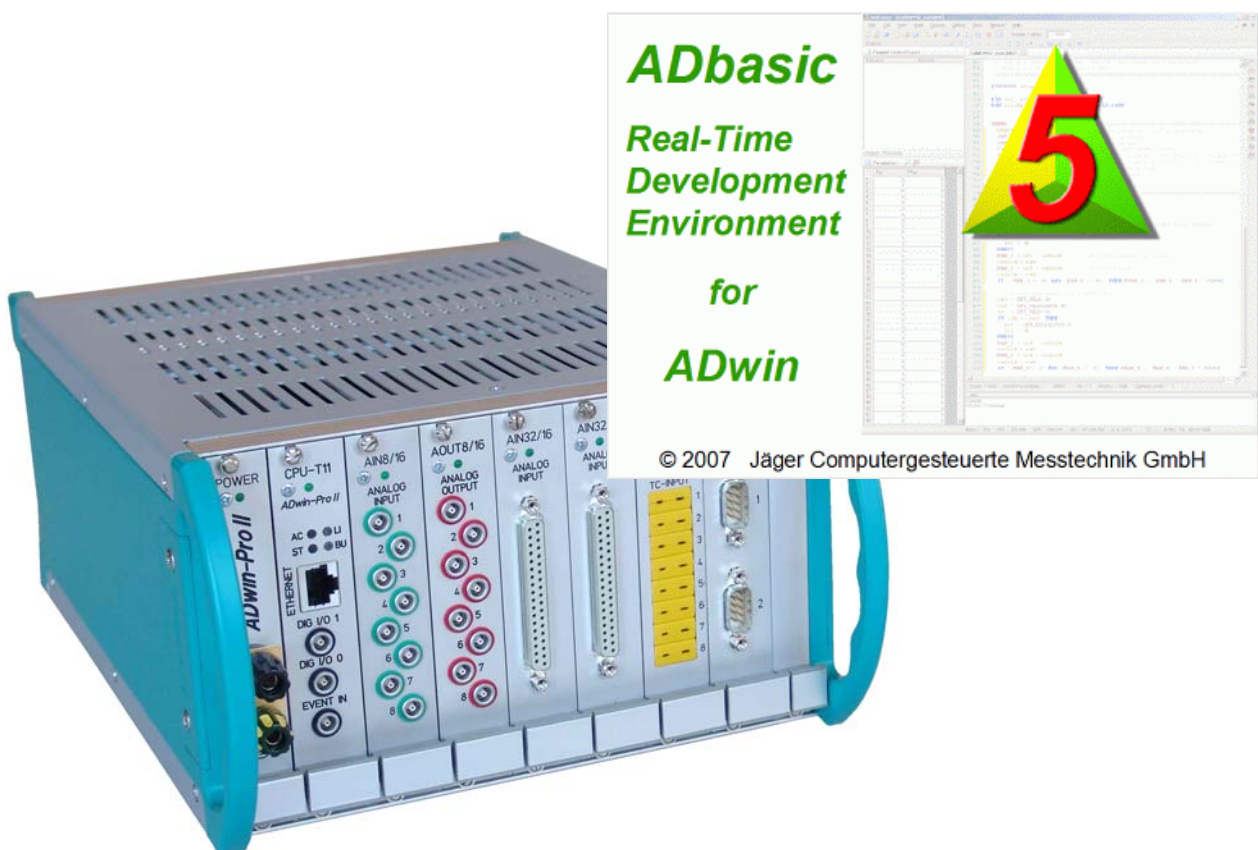


ADwin-Pro II

Systembeschreibung Programmierung in *ADbasic*



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
3 ADbasic -Befehle	3
3.1 Pro II: Allgemeine Befehle	3
3.2 Pro II: Digitalkanäle der CPU	21
3.3 Pro II: Multi-I/O	26
3.4 Pro II: Analoge Eingänge	35
3.5 Pro II: Analoge Ausgänge	124
3.6 Pro II: Digitale Ein-/Ausgänge	144
3.7 Pro II: Zähler	186
3.8 Pro II: CAN-Bus	212
3.9 Pro II: CAN FD-Bus	232
3.10 Pro II: LIN-Bus-Schnittstelle	269
3.11 Pro II: PWM-Ausgänge	283
3.12 Pro II: Temperaturmess-Module	294
3.13 Pro II: Dehnungsmessstreifen-Module	312
3.14 Pro II: RSxxx	324
3.15 Pro II: Profibus/Profinet-Schnittstelle	338
3.16 Pro II: MIL-STD-1553	348
3.17 Pro II: ARINC-429	358
3.18 Pro II: EtherCAT-Schnittstelle	372
3.19 Pro II: Flexray	385
3.20 Pro II: SENT-Schnittstelle	392
3.21 Pro II: SPI-Schnittstelle	433
4 Programmbeispiele	458
4.1 Online-Auswertung von Messwerten (Pro II)	458
4.2 Digitaler Proportional-Regler (Pro II)	459
4.3 Datenaustausch mit DATA-Feldern (Pro II)	459
4.4 Digitaler PID-Regler (Pro II)	460
4.5 Beispiele für RS232 und RS485 (Pro II)	462
4.6 Kontinuierliche Messwertwandlung (Pro II)	466
Befehlsübersichten	A-1
A.1 Alphabetische Befehlsübersicht	A-1
A.2 Befehlsübersicht nach Modulen	A-5
A.3 Thematische Befehlsübersicht	A-24

Typografische Konventionen

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



Einen „Hinweis“ finden Sie bei

- Informationen, die für einen fehlerfreien Betrieb unbedingt beachtet werden müssen.
- Tipps und Ratschlägen für einen effizienten Betrieb.

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



C:\ADwin\...

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

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.

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.

Beachten Sie: Befehle zur Programmierung von Modulen am LS-Bus sind in separaten Handbüchern enthalten, beispielsweise im Handbuch HSM-24V.

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.

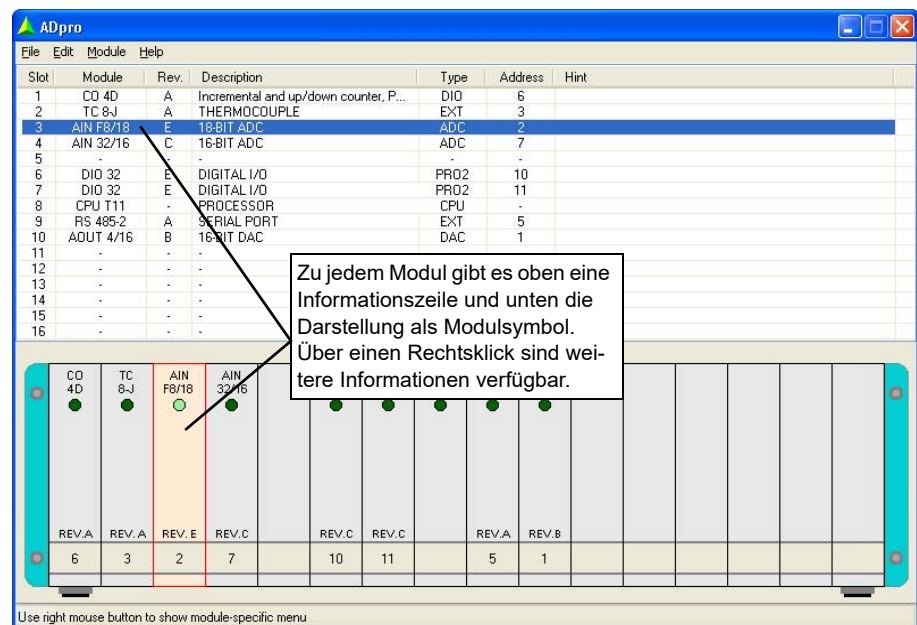
2 Das Programm ADpro.exe

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

- Bestückung eines ADwin-Pro Systems anzeigen sowie Informationen zu den Modulen.
- Moduladresse für Pro II-Module einstellen (siehe Hardware-Handbuch).
Bei Pro I-Modulen wird die Moduladresse manuell eingestellt; im Programm wird die Adresse nur angezeigt.
- Funktion von Pro I- und Pro II-Modulen prüfen: analoge Ein-/Ausgangsmodule, Digital- und Zählermodule, einige Busmodule.
- Pro I- und Pro II-Module kalibrieren (analoge Ein-/Ausgangsmodule).

Die Kalibrierung erfüllt nur einfache Ansprüche.

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



Hinweise

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

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

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

3 ADbasic-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)
- [Calc_Processdelay](#) (Seite 6)
- [Calc_TicksToNs](#) (Seite 7)
- [P2_Event_Enable](#) (Seite 8)
- [P2_Event_Config](#) (Seite 9)
- [P2_Event2_Config](#) (Seite 10)
- [P2_Event_Read](#) (Seite 12)
- [P2_Sync_All](#) (Seite 13)
- [P2_Sync_Enable](#) (Seite 15)
- [P2_Sync_Mode](#) (Seite 17)
- [P2_Sync_Stat](#) (Seite 19)



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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-TiCo Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo 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, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CAN-FD-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, Comp-16 Rev. E, CPU-T11, CPU-T12, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, EtherCAT-SL Rev. E, EtherCAT-SL-40 Rev. E, FlexRay-2 Rev. E, LIN-2 Rev. E, LS-2 Rev. E, MIL-1553 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, Profi-IRT-CU Rev. E, Profi-IRT-FO Rev. E, Profi-SL Rev. E, Profi-SL-40 Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, RTD-8 Rev. E, SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SG-4/18 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 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 zusätzlicher LED wird mit separaten Befehlen eingestellt.

Siehe auch

[P2_Check_LED](#), [P2_CAN_Set_LED](#), [P2_LIN_Set_LED](#), [P2_RS_Set_LED](#),
[P2_MIL_Set_LED](#), [P2_FlexRay_Set_LED](#), [P2_CANFD_Set_LED](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo 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, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, ARINC-429 Rev. E, CAN-2 Rev. E, CAN-FD-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, Comp-16 Rev. E, CPU-T11, CPU-T12, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, EtherCAT-SL Rev. E, EtherCAT-SL-40 Rev. E, FlexRay-2 Rev. E, LIN-2 Rev. E, LS-2 Rev. E, MIL-1553 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, Profi-IRT-CU Rev. E, Profi-IRT-FO Rev. E, Profi-SL Rev. E, Profi-SL-40 Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, RTD-8 Rev. E, SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SG-4/18 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 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

Calc_ Processdelay

Calc_Processdelay gibt die Anzahl der Zähler-Taktzyklen (Processdelay oder Zykluszeit) zu einer Prozessfrequenz zurück.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = Calc_Processdelay(frequency)
```

Parameter

frequency	Prozessfrequenz in Hertz.	__LONG
ret_val	Anzahl Prozesszyklen (= Processdelay).	LONG

Bemerkungen

- / -

Siehe auch

[Calc_TicksToNs](#)

Gültig für

CPU-T10, CPU-T11, CPU-T12, CPU-T9

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
Rem Processdelay für 150kHz einstellen  
Processdelay = Calc_Processdelay(150000)
```

Calc_TicksToNs berechnet aus eine Anzahl Zähler-Taktzyklen die zugehörige Zeit in Nanosekunden.

Syntax

```
#Include ADwinPro_All.Inc
ret_val = Calc_TicksToNs(ticks)
```

Parameter

ticks	Anzahl Zähler-Taktzyklen.	LONG
ret_val	Zeit in Nanosekunden	FLOAT

Bemerkungen

Es wird berücksichtigt, dass die Zähler-Taktzyklen von der Prozesspriorität abhängig ist.

Siehe auch

[Calc_Processdelay](#)

Gültig für

CPU-T10, CPU-T11, CPU-T12, CPU-T9

Beispiel

```
#Include ADwinPro_All.Inc
Dim time As Float
```

Event:

```
Rem Dauer eines Programmabschnitts feststellen
Par_1 = Read_Timer()
Rem Programmabschnitt
Rem ...
Par_2 = Read_Timer()
time = Calc_TicksToNs(Par_2 - Par_1)
```

Calc_TicksToNs

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).

Module mit D-Sub-Buchsen verfügen in der Regel 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 **P2_Event_Enable** nicht gesperrt werden. Der Event-Eingang an allen anderen Modulen ist nach dem Einschalten der Pro-Hardware 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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-TiCo Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo 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, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 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 ein Event-Signal nach der Flanke anliegen muss, damit es 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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-TiCo Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo 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, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 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_Event_Config



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_enable)
```

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_enable	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_Enable** konfiguriert. **P2_Event_Enable** gibt das resultierende Event-Signal für die Steuerung des Event-Prozesses frei.

Je nach Modul stehen verschiedene Modi **mode** zur Verfügung:

Modul	Verfügbare Modi
Pro-Aln-F-8/14-D	0, 1, 2, 3
Pro-Aln-F-8/16-D	0, 1, 2, 3
Pro-Aln-F-8/18-D	0, 1

Die Modi arbeiten wie folgt:

mode=0: Das Modul verwendet das Signal am Eingang **EVENT / A** als resultierendes Event-Signal. Der Parameter **edge_enable** hat keine Bedeutung.

mode=1: Der Eingang **EVENT/A** ist zunächst gesperrt, das resultierende Signal ist TTL-Pegel low. Sobald ein einzelner Freigabeimpuls vom Typ **edge_enable** am Eingang **ENABLE** eingegangen ist, gibt das Modul den Eingang **EVENT/A** frei und verwendet dessen Signal als resultierendes Event-Signal.

Der Eingang **EVENT/A** bleibt freigeschaltet, auch wenn sich das Signal am Eingang **ENABLE** ändert. Um den Freigabeimpuls erneut zu verwenden, rufen Sie **P2_Event2_Config** mit **mode**=0 auf und anschließend mit **mode**=1.

mode=2: 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** (siehe **P2_Event_Config**) 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.

`mode=3`: Der Eingang `EVENT/A` ist zunächst gesperrt, das resultierende Signal ist TTL-Pegel low. Sobald ein einzelner Impuls vom Typ `edge_enable` am Eingang `ENABLE` eingegangen ist, gibt das Modul das resultierende Event-Signal aus den Rechtecksignalen an den Eingängen `EVENT / A` und `B` (siehe [Signal aus AB-Modus](#)) frei.

Die Eingänge `EVENT / A` und `B` bleiben freigeschaltet, auch wenn sich das Signal am Eingang `ENABLE` ändert. Um den Freigabeimpuls erneut zu verwenden, rufen Sie `P2_Event2_Config` mit `mode=0` auf und anschließend mit `mode=3`.

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	Eingestellte Moduladresse (1...15).	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	31:3	2	1	0
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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-TiCo Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo 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, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, CAN-2 Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 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)
```


P2_Sync_All startet auf mehreren angegebenen Modulen synchron eine bestimmte Aktion.

Syntax

```
#Include ADwinPro_All.Inc

P2_Sync_All (module_pattern)
```

Parameter

module_pattern Bitmuster zum Ansprechen der Moduladressen, die **_LONG** synchron starten sollen:
 Bit = 0: Modul ignorieren.
 Bit = 1: Modul synchron starten.

Bits in pattern	31:15	14	13	...	1	0
Moduladresse	–	15	14	...	2	1

Bemerkungen

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

Modultyp	Aktion	entspricht
Analoge Eingänge	A/D-Wandlung auf allen freigegebenen ADC starten. Die Wandlung ist eine Einzelmessung. Für Burst-Messreihen siehe P2_Sync_Mode . Aln-F-x/14: für alle Kanäle wird der aktuell gewandelte Wert zwischengespeichert.	P2_Start_Conv / P2_Start_ConvF - / -
Analoge Ausgänge	D/A-Wandlung auf allen freigegebenen DAC starten mit dem Wert aus dem DAC-Register.	P2_Start_DAC
Digitale Eingänge	Aktuellen Zustand der Eingänge in das Eingangs-Zwischenregister übertragen. Register lesen: P2_Dig_Read_Latch .	P2_Start_DAC
Digitale Ausgänge	Wert aus dem Ausgangs-Zwischenregister lesen und auf die digitalen Ausgänge schalten. Register schreiben: P2_Dig_Write_Latch oder P2_PWM_Write_Latch .	P2_Dig_Latch / P2_PWM_Latch
Zähler	Aktuelle Zählerstände in die Zähler-Zwischenregister übertragen. Register lesen z. B. P2_Cnt_Read_Latch oder P2_Cnt_Get_PW .	P2_Cnt_Latch
Temperatur	Aktuelle Werte an den Kanälen in Zwischenregister übertragen. Register lesen z. B. P2_TC_Read_Latch .	P2_TC_Latch
Multi-I/O	Das Synchronsignal bezieht sich auf alle freigegebenen Gruppen: – Analoge Eingänge – Analoge Ausgänge – Digitale Ein- und Ausgänge – Zähler Die ausgelösten Aktionen sind wie oben beschrieben.	P2_Start_Conv P2_Start_DAC P2_MIO_Dig_Latch P2_Cnt_Latch

Als Voreinstellung nehmen alle Ein- oder Ausgänge der gewählten Module an der Aktion teil. Mit **P2_Sync_Enable** können Sie bei bestimmten Modulen ein

P2_Sync_All

oder mehrere Ein- oder Ausgänge für die Synchronaktion sperren oder freigeben.

Nur bei Aln-F-4/14 und Aln-F-8/14: Alle zwischengespeicherten Werte müssen auf einmal mit einem einzigen Befehl gelesen werden:

P2_Read_ADCF_4, **P2_Read_ADCF_8**, **P2_Read_ADCF_4_Packed**, **P2_Read_ADCF_8_Packed**.

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.

Siehe auch

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

[P2_Start_Conv](#), [P2_Start_ConvF](#), [P2_Read_ADCF4](#), [P2_Read_ADCF8](#), [P2_Start_DAC](#), [P2_Burst_Start](#)

[P2_Dig_Latch](#), [P2_PWM_Latch](#), [P2_TC_Latch](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-TiCo Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo 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, AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TC-8-ISO Rev. E, TRA-16 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 Kanäle oder Funktionsgruppen, die aktiviert oder deaktiviert werden: LONG |

Bit = 0: deaktivieren.
Bit = 1: aktivieren

Analog-Module: Aln-F-x/16 Rev. E, Aln-F-x/18 Rev. E, AOut-x/16 Rev. E

Bits in channel	31:8	7	6	5	4	3	2	1	0
Kanalnr.	–	8	7	6	5	4	3	2	1

Digital-Module: DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-8-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, PWM-16 Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Bits in channel	31:2	0
Funktionsgruppe	–	digitale Kanäle

Multi-I/O-Module: MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Bits in channel	31:4	3	2	1	0
Funktionsgruppe	–	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; gleiches gilt für Funktionsgruppen.

Bei Digitalmodulen können Sie keine einzelnen Kanäle auswählen, nur alle Kanäle gemeinsam.

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

Siehe auch

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

Gültig für

Aln-F-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 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

[Aln-F-4/16 Rev. E](#), [Aln-F-4/18 Rev. E](#), [Aln-F-8/16 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 Eingänge, Ausgänge oder Funktionsgruppen: Bit = 0: deaktiviert. Bit = 1: aktiviert.	LONG

Analog-Module: Aln-F-x/16 Rev. E, Aln-F-x/18 Rev. E, AOut-x/16 Rev. E

Bits in channel	31:8	7	6	5	4	3	2	1	0
Kanalnr.	–	8	7	6	5	4	3	2	1

Digital-Module: DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-8-D12 Rev. E, OPT-16 Rev. E, OPT-32 Rev. E, PWM-16 Rev. E, REL-16 Rev. E, TRA-16 Rev. E

Bits in channel	31:2	0
Funktionsgruppe	–	digitale Kanäle

Multi-I/O-Module: MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Bits in channel	31:4	3	2	1	0
Funktionsgruppe	–	Zähler	analoge Eingänge	analoge Ausgänge	digitale Kanäle

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-4/16 Rev. E, Aln-F-4/18 Rev. E, Aln-F-8/16 Rev. E, Aln-F-8/18 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, PWM-16(-I) Rev. E, PWM-16(-I) Rev. E, REL-16 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 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: Digitalkanäle der CPU

Dieser Abschnitt beschreibt Befehle für die Digitalein- und -ausgänge an CPU-Modulen:

- [CPU_Digin](#) (Seite 22)
- [CPU_Digout](#) (Seite 23)
- [CPU_Dig_IO_Config](#) (Seite 24)
- [CPU_Event_Config](#) (Seite 25)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

CPU_Digin

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

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](#)

Gültig für

[CPU-T11](#), [CPU-T12](#)

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_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](#), [CPU-T12](#)

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
Rem DIG I/O-0 und DIG I/O-1 als Ausgang einstellen
CPU_Dig_IO_Config(110011b)
```

Event:

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

CPU_Digout

CPU_Dig_IO_Config

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 Konfiguration der DIG I/O-Kanäle. Je 2 LONG | Bits konfigurieren jeweils einen Kanal. Die übrigen Bits haben keine Funktion.

Bits in pattern	31:6	5:4	3:2	1:0
gelten für Kanal	—	DIG I/O-1	—	DIG I/O-0
Bitkombination für Konfiguration einen Kanal				
00	Kanal arbeitet als Eingang für fallende Flanken.			
10	Kanal arbeitet als Eingang für steigende Flanken.			
01 oder 11	Kanal arbeitet als Ausgang.			

Bemerkungen

Der Flankentyp kann nur für Eingänge eingestellt werden.

Die DIG I/O-Kanäle sind mit Pull-up-Widerständen bestückt; um eine steigende Flanke erkennen zu können, muss der Eingang daher erst aktiv auf das Signal TTL low gezogen werden.

Nach einem Neustart sind die DIG I/O-Kanäle als Eingang für fallende Flanken konfiguriert; das entspricht der Bitkombination **00**.

Siehe auch

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

Gültig für

[CPU-T11](#), [CPU-T12](#)

Beispiel

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

Init:

```
Rem DIG I/O-0 als Eingang mit steigender Flanke einstellen,
Rem DIG I/O-1 als Ausgang
CPU_Dig_IO_Config(110010b)
Rem Statusmeldung an DIG I/O 0 lesen und dadurch zurücksetzen
dummy = CPU_Digin(0)
Rem ...
```

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 –, können Sie 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](#)

Gültig für

[CPU-T11](#), [CPU-T12](#)

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 ...
```

CPU_Event_Config

3.3 Pro II: Multi-I/O

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

- [P2_MIO_Dig_Latch](#) (Seite 27)
- [P2_MIO_Dig_Read_Latch](#) (Seite 28)
- [P2_MIO_Dig_Write_Latch](#) (Seite 29)
- [P2_MIO_Digin_Long](#) (Seite 30)
- [P2_MIO_Digout](#) (Seite 31)
- [P2_MIO_Digout_Long](#) (Seite 32)
- [P2_MIO_DigProg](#) (Seite 33)
- [P2_MIO_Get_Digout_Long](#) (Seite 34)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

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

Wir empfehlen, die Kanäle zunächst mit der Anweisung **P2_MIO_DigProg** als Eingänge oder Ausgänge zu programmieren; davon ausgenommen sind TRA- und OPT-Kanäle.

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_DigProg](#), [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](#), [MIO-D12 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
Rem Beispiel für MIO-4 / MIO-4-ET1
```

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_Latch

P2_MIO_Dig_Read_Latch

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

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

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

MIO-D12 Rev. E

Bitnr.	31:12	11:0
Eingang	–	OPT11:OPT0

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, MIO-D12 Rev. E

Beispiel

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

Init:

```
Rem Kanäle 7:0 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_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

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

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

MIO-D12 Rev. E

Bitnr.	31:12	27:16	15:0
Eingang	–	TRA11:TRA0	–

Bemerkungen

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

Sie können digitale Ausgänge mit der Anweisung **P2_MIO_Digout** direkt 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, MIO-D12 Rev. E

Beispiel

```
#Include ADwinPro_All.inc

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

P2_MIO_Digin_Long

P2_MIO_Digin_Long gibt den Zustand der Eingänge 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

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

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

MIO-D12 Rev. E

Bitnr.	31:12	11:0
Eingang	–	OPT11:OPT0

Bemerkungen

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

Siehe auch

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

Gültig für

[MIO-4 Rev. E](#), [MIO-4-ET1 Rev. E](#), [MIO-D12 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
P2_MIO_Digprog(1,00b) 'Kanäle 7:0 als Eingang
```

Event:

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

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. MIO-4 Rev. E, MIO-4-ET1 Rev. E 0...7: DIO-Kanäle. 24...27: TRA-Ausgänge. MIO-D12 Rev. E 16...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 TRA-Kanäle.

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](#), [MIO-D12 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 = 100b) 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

P2_MIO_Digout_Long

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

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

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

MIO-D12 Rev. E

Bitnr.	31:12	27:16	15:0
Eingang	–	TRA11:TRA0	–

Bemerkungen

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

Siehe auch

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

Gültig für

[MIO-4 Rev. E](#), [MIO-4-ET1 Rev. E](#), [MIO-D12 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_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.	–	7:4	3:0

Bemerkungen

Nach dem Einschalten des Systems sind die Kanäle 0...7 als Eingänge 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-Kanäle sind fest als Eingänge und TRA-Kanäle fest als Ausgänge konfiguriert; sie können nicht geändert 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_DigProg

P2_MIO_Get_Digout_Long

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

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

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

MIO-D12 Rev. E

Bitnr.	31:12	27:16	15:0
Eingang	–	TRA11:TRA0	–

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, MIO-D12 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Event:

```
Rem Bits 31:0 aus dem Latch zurücklesen
Par_1 = P2_MIO_Get_Digout_Long(1)
```

3.4 Pro II: Analoge Eingänge

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

Analogue Eingänge mit Multiplexer

- [P2_ADC](#) (Seite 37)
- [P2_ADC24](#) (Seite 38)
- [P2_ADC_Read_Limit](#) (Seite 39)
- [P2_ADC_Set_Limit](#) (Seite 41)
- [P2_Read_ADC](#) (Seite 42)
- [P2_Read_ADC24](#) (Seite 43)
- [P2_Read_ADC_SConv](#) (Seite 44)
- [P2_Read_ADC_SConv24](#) (Seite 45)
- [P2_SE_Diff](#) (Seite 46)
- [P2_Seq_Init](#) (Seite 47)
- [P2_Seq_Read](#) (Seite 50)
- [P2_Seq_Read24](#) (Seite 51)
- [P2_Seq_Read_Packed](#) (Seite 53)
- [P2_Seq_Start](#) (Seite 54)
- [P2_Seq_Wait](#) (Seite 55)
- [P2_Set_Mux](#) (Seite 56)
- [P2_Start_Conv](#) (Seite 57)
- [P2_Wait_EOC](#) (Seite 58)
- [P2_Wait_Mux](#) (Seite 59)

Analogue Eingänge mit Fast-ADC und Burst-Messreihe

- [P2_Burst_CRead_Unpacked1](#) (Seite 60)
- [P2_Burst_CRead_Unpacked2](#) (Seite 62)
- [P2_Burst_CRead_Unpacked4](#) (Seite 64)
- [P2_Burst_CRead_Unpacked8](#) (Seite 66)
- [P2_Burst_Init](#) (Seite 76)
- [P2_Burst_Read_Index](#) (Seite 79)
- [P2_Burst_Read](#) (Seite 81)
- [P2_Burst_Read_Unpacked1](#) (Seite 84)
- [P2_Burst_Read_Unpacked2](#) (Seite 86)
- [P2_Burst_Read_Unpacked4](#) (Seite 88)
- [P2_Burst_Read_Unpacked8](#) (Seite 90)
- [P2_Burst_Reset](#) (Seite 92)
- [P2_Burst_Start](#) (Seite 94)
- [P2_Burst_Status](#) (Seite 95)
- [P2_Burst_Stop](#) (Seite 96)
- [P2_Set_Average_Filter](#) (Seite 97)

Analoge Eingänge mit Fast-ADC

- [P2_ADCF \(Seite 98\)](#)
- [P2_ADCF24 \(Seite 99\)](#)
- [P2_ADCF_Mode \(Seite 100\)](#)
- [P2_ADCF_Read_Limit \(Seite 102\)](#)
- [P2_ADCF_Set_Limit \(Seite 103\)](#)
- [P2_ADCF_Reset_Min_Max \(Seite 104\)](#)
- [P2_ADCF_Read_Min_Max4 \(Seite 105\)](#)
- [P2_ADCF_Read_Min_Max8 \(Seite 107\)](#)
- [P2_Read_ADCF \(Seite 109\)](#)
- [P2_Read_ADCF24 \(Seite 110\)](#)
- [P2_Read_ADCF4 \(Seite 111\)](#)
- [P2_Read_ADCF4_24B \(Seite 112\)](#)
- [P2_Read_ADCF8 \(Seite 113\)](#)
- [P2_Read_ADCF8_24B \(Seite 114\)](#)
- [P2_Read_ADCF4_Packed \(Seite 115\)](#)
- [P2_Read_ADCF8_Packed \(Seite 116\)](#)
- [P2_Read_ADCF32 \(Seite 117\)](#)
- [P2_Read_ADCF_SConv \(Seite 118\)](#)
- [P2_Read_ADCF_SConv24 \(Seite 119\)](#)
- [P2_Read_ADCF_SConv32 \(Seite 120\)](#)
- [P2_Set_Gain \(Seite 121\)](#)
- [P2_Start_ConvF \(Seite 122\)](#)
- [P2_Wait_EOCF \(Seite 123\)](#)

Die Befehlsübersicht nach Modulen (Anhang A.2 Befehlsübersicht nach Modulen) zeigt, welche Befehle für einen Modultyp anwendbar sind.

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

Mehrere Wandlungen können schneller als mit **P2_ADC** durchgeführt werden, wenn Sie die Einzelfunktionen geschickt einsetzen, siehe Wartezeiten nutzen (Seite 148).

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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#define value Par_1
```

Event:

```
Rem 16Bit-Wert am analogen Eingang 4 messen
value = P2_ADC(1, 4)
FPar_1 = (value - 8000h) * 20 / 10000h 'Wert in Volt
```

P2_ADC

P2_ADC24

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 = $2^{24}-1$); bei 18 Bit-ADC sind die 6 niederwertigsten Bits immer gleich 0.	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				A/D-Wandlung starten		Ende der Wandlung abwarten		Gewandelten Wert auslesen

Mehrere Wandlungen können schneller als mit **P2_ADC24** durchgeführt werden, wenn Sie die Einzelfunktionen geschickt einsetzen, siehe Wartezeiten nutzen (Seite 148).

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

Die Verstärkung ist immer 1, die Einstellung mit **P2_Set_Mux** hat keine Auswirkung.

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 23:6 des Rückgabewerts; hier ist der Messwert um 6 Bits nach links verschoben und die Bits 5:0 sind Null.

Bitnr.	31:24	23:6	5:0
Inhalt	0	18-Bit Messwert	0

Siehe auch

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

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#define value Par_1
```

Event:

```
Rem 24Bit-Wert am analogen Eingang 4 messen
value = P2_ADC24(1, 4)
FPar_1 = (value - 800000h) * 20 / 1000000h 'Wert in Volt
```

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	Bitnr.	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	Bitnr.	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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

P2_ADC_Read_Limit

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 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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_Read_ADC

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_Set_Mux](#), [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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_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 23:6 des Rückgabewerts; hier ist der Messwert um 6 Bits nach links verschoben und die Bits 5:0 sind Null.

Bitnr.	31:24	23:6	5:0
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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_ADC24

P2_Read_ADC_SConv

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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_Read_ADC_SConv24

P2_SE_Diff

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-16/18-C Rev. E](#), [Aln-32/18-D Rev. E](#), [Aln-32/18-D-Ti-Co Rev. E](#), [MIO-4 Rev. E](#), [MIO-4-ET1 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_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 mode = 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 (nur für die Modi 1...3), das die zu wandelnden Kanäle als 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 für die Einschwingzeit der Ablaufsteuerung: 0: Werkseinstellung (125 = 2,5µs, Default). 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 Messgruppe.

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.

P2_Seq_Init

Modus	Art der Messung
3 continuous max:	<p>Die Ablaufsteuerung wandelt ununterbrochen neue Messwerte an den gewählten Kanälen, d.h. Wandlung und Prozesszyklus laufen asynchron.</p> <p>Die Wandlung wird mit P2_Seq_Start gestartet. Im Prozesszyklus wird mit P2_Seq_Read der jeweils neueste Messwert gelesen.</p>

Beachten Sie beim Modus 2 (continuous): Die Synchronisierung geschieht nur einmal und gilt nur für die gerade eingestellte Zykluszeit (**Processdelay**); das **Processdelay** muss gesetzt werden, bevor **Seq_Init** ausgeführt wird. 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.

Der Befehl ist nicht für Libraries geeignet, es sei denn, der Parameter **mode** wird als Konstante übergeben, und es wird einer der Modi 0, 1, 3 gewählt.

Bei Modul Aln-16/18-C Rev. E stellt **gain** keine Spannungs- sondern Strombereiche ein:

- 0: Faktor = 1, Strombereich -20mA...+20mA.
- 1: Faktor = 2, Strombereich -10mA...+10mA.
- 2: Faktor = 4, Strombereich -5mA...+5mA.
- 3: Faktor = 8, Strombereich -2,5mA...+2,5mA.

Sie können in der Messgruppe **channels** 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. Nicht ausgewählte Kanäle werden übersprungen, ihnen ist kein Messwert zugeordnet.

Die weiteren **Seq**-Befehle beziehen sich immer und ausschließlich auf die Kanäle der Messgruppe. Wenn also die Messgruppe aus 3 Kanälen besteht, können auch nur 3 Messwerte abgeholt werden.

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

Der Zeitabstand zwischen 2 Wandlungen der Ablaufsteuerung ist gleich der Einschwingzeit.

Tendenziell ergeben kürzere Einschwingzeiten ungenauere und längere Einschwingzeiten genauere Messergebnisse. Der angegebene Wertebereich soll daher nicht unterschritten werden.

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

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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E



Beispiel

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

Dim Data_1[16] As Long At DM_Local

Init:
    Processdelay = 300000      'T11: 1ms
    P2_SE_Diff(module, 0)      'Eingänge auf single ended stellen
    Rem Ablaufsteuerung: Modus continuous,
    Rem Verstärkungsfaktor 1 (-10V..+10V / -20mA..+20mA),
    Rem Kanäle 2,4,6,8, Standard-Einschwingzeit
    P2_Seq_Init(module, 2, 0, 0AAh, 0)
    Rem Messesequenzen auf dem Modul starten
    P2_Seq_Start(Shift_Left(1, module-1))

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

P2_Seq_Read

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. Es sollen nur so viele Messwerte gelesen werden, wie Kanäle in der Messgruppe sind.	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden.	ARRAY LONG 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 und eine Messgruppe festgelegt wurde.

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

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

Siehe auch

[P2_Seq_Init](#), [P2_Seq_Read24](#), [P2_Seq_Read_Packed](#), [P2_Seq_Start](#), [P2_Seq_Wait](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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: Modus continuous max, 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_Read(module, 16, Data_1, 1)
```

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

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl (1...32) der zu lesenden Messwerte. Es sollen nur so viele Messwerte gelesen werden, wie Kanäle in der Messgruppe sind.	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden.	ARRAY LONG 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 und eine Messgruppe festgelegt wurde.

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

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

Wenn ein Messwert eine geringere Auflösung 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 23:6 des Rückgabewerts; hier ist der Messwert um 6 Bits nach links verschoben und die Bits 5:0 sind Null.

Bitnr.	31:24	23:6	5:0
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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

P2_Seq_Read24

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: Modus continuous max, 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 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). Es sollen nur so viele Messwerte gelesen werden, wie Kanäle in der Messgruppe sind.	LONG
array[]	Ziel-Feld, in das die Messwerte-Paare übertragen werden.	ARRAY LONG 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 und eine Messgruppe festgelegt wurde.

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

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

Siehe auch

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

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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 Modus continuous max, 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_Read_Packed

P2_Seq_Start

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	...	1	0
Moduladresse	–	15	14	...	2	1

Bemerkungen

- / -

Siehe auch

[P2_Seq_Init](#), [P2_Seq_Read](#), [P2_Seq_Read24](#), [P2_Seq_Read_Packed](#), [P2_Seq_Wait](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_Wait wartet, bis die Ablaufsteuerung auf dem angegebenen Modul alle Kanäle der Messgruppe gewandelt und gespeichert hat.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_Seq_Wait(module)
```

Parameter

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

Bemerkungen

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

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_Read_Packed](#), [P2_Seq_Start](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

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

```
#Define channels 16        'Anzahl Messkanäle ggf. anpassen
```

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

```
Dim Data_2[channels] 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, Anzahl Messkanäle als Bitmuster,
```

```
  Rem Standard-Einschwingzeit
```

```
  P2_Seq_Init(module,1,0,(2^channels-1),0)
```

```
  P2_Seq_Start(Shift_Left(1,module-1)) 'Messesequenz starten
```

Event:

```
  P2_Seq_Wait(module) 'Ende der Messung abwarten
```

```
  P2_Seq_Read(module,channels,Data_1,1) 'diff. Kanäle einlesen
```

```
  For i=1 To channels
```

```
    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)) 'Messesequenz starten
```

P2_Seq_Wait

P2_Set_Mux

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 9:8 stellen die Verstärkung ein, die Bits 4:0 die Nummer des Eingangs.	LONG

Bitnr.							
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 das Hex-Format den angehängten Buchstaben **h** und für das Binärformat den Buchstaben **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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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
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
value1 = P2_Read_ADC(1) 'Wert vom ADC einlesen
```

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

Die Wandlung kann synchron mit Aktionen auf anderen Modulen gestartet werden. Verwenden Sie hierzu den Befehl **P2_Sync_All**.

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_Read_ADC](#), [P2_Set_Mux](#), [P2_Wait_EOC](#), [P2_Sync_All](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_Start_Conv

P2_Wait_EOC

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_Set_Mux](#), [P2_Start_Conv](#), [P2_Read_ADC](#)

Gültig für

Aln-16/18-8B Rev. E, Aln-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_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-16/18-C Rev. E, Aln-32/18-D Rev. E, Aln-32/18-D-Ti-Co Rev. E, Aln-8/18 Rev. E, Aln-8/18-8B Rev. E, Aln-8/18-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 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_Wait_Mux

P2_Burst_CRead_Unpacked1

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. 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

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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. **count** sollte etwa um den Faktor 2 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der durchzuführenden Messungen (**buffer_count**).

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_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](#)



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 mit 250ns (Fx16) / 500ns
  Rem (Fx14) Periodendauer, 2^26 Daten speichern ab Adresse 0.
  P2_Burst_Init (module,1,0,67108864,25,010b)
  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

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.	ARRAY
	Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	LONG FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
flowrate	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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 (**buffer_count**).

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

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_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](#)



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 mit 320ns(Fx16) / 640ns
Rem (Fx14) Periodendauer, 2^26 Messwerte je Kanal ab Adresse 0.
P2_Burst_Init (module,3,0,67108860,32,010b)
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

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.	ARRAY
	Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	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

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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 (*buffer_count*).

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



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 mit 320ns(Fx16) / 640ns
  Rem (Fx14) Periodendauer, 2^25 je Kanal speichern ab Adresse 0.
  P2_Burst_Init (module,15,0,3355444,32,010b)
  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

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.	ARRAY
	Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	LONG
		FLOAT
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
flowrate	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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 (**buffer_count**).

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

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_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



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 mit 250ns(Fx16) / 500ns
  Rem (Fx14) Periodendauer, 1 Mio. Messwerte je Kanal speichern
  Rem ab Adresse 100
  P2_Burst_Init (module,255,100,1000000,25,010b)
  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_CRead_Pos_Unpacked1

P2_Burst_CRead_Pos_Unpacked1 kopiert eine Anzahl Messwerte von einem Kanal ab einer Speicherposition im Modulspeicher in ein Feld.

Wenn beim Lesen der Messwerte das Ende des Speicherabschnitts erreicht wird, werden die weiteren Messwerte vom Beginn des Abschnitts an gelesen.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Pos_Unpacked1(module, buffer_start,
    buffer_count, count, startadr, array1[],
    array_idx, flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
buffer_start	Startadresse des Speicherabschnitts, aus dem gelesen wird.	LONG
buffer_count	Anzahl der speicherbaren Messungen pro Kanal im Speicherabschnitt, aus dem gelesen wird.	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal.	LONG
startadr	Startadresse innerhalb des Speicherabschnitts, ab der Messwerte gelesen werden.	LONG
array1[]	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 niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Wir empfehlen, während der Entwicklungsphase eines Programms den Debug-Modus einzuschalten. Sie erhalten dann Hinweise auf Programmfehler.

Der Befehl soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit einem Kanal eingerichtet wurde (siehe **P2_Burst_Init**, Parameter **mode**, **channels**).

Die Angaben zum Speicherabschnitt **buffer_start** und **buffer_count** unterstützen u. a. die Aufteilung des Modulspeichers in mehrere Abschnitte. Geben Sie – auch wenn Sie den Modulspeicher ohne Aufteilung nutzen – die gleichen Werte an wie beim Einrichten des Speicherabschnitts mit **P2_Burst_Init**.

Die Anweisung liest die Anzahl **count** Messwerte ab der Adresse **startadr**. Wenn die Burst-Messreihe weiter läuft, sollte die Anzahl **count** mindestens um den Faktor 10 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der Messungen (**buffer_count**).

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

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_CRead_Unpacked1, P2_Burst_CRead_Pos_Unpacked2, P2_Burst_CRead_Pos_Unpacked4, P2_Burst_CRead_Pos_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

Beispiel

- / -

P2_Burst_CRead_Pos_Unpacked2

P2_Burst_CRead_Pos_Unpacked2 kopiert eine Anzahl Messwerte von 2 Kanälen ab einer Speicherposition im Modulspeicher in 2 Felder.

Wenn beim Lesen der Messwerte das Ende des Speicherabschnitts erreicht wird, werden die weiteren Messwerte vom Beginn des Abschnitts an gelesen.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Pos_Unpacked2(module, buffer_start,
    buffer_count, count, startadr, array1[],
    array2[], array_idx, flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
buffer_start	Startadresse des Speicherabschnitts, aus dem gelesen wird.	LONG
buffer_count	Anzahl der speicherbaren Messungen pro Kanal im Speicherabschnitt, aus dem gelesen wird.	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal.	LONG
startadr	Startadresse innerhalb des Speicherabschnitts, ab der Messwerte gelesen werden.	LONG
arrayx[]	Ziel-Felder für die Messwerte der Kanäle 1...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 niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Wir empfehlen, während der Entwicklungsphase eines Programms den Debug-Modus einzuschalten. Sie erhalten dann Hinweise auf Programmfehler.

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

Die Angaben zum Speicherabschnitt **buffer_start** und **buffer_count** unterstützen u. a. die Aufteilung des Modulspeichers in mehrere Abschnitte. Geben Sie – auch wenn Sie den Modulspeicher ohne Aufteilung nutzen – die gleichen Werte an wie beim Einrichten des Speicherabschnitts mit **P2_Burst_Init**.

Die Anweisung liest die Anzahl **count** Messwerte ab der Adresse **startadr**. Wenn die Burst-Messreihe weiter läuft, sollte die Anzahl **count** mindestens um den Faktor 10 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der Messungen (**buffer_count**).

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

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_CRead_Unpacked2, P2_Burst_CRead_Pos_Unpacked1, P2_Burst_CRead_Pos_Unpacked4, P2_Burst_CRead_Pos_Unpacked8, P2_Burst_CRead_Pos_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

Beispiel

- / -

P2_Burst_CRead_Pos_Unpacked4

P2_Burst_CRead_Pos_Unpacked4 kopiert eine Anzahl Messwerte von 4 Kanälen ab einer Speicherposition im Modulspeicher in 4 Felder.

Wenn beim Lesen der Messwerte das Ende des Speicherabschnitts erreicht wird, werden die weiteren Messwerte vom Beginn des Abschnitts an gelesen.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Pos_Unpacked4(module, buffer_start,
    buffer_count, count, startadr, array1[],
    array2[], array3[], array4[],
    array_idx, flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
buffer_start	Startadresse des Speicherabschnitts, aus dem gelesen wird.	LONG
buffer_count	Anzahl der speicherbaren Messungen pro Kanal im Speicherabschnitt, aus dem gelesen wird.	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal.	LONG
startadr	Startadresse innerhalb des Speicherabschnitts, ab der Messwerte gelesen werden.	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 niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Wir empfehlen, während der Entwicklungsphase eines Programms den Debug-Modus einzuschalten. Sie erhalten dann Hinweise auf Programmfehler.

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

Die Angaben zum Speicherabschnitt **buffer_start** und **buffer_count** unterstützen die Aufteilung des Modulspeichers in mehrere Abschnitte. Geben Sie – auch wenn Sie den Modulspeicher ohne Aufteilung nutzen – die gleichen Werte an wie beim Einrichten des Speicherabschnitts mit **P2_Burst_Init**.

Die Anweisung liest die Anzahl **count** Messwerte ab der Adresse **startadr**. Wenn die Burst-Messreihe weiter läuft, sollte die Anzahl **count** mindestens um den Faktor 10 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der Messungen (**buffer_count**).

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

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_CRead_Unpacked4, P2_Burst_CRead_Pos_Unpacked1, P2_Burst_CRead_Pos_Unpacked2, P2_Burst_CRead_Pos_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

Beispiel

- / -

P2_Burst_CRead_Pos_Unpacked8

P2_Burst_CRead_Pos_Unpacked8 kopiert eine Anzahl Messwerte von 8 Kanälen ab einer Speicherposition im Modulspeicher in 8 Felder.

Wenn beim Lesen der Messwerte das Ende des Speicherabschnitts erreicht wird, werden die weiteren Messwerte vom Beginn des Abschnitts an gelesen.

Syntax

```
#Include ADwinPro_All.Inc

P2_Burst_CRead_Pos_Unpacked8(module, buffer_start,
    buffer_count, count, startadr, array1[],
    array2[], array3[], array4[], array5[], array6[],
    array7[], array8[], array_idx, flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
buffer_start	Startadresse des Speicherabschnitts, aus dem gelesen wird.	LONG
buffer_count	Anzahl der speicherbaren Messungen pro Kanal im Speicherabschnitt, aus dem gelesen wird.	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal.	LONG
startadr	Startadresse innerhalb des Speicherabschnitts, ab der Messwerte gelesen werden.	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 niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Wir empfehlen, während der Entwicklungsphase eines Programms den Debug-Modus einzuschalten. Sie erhalten dann Hinweise auf Programmfehler.

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

Die Angaben zum Speicherabschnitt **buffer_start** und **buffer_count** unterstützen die Aufteilung des Modulspeichers in mehrere Abschnitte. Geben Sie – auch wenn Sie den Modulspeicher ohne Aufteilung nutzen – die gleichen Werte an wie beim Einrichten des Speicherabschnitts mit **P2_Burst_Init**.

Die Anweisung liest die Anzahl **count** Messwerte ab der Adresse **startadr**. Wenn die Burst-Messreihe weiter läuft, sollte die Anzahl **count** mindestens um den Faktor 10 kleiner sein als die bei **P2_Burst_Init** festgelegte Anzahl der Messungen (**buffer_count**).

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

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_CRead_Unpacked8, P2_Burst_CRead_Pos_Unpacked1, P2_Burst_CRead_Pos_Unpacked2, P2_Burst_CRead_Pos_Unpacked4, P2_Burst_Start, P2_Burst_Status

Gültig für

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

Beispiel

- / -

P2_Burst_Init

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, buffer_start,  
              buffer_count, pulses, mode)
```

Parameter

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

channels legt Anzahl und Nummern der Messkanäle fest. Nur die unten aufgeführten Werte sind zulässig. LONG |

Gemeinsam mit der Speichergröße des Moduls ergibt sich die max. Anzahl der Messwerte pro Kanal:

channels	Kanalnr.	max. Anzahl der Messwerte pro Kanal für <code>buffer_start=0</code> :
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:

channels	Kanalnr.	Zeitkanal	max. Anzahl der Messwerte pro Kanal für <code>buffer_start = 0</code> :
131	1	2	67108860
143	1...3	4	33554428
127	1...7	8	16777212

buffer_start Startadresse ($0 \dots 67\,108\,860 = 2^{26} - 4$ Longs) des Speicherabschnitts im Modulspeicher. Die Adresse ist in Longs angegeben und muss durch 4 teilbar sein. LONG |

buffer_count Anzahl der speicherbaren Messungen pro Kanal im Speicherabschnitt; die Anzahl bestimmt die Größe des Speicherabschnitts. LONG |

Der Maximalwert für `buffer_count` wird durch `channels` (und die Startadresse) bestimmt. Die Anzahl muss durch 4 teilbar sein, bei `channels=1` (1 Kanal) durch 8 teilbar.

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

Aln-F-x/14: Periodendauer = **pulses** * 20ns.

Aln-F-x/16: Periodendauer = **pulses** * 10ns.

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

Nur *Aln-F-x/14*: Wertebereich 1 ... 65535; bei 8 Kanälen (**channels** = 255 oder 127) beginnt der Wertebereich mit 2.

Nur *Aln-F-x/16*: Wertebereich 25 ... 65535. Der Wert für **pulses** hängt vom Parameter **mode** bei **P2_Set_Average_Filter** ab:

mode (P2_Set_Average_Filter)	pulses_{min}
0	25
1	67
2	154
3	313
4	645
5	1333

mode Bitmuster, das die Art der Burst-Messreihe angibt: **LONG** |

Bitnr.	31:3	2	1	0
Funktion	–	Art der Taktrate: Bit = 0: Zeitgesteuert (siehe pulses). Bit = 1: Extern gesteuert (Event-Eingang).	Betriebsart der Burst-Messreihe: Bit = 0: Einfach Bit = 1: Kontinuierlich	–

Bemerkungen

Bei Modulen *Aln-F-x/16* ist der Befehl ab Revision E04 verfügbar.

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).

Es gibt 2 Betriebsarten:

- Einfache Burst-Messreihe: Das Modul erfasst eine feste Anzahl **buffer_count** von Messungen. Sobald alle Messwerte gewandelt sind, endet die Burst-Messreihe. Die Messdaten sollen mit **P2_Burst_Read_Unpacked...** gelesen werden.
- Kontinuierliche Burst-Messreihe: Das Modul wandelt dauernd Messwerte mit der festgelegten Periodendauer. Sie brechen die Messreihe mit **P2_Burst_Stop** ab und lesen mit **P2_Burst_CRead_...** die Messwerte aus. Das Modul speichert die Messwerte im (mit **P2_Burst_Init**) reservierten Speicherabschnitt in einem Ringspeicher, d.h. die jüngsten Messwerte überschreiben die jeweils ältesten Daten.

Betriebsart

Speicher in Abschnitte aufteilen

Taktrate

Achten Sie darauf, dass der Lesebefehl zu der eingestellten Anzahl der Kanäle passt, beispielsweise **P2_Burst_Read_Unpacked4** für `channels=15` (4 Kanäle).

Sie können den Modulspeicher in mehrere Abschnitte für verschiedene Burst-Messreihen einteilen. Nur einer der Speicherabschnitte kann jeweils aktiv, also mit **P2_Burst_Init** initialisiert sein und Messwerte speichern.

Die Länge `buffer_length` eines Speicherabschnitts in Longs ist

$$\text{buffer_length} = \frac{1}{2} \cdot \text{Anzahl Kanäle} \cdot \text{buffer_count}$$

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_Pos_Unpacked1](#), [P2_Burst_CRead_Pos_Unpacked2](#), [P2_Burst_CRead_Pos_Unpacked4](#), [P2_Burst_CRead_Pos_Unpacked8](#)

[P2_Burst_CRead_Unpacked1](#), [P2_Burst_CRead_Unpacked2](#), [P2_Burst_CRead_Unpacked4](#), [P2_Burst_CRead_Unpacked8](#), [P2_Set_Average_Filter](#)

[P2_Burst_Read](#), [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

Für Module Aln-F-x/14 siehe auch das Beispiel für Kontinuierliche Messwertwandlung (Pro II): "1 Kanal wandeln" auf [Seite 2099](#).

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

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

Init:

```
REM Initiate continuous burst sequence, channel 1 using 250ns/
REM /500ns period duration, save 2^26 samples from address 0
REM Period duration depends on module type, see parameter pulses
P2_Burst_Init (module, 1, 0, 67108864, 25, 010b)
REM Start burst sequence
pattern = Shift_Left(1, module-1) 'access single module only
P2_Burst_Start(pattern)
Processdelay = 10000000
```

Event:

```
REM Read last 1000 samples from channel (slowly) and store
REM in Data_1
P2_Burst_CRead_Unpacked1(module, 1000, Data_1, 1, 1)
```

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...67 108 860 = $2^{26} - 4$) im Modulspeicher. Die Adresse ist durch 4 teilbar.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

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:

$$n = 2 \cdot \frac{\text{ret_val} - \text{Startadresse}}{\text{Anzahl Kanäle}}$$

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 **buffer_start + buffer_count · (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

P2_Burst_Read_Index

Für Module Aln-F-x/14 siehe auch das Beispiel für Kontinuierliche Messwertwandlung (Pro II): "1 Kanal wandeln" auf [Seite 2099](#).

```
#Include ADwinPro_All.Inc

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

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

LowInit:
For i = 1 To buffer_count
    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, 500/1000ns Periodendauer
P2_Burst_Init(module, 15, 0, buffer_count, 50, 010b)
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 = buffer_count - 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 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...67 108 860 = $2^{26} - 4$) 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

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

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_Read_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_Read_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
...
Anzahl Kanäle: 1		

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

P2_Burst_Read



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
...

Anzahl Kanäle: 4

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	K8 / M1	K7 / M1
startadr+4	K2 / M2	K1 / M2
...

Anzahl Kanäle: 8

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

Für Module AIn-F-x/14 siehe auch das Beispiel für Kontinuierliche Messwertwandlung (Pro II): "1 Kanal wandeln" auf [Seite 2099](#).

```
#Include ADwinPro_All.Inc
REM Beispiel für AIn-F-x/14
#Define module 4
#Define buffer_count 500000 'Größe des Buffers
#Define channels 4           '1, 2, 4 oder 8 Kanäle
#Define frq_Hz 5000
#Define mem_idx Par_1
#Define count Par_2
#Define overflow Par_3

Dim Data_1[500000] As Long 'deutlich größer als Buffer
Dim i, prev_mem_idx, start_idx As Long

LowInit:
  For i = 1 To buffer_count
    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, n Kanäle, 500/1000ns Periodendauer
  P2_Burst_Init(module, 2^channels-1, 0, buffer_count, 50, 010b)
  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 Modulspeicher lesen
    P2_Burst_Read(module, count, prev_mem_idx, Data_1, start_idx, 0)
    Rem Start-Index in Data_1 für den nächsten Zyklus
    start_idx = start_idx + count
    prev_mem_idx = mem_idx 'Index im Buffer merken
  EndIf

  If (count < 0) Then
    Rem Anzahl der Messwerte bis zum Ende des Buffers
    count = buffer_count - prev_mem_idx * 2 / channels
    Rem Messwerte aus dem Modulspeicher lesen
    P2_Burst_Read(module, count, prev_mem_idx, Data_1, start_idx, 0)
    Rem Start-Index in Data_1 für den nächsten Zyklus
    start_idx = 1
    prev_mem_idx = 0          'Index im Buffer merken
    Inc(overflow)            'Überlaufzähler erhöhen
  EndIf
  Rem Messwerte liegen gepackt im Feld Data_1

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

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

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu übertragenden Messwerte. Die Anzahl muss durch 8 teilbar sein.	LONG
startadr	Startadresse ($0 \dots 67\,108\,860 = 2^{26} - 4$) 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 niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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 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_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



Für Module Aln-F-x/14 siehe auch das Beispiel für Kontinuierliche Messwertwandlung (Pro II): "1 Kanal wandeln" auf Seite 2099.

```
#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 300ns/
    Rem 600ns Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
    P2_Burst_Init (module,1,0,1000,30,0)
    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

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal. Die Anzahl muss durch 4 teilbar sein.	LONG
startadr	Startadresse ($0 \dots 67\,108\,860 = 2^{26} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse 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 niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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



Für Module Aln-F-x/14 siehe auch das Beispiel für Kontinuierliche Messwertwandlung (Pro II): "1 Kanal wandeln" auf Seite 2099.

```
#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 mit 400ns/800ns
  Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
  P2_Burst_Init (module,3,0,1000,40,0)
  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

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal. Die Anzahl muss durch 2 teilbar sein.	LONG
startadr	Startadresse (0...67 108 860 = $2^{26} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse muss durch 4 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 niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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 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_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



Für Module Aln-F-x/14 siehe auch das Beispiel für Kontinuierliche Messwertwandlung (Pro II): "1 Kanal wandeln" auf Seite 2099.

```
#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 mit 400ns/800ns
    Rem Periodendauer, 1000 Messwerte je Kanal ab Adr. 0 speichern
    P2_Burst_Init (module,15,0,1000,40,0)
    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 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

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu übertragenden Messwerte je Kanal.	LONG
startadr	Startadresse (0...67108860 = $2^{26} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse muss durch 4 teilbar sein.	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 niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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 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_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



Für Module Aln-F-x/14 siehe auch das Beispiel für Kontinuierliche Messwertwandlung (Pro II): "1 Kanal wandeln" auf Seite 2099.

```
#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 Kanäle 1..8 mit 280ns/560ns
Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern
P2_Burst_Init (module,255,0,1000,28,0)
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 je Kanal
    Rem (schnell) abholen und in Data_1 bis Data_8 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.

Bitnr.	31:15	14	13	...	1	0
Moduladresse	–	15	14	...	2	1

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

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 mit 250ns/500ns
    Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern
    P2_Burst_Init (module,1,0,1000,25,0)
    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.

Bitnr.	31:15	14	13	...	1	0
Moduladresse	–	15	14	...	2	1

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

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

Für Module Aln-F-x/14 siehe auch das Beispiel für Kontinuierliche Messwertwandlung (Pro II): "[1 Kanal wandeln](#)" auf [Seite 2099](#).

```
#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 300ns/600ns
Rem Periodendauer, 2^26 Daten speichern ab Adresse 0.
P2_Burst_Init (module,1,0,67108864,30,010b)
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

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

Der Befehl 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

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 mit 250ns/500ns
Rem Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
P2_Burst_Init (module,1,0,1000,25,0)
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)
If (rest = 0) Then state = 1 'Alle Messwerte liegen vor
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_Status

P2_Burst_Stop

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.

Bitnr.	31:15	14	13	...	1	0
Moduladresse	–	15	14	...	2	1

Bemerkungen

Bei Modulen Aln-F-x/16 ist der Befehl ab Revision E04 verfügbar.

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](#)

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 300ns/600ns Periodendauer, 2^26 Daten
Rem speichern ab Adresse 0
P2_Burst_Init (module,1,0,67108864,30,0)
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)
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 (Default). 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. Die Berechnungsmethode ist je nach Modultyp verschieden:

- Aln-F-x/14: gleitender Mittelwert.
Mit jedem neuen Messwert wird ein neuer Mittelwert (aus den letzten n Werten) gebildet.
- Aln-F-x/16: arithmetisches Mittel.
Für jeden Mittelwert wird erneut die festgelegte Anzahl an Messwerten gewandelt und gemittelt.

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

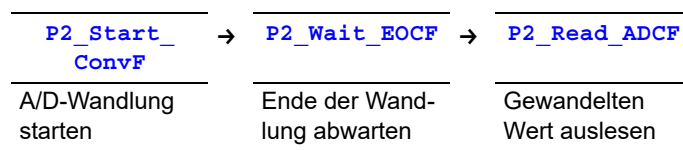
ret_val = P2_ADCF(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Nummer (1...4 oder 1...8) des analogen Eingangs.	LONG
ret_val	Wandlungsergebnis (0...65535); bei 14 Bit-ADC sind die 2 niederwertigsten Bits immer gleich 0.	LONG

Bemerkungen

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



Mehrere Wandlungen können schneller als mit **P2_ADCF** durchgeführt werden, wenn Sie die Einzelfunktionen geschickt einsetzen, siehe Wartezeiten nutzen (Seite 148).

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
#Define value Par_1
```

Event:

```
Rem 16Bit-Wert am analogen Eingang 4 messen
value = P2_ADCF(1, 4)
FPar_1 = (value - 8000h) * 20 / 10000h 'Wert in Volt
```

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

Syntax

```
#Include ADwinPro_All.Inc

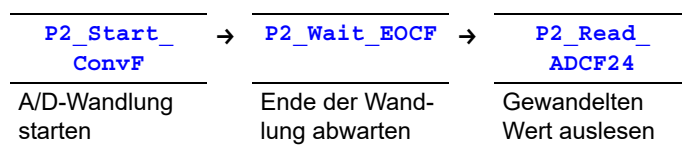
ret_val = P2_ADCF24(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Nummer (1...4 oder 1...8) des analogen Eingangs.	LONG
ret_val	Wandlungsergebnis (0...16777215 = $2^{24}-1$); bei 18 Bit-ADC sind die 6 niederwertigsten Bits immer gleich 0.	LONG

Bemerkungen

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



Mehrere Wandlungen können schneller als mit **P2_ADCF24** durchgeführt werden, wenn Sie die Einzelfunktionen geschickt einsetzen, siehe Wartezeiten nutzen (Seite 148).

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
#define value Par_1
```

Event:

```
Rem 24Bit-Wert am analogen Eingang 4 messen
value = P2_ADCF24(1, 4)
FPar_1 = (value - 800000h) * 20 / 1000000h 'Wert in Volt
```

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	...	1	0
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 AIn-F-4/16 und AIn-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. Der Befehl ist – außer im Standard-Modus – nicht für Libraries geeignet.

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 Les-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.

Der Timer-Modus sollte möglichst nur in Kombination mit einem einzigen hoch-prioritären 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.



Standard-Modus

Timer-Modus



Timer-Modus mit
Multiplex-Option

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

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
```

Event-Modus

Event-Modus mit Multiplex-Option



P2_ADCF_Read_Limit

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

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

In einer Burst-Messreihe erhalten Sie mit **P2_Burst_Read_Index_Limit** die Adresse des Messwerts, der außerhalb der Grenzwerte liegt.

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:

```
Rem für Aln-F-x/16 Rev. E und Aln-F-x/18 Rev. E die folgende
Rem Zeile aktivieren
' P2_ADCF_Mode(1,1)
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_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, adc_no, high, low)
```

Parameter

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

Bei den Modulen AIn-F-x/16 Rev. E und AIn-F-x/18 Rev. E ist dieser Befehl 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.

Die Grenzwerte werden auch dann registriert, wenn ein Eingang in einer Burst-Messreihe verwendet wird.

Siehe auch

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

Gültig für

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

Beispiel

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

Init:

```
Rem für AIn-F-x/16 Rev. E und AIn-F-x/18 Rev. E die folgende
Rem Zeile aktivieren
' P2_ADCF_Mode(1,1)
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_Set_Limit

P2_ADCF_Reset_Min_Max

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_pattern	Bitmuster zur Auswahl der Kanäle, auf denen die Extremwerte zurückgesetzt werden.	LONG

Bitnr.	15:8	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_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

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

Bemerkungen

Minimal- und Maximalwerte werden gesammelt, sobald Wandlungen ausgeführt werden. Zum Starten von Wandlungen sind z.B. die Befehle **P2_ADCF_Mode** mit Timer-Modus (Modul wandelt eigenständig) oder **P2_Start_ConvF** (einzelne Wandlung) geeignet.

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

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

P2_ADCF_Read_Min_Max4

Beispiel

```
#Include ADwinPro_All.inc
#Define module 3
Dim Data_10[8] As Long
Dim i As Long

Init:
    Rem start module timer mode (continuous AD conversions)
    P2_ADCF_Mode(Shift_Left(1, module - 1), 1)
    P2_ADCF_Reset_Min_Max(module,1111b)'reset all 4 F-ADC

Event:
    Rem read high and low values of F-ADC 1...4
    P2_ADCF_Read_Min_Max4(module,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(module,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 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

Minimal- und Maximalwerte werden gesammelt, sobald Wandlungen ausgeführt werden. Zum Starten von Wandlungen sind z.B. die Befehle **P2_ADCF_Mode** mit Timer-Modus (Modul wandelt eigenständig) oder **P2_Start_ConvF** (einzelne Wandlung) geeignet.

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

P2_ADCF_Read_Min_Max8

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 5
Dim Data_4[16] As Long
Dim i As Long

Init:
    Rem start module timer mode (continuous AD conversions)
    P2_ADCF_Mode(Shift_Left(1, module - 1), 1)
    P2_ADCF_Reset_Min_Max(module,11111111b) 'reset all 8 F-ADC

Event:
    Rem read high and low values of F-ADC 1...8
    P2_ADCF_Read_Min_Max8(module,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(module,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 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

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

Beispiel

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

Event:

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

P2_Read_ADCF

P2_Read_ADCF24

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_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

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. 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 erfordl. 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

P2_Read_ADCF4_24B

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 LONG
index	Index des Elements im Zielfeld, in dem der erste Messwert gespeichert wird.	LONG

Bemerkungen

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

Die Anweisung ist wesentlich schneller als das mehrfache Auslesen mit **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_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

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. 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 erfordl. 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

P2_Read_ADCF8_24B

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_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_ADCF4_Packed

P2_Read_ADCF8_Packed

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

AIn-F-8/14 Rev. E, AIn-F-8/16 Rev. E, AIn-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 AIn-F-8/14
P2_Start_ConvF(1, 0FFh)
```

Event:

```
Rem Warten auf Wandlung-Ende; nicht erforderlich für AIn-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 AIn-F-8/14
P2_Start_ConvF(1, 0FFh)
```


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_pair)
```

Parameter

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

adc_pair Kennziffer (1...2 oder 1...4) für das zu lesende F-ADC-Paar. LONG

adc_pair	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

Im unteren Wort steht das Wandlungsergebnis des ADC mit ungerader Nummer ($\text{adc_pair} \cdot 2 - 1$), im oberen Wort das Ergebnis des ADC mit gerader Nummer ($\text{adc_pair} \cdot 2$).

Die Nummer des ersten F-ADC ist 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_ADCF32

P2_Read_ADCF_SConv

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_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_SConv24

P2_Read_ADCF_SConv32

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_pair)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_pair	Nummer (1...2 oder 1...4) eines F-ADC-Paars:	LONG

adc_pair	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
----------------	---	------

Bemerkungen

Im unteren Wort steht das Wandlungsergebnis des ADC mit ungerader Nummer ($\text{adc_pair} * 2 - 1$), im oberen Wort das Ergebnis des ADC mit gerader Nummer ($\text{adc_pair} * 2$).

Die Nummer des ersten F-ADC ist 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_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_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 mode	Verstärkung 2^n	Messbereich $\pm 10V / 2^n$
0	1	$\pm 10V$
1	2	$\pm 5V$
2	4	$\pm 2,5V$
3	8	$\pm 1,25V$

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_Set_Gain

P2_Start_ConvF

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_pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_pattern	Bitmuster, das die ADC festlegt, deren Konvertierung gestartet werden soll (siehe Tabelle).	LONG

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

Bemerkungen

Die Angabe der ADC erfolgt mit einem Bitmuster, 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.

Beim Modul Aln-F-x/14 ist **P2_Start_ConvF** nicht erforderlich, weil die ADC kontinuierlich mit fest eingestellter Wandlungsrate arbeiten. Vorsicht: Wenn Sie den Befehl dennoch verwenden, wird der aktuelle Messwert in das Latch-Register kopiert; ein folgendes **P2_Read_ADCF** liest den kopierten (nicht den dann aktuellen) Wert, selbst wenn das viel später geschieht.

Sie können mit dem Befehl **P2_Sync_All** Wandlungen auf mehreren Modulen synchron starten. Dabei können Sie einzelne Kanäle mit **P2_Sync_Enable** für die Synchronisierung deaktivieren.

Sie können Wandlungen auf mehreren Modulen 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](#), [P2_Sync_Enable](#), [P2_Sync_Mode](#)

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_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

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

Bemerkungen

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

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
```

P2_Wait_EOCF

3.5 Pro II: Analoge Ausgänge

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit analogen Ausgängen gelten:

- [P2_DAC \(Seite 125\)](#)
- [P2_DAC4 \(Seite 126\)](#)
- [P2_DAC4_Packed \(Seite 127\)](#)
- [P2_DAC8 \(Seite 128\)](#)
- [P2_DAC8_Packed \(Seite 129\)](#)
- [P2_Start_DAC \(Seite 130\)](#)
- [P2_Write_DAC \(Seite 131\)](#)
- [P2_Write_DAC4 \(Seite 132\)](#)
- [P2_Write_DAC4_Packed \(Seite 133\)](#)
- [P2_Write_DAC8 \(Seite 134\)](#)
- [P2_Write_DAC8_Packed \(Seite 135\)](#)
- [P2_Write_DAC32 \(Seite 136\)](#)
- [P2_DAC1_DIO \(Seite 137\)](#)
- [P2_DAC_Ramp_Write \(Seite 138\)](#)
- [P2_DAC_Ramp_Status \(Seite 140\)](#)
- [P2_DAC_Ramp_Buffer_Free \(Seite 142\)](#)
- [P2_DAC_Ramp_Stop \(Seite 143\)](#)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

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)
```

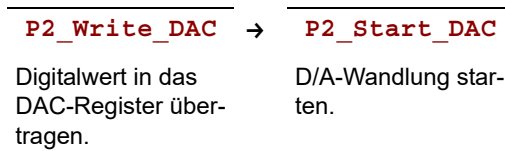
Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dac_no	Nummer (1, 1...4 oder 1...8) des Ausganges.	LONG
value	Auszugebender Wert (0...65535).	LONG

Bemerkungen

Bei mehreren DAC empfehlen wir, 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](#), [P2_DAC_Ramp_Write](#)

Gültig für

AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

```
Rem Digitaler P-Regler
#include ADwinPro_All.Inc
#define set_to Par_1      'Sollwert
#define gain Par_2        'Verstärkungsfaktor
#define dev Par_3         'Regelabweichung
#define actuate Par_4     'Stellgröße
```

Event:

```
dev = set_to - P2_ADC(1,1) 'Regelabweichung berechnen
actuate = dev * gain      'Stellgröße berechnen
P2_DAC(1,1,actuate)      'Ausgabe der Stellgröße
```

P2_DAC

P2_DAC4

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

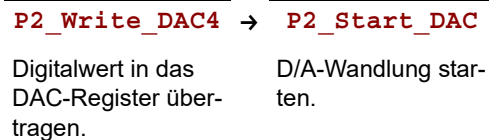
P2_DAC4(module,array[],index)
```

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-4/16-TiCo Rev. E](#), [AOut-8/16 Rev. E](#), [AOut-8/16-TiCo Rev. E](#), [MIO-4 Rev. E](#), [MIO-4-ET1 Rev. E](#)

Beispiel

Rem Digitaler P-Regler für 4 Kanäle

```
#Include ADwinPro_All.Inc
#Define setpoint Par_1 'Sollwert in Digits
#Define gain FPar_2 'Verstärkung
Dim i, deviation As Long
Dim input[4], output[4] As Long
```

Event:

```
P2_Read_ADCF4(1,input,1) '4 Eingangswerte lesen
For i = 1 To 4
    deviation = setpoint - input[i] 'Regelabweichung berechnen
    output[i] = deviation * gain 'Stellgröße berechnen
Next i
P2_DAC4(2,output,1) '4 Stellgrößen ausgeben
```

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

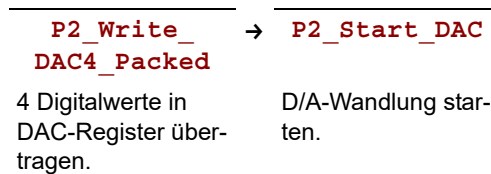
P2_DAC4_Packed(module,array[],index)
```

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]	
Bitnr.	31:16	15:0	31:16	15:0
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-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo Rev. E, MIO-4 Rev. E, MIO-4-ET1 Rev. E

Beispiel

Rem Digitaler P-Regler für 4 Kanäle

```
#Include ADwinPro_All.Inc
#Definesetpoint Par_1      'Sollwert in Digits
#Definegain FPar_2        'Verstärkung
Dim i, deviation1, deviation2 As Long
Dim input[2], output[2] As Long
```

Event:

```
P2_Read_ADCF4_Packed(1,input,1) '4 Eingangswerte lesen
For i = 1 To 2
  Rem Regelabweichung berechnen
  deviation1 = setpoint - (input[i] And 0FFFFh)
  deviation2 = setpoint - (Shift_Right(input[i],16) And 0FFFFh)
  Rem Stellgrößen berechnen und speichern
  output[i] = Shift_Left(deviation2*gain, 16) + deviation1*gain
Next i
P2_DAC4_Packed(2,output,1) '4 Stellgrößen ausgeben
```

P2_DAC4_Packed

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

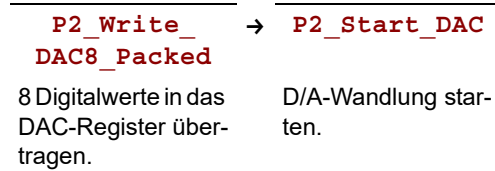
P2_DAC8(module,array[],index)
```

Parameter

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_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, AOut-8/16-TiCo Rev. E

Beispiel

Rem Digitaler P-Regler für 8 Kanäle

```
#Include ADwinPro_All.Inc
#Define setpoint Par_1 'Sollwert in Digits
#Define gain FPar_2 'Verstärkung
Dim i, deviation As Long
Dim input[8], output[8] As Long
```

Event:

```
P2_Read_ADCF8(1,input,1) '8 Eingangswerte lesen
For i = 1 To 8
  deviation = setpoint - input[i] 'Regelabweichung berechnen
  output[i] = deviation * gain 'Stellgröße berechnen
Next i
P2_DAC8(2,output,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

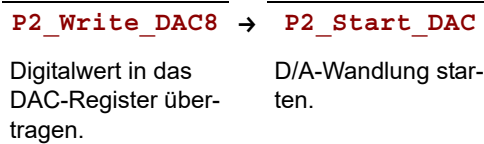
P2_DAC8_Packed(module,array[],index)
```

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]
Bitnr.	31:16	15:0	31:16	15:0
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](#), [AOut-8/16-TiCo Rev. E](#)

Beispiel

Rem Digitaler P-Regler für 4 Kanäle

```
#Include ADwinPro_All.Inc
#Definesetpoint Par_1      'Sollwert in Digits
#Definegain FPar_2         'Verstärkung
Dim i, deviation1, deviation2 As Long
Dim input[2], output[2] As Long
```

Event:

```
P2_Read_ADCF8_Packed(1,input,1) '8 Eingangswerte lesen
For i = 1 To 4
    Rem Regelabweichung berechnen
    deviation1 = setpoint - (input[i] And 0FFFFh)
    deviation2 = setpoint - (Shift_Right(input[i],16) And 0FFFFh)
    Rem Stellgrößen berechnen und speichern
    output[i] = Shift_Left(deviation2*gain, 16) + deviation1*gain
Next i
P2_DAC8_Packed(2,output,1) '8 Stellgrößen ausgeben
```

P2_Start_DAC

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

Sie können mit dem Befehl **P2_Sync_All** Wandlungen auf mehreren Modulen synchron starten. Dabei können Sie einzelne Kanäle mit **P2_Sync_Enable** für die Synchronisierung deaktivieren.

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](#), [P2_DAC_Ramp_Write](#), [P2_Sync_All](#), [P2_Sync_Enable](#)

Gültig für

[AOut-1/16 Rev. E](#), [AOut-4/16 Rev. E](#), [AOut-4/16-TiCo Rev. E](#), [AOut-8/16 Rev. E](#), [AOut-8/16-TiCo 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_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, 1...4 oder 1...8) des Ausgangs.	LONG
value	Auszugebender Wert (0...65535)	LONG

Bemerkungen

Bei mehreren DAC empfehlen wir, die Befehle **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](#), [P2_DAC_Ramp_Write](#)

Gültig für

AOut-1/16 Rev. E, AOut-4/16 Rev. E, AOut-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo 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_DAC

P2_Write_DAC4

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

<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

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-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo 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_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]	
Bitnr.	31:16	15:0	31:16	15:0
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-4/16-TiCo Rev. E, AOut-8/16 Rev. E, AOut-8/16-TiCo 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_DAC4_Packed

P2_Write_DAC8

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

<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

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, AOut-8/16-TiCo 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_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]
Bitnr.	31:16	15:0	31:16	15:0
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, AOut-8/16-TiCo 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_DAC8_Packed

P2_Write_DAC32

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	Auszugebender Wert (0h...0FFFFFFFh).	LONG

Siehe auch

Das untere Wort (Bits15:0) des Digitalwerts value32 wird in den DAC mit der ungeraden Nummer geschrieben, das obere Wort (Bits 31:16) 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-4/16-TiCo Rev. E](#), [AOut-8/16 Rev. E](#), [AOut-8/16-TiCo 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 5+6 setzen
P2_Start_DAC(1) 'Ausgabe auf allen DAC starten
Inc(i)
If (i>1000) Then i=1
```

P2_DAC1_DIO gibt auf dem DAC-Kanal 1 eine (analoge) Spannung aus und setzt oder löscht die digitalen Ausgänge des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.Inc
P2_DAC1_DIO(module, value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
value	Kombinierter Ausgabewert für DAC und digitale Ausgänge (zur Zuordnung der Ausgänge siehe Tabelle). Bits 15:0: DAC-Ausgabewert (0...65535). Bits 31:16: Bitmuster, nach dem die digitalen Ausgänge gesetzt werden. Bit = 0: Ausgang auf Pegel Low setzen. Bit = 1: Ausgang auf Pegel High setzen.	LONG

Bitnr.	31	...	16	15:0
Ausgang	31	...	16	–

Bemerkungen

Die Ausgabe auf dem DAC1 und auf den digitalen Ausgängen geschieht gleichzeitig.

Siehe auch

[P2_DAC](#), [P2_Digout_Long](#)

Gültig für

[AOut-1/16 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.Inc
#Define module 5
Dim value As Long
```

Event:

```
Rem DAC output 0V (32768) and set dig. outputs 16..19
P2_DAC1_DIO(module, Join_DAC_DIO(32768, 01111b))
```

```
Function Join_DAC_DIO(dac_val, dio_val) As Long
    Join_DAC_DIO = dac_val And Shift_Left(dio_val, 16)
EndFunction
```

P2_DAC1_DIO

P2_DAC_Ramp_Write

P2_DAC_Ramp_Write definiert die Parameter für die Ausgabe der nächsten Rampe und startet die DAC-Ausgabe.

Syntax

```
#Include ADwinPro_All.Inc

P2_DAC_Ramp_Write(module, dac_no, start_value,
                  end_value, dio_start, dio_end, time, out_mode)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dac_no	Nummer (1) des Ausgangs.	LONG
start_value	Startwert (0...65535) der DAC-Rampe.	LONG
end_value	Endwert (0...65535) der DAC-Rampe.	LONG
dio_start	Auszugebender Wert (10000h...0FFF0000h) für digitale Ausgänge am Beginn der Rampe.	LONG
dio_end	Auszugebender Wert (10000h...0FFF0000h) für digitale Ausgänge am Ende der Rampe.	LONG
time	Dauer (20, 40, 60 ... 65520) der Rampe in ns.	LONG
out_mode	Bitmuster zum Festlegen des Ausgabemodus: Bit 0: Ausgabe der DAC-Rampe. Bit 1: Ausgabe des Digitalwerts dio_start . Bit 2: Ausgabe des Digitalwerts dio_end .	LONG

Bit = 0: Ausgabe inaktiv.
Bit = 1: Ausgabe aktiv.

Bitnr. in	31	30	...	17	16	15:0
dio_start, dio_end						
Dig. Ausgang	31	30	...	17	16	–

Bemerkungen

Eine Rampe wird definiert durch den Startwert **start_value**, den Endwert **end_value** und die Zeitdauer **time**. Nach Ausgabe des Startwerts erhöht (oder erniedrigt) das Modul die Ausgabespannung alle 20ns um einen festen Betrag, bis der Endwert erreicht ist. Je nach Daten kann die tatsächliche Rampendauer vom Wert **time** geringfügig abweichen.

Wenn Start- und Endwert gleich sind, wird die Ausgabespannung am DAC für die Zeitdauer **time** konstant gehalten.

Durch mehrere Rampen hintereinander kann eine frei definierbare Spannungs-kurve ausgegeben werden. Während eine Rampe ausgegeben wird, kann die nächste Rampe mit **P2_DAC_Ramp_Write** in einen Zwischenpuffer geschrieben werden. Es kann jeweils nur eine Rampe zwischengepuffert werden.

Fragen Sie erst mit **P2_DAC_Ramp_Buffer_Free** ab, ob Platz im Zwischenpuffer frei ist, und schreiben Sie erst dann die nächste Rampe mit **P2_DAC_Ramp_Write** in den Zwischenpuffer.

Mit **P2_DAC_Ramp_Status** fragen Sie ab, ob aktuell eine Rampe ausgegeben wird.

Sie können die Rampen-Ausgabe mit **P2_DAC_Ramp_Stop** sofort abbrechen.

Alternativ zur Ausgabe als Rampe können Sie Spannungswerte auch einzeln oder über den Ausgangs-Fifo ausgeben. Eine direkte Verbindung der verschiedenen Ausgabemethoden (z. B. Rampe gefolgt von Ausgabe-Fifo) wird nicht unterstützt.

Siehe auch

[P2_DAC](#), [P2_DAC_Ramp_Status](#), [P2_DAC_Ramp_Buffer_Free](#), [P2_DAC_Ramp_Stop](#), [P2_DAC1_DIO](#), [P2_Dig_Fifo_Mode](#)

Gültig für

AOut-1/16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define module 5
#Define dac_no 1
Dim array[5] As Long
Dim ramp_no, v_start, v_end As Long
```

Init:

```
Rem stop possibly running ramp
P2_DAC_Ramp_Stop(module, dac_no)
Rem define values for 4 ramps
array[1] = 0           '-2.0 V
array[2] = 16384        '-1.0 V
array[3] = 57344        '+1.5 V
array[4] = 32768        ' 0.0 V
array[5] = 49152        '+1.0 V
ramp_no = 1
```

Event:

```
Rem check if buffer has free space
If (P2_DAC_Ramp_Buffer_Free(module,dac_no) >= 0) Then
    v_start = array[ramp_no]
    v_end = array[ramp_no+1]
    Rem write ramp, set ramp time 1.5µs, no digital values
    P2_DAC_Ramp_Write(module,dac_no,v_start,v_end,0,0,1500,001b)
    Inc ramp_no
    If (ramp_no > 4) Then ramp_no = 1
EndIf
```

P2_DAC_Ramp_ Status

P2_DAC_Ramp_Status gibt zurück, ob eine Rampenausgabe aktiv ist.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_DAC_Ramp_Status(module, dac_no)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>dac_no</code>	Nummer (1) des Ausgangs.	LONG
<code>ret_val</code>	Status der Rampenausgabe: 0: keine Rampenausgabe aktiv. <>0: Rampenausgabe ist aktiv.	LONG

Bemerkungen

Sie können die Rampen-Ausgabe mit **P2_DAC_Ramp_Stop** sofort abbrechen.

Alternativ zur Ausgabe als Rampe können Sie Spannungswerte auch einzeln oder über den Ausgangs-Fifo ausgeben. Prüfen Sie erst mit **P2_DAC_Ramp_Status**, ob die Rampenausgabe beendet ist, bevor Sie eine andere Ausgabemethode verwenden.

Siehe auch

[P2_DAC](#), [P2_DAC_Ramp_Write](#), [P2_DAC_Ramp_Buffer_Free](#), [P2_DAC_Ramp_Stop](#), [P2_DAC1_DIO](#), [P2_Dig_Fifo_Mode](#)

Gültig für

AOut-1/16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define module 5
#Define dac_no 1
Dim value[4] As Long
Dim ramp_active As Long

Init:
    Processdelay = 6000      '6000 x 3.3 ns = 20µs

    Rem stop possibly running ramp
    P2_DAC_Ramp_Stop(module, dac_no)
    Rem write ramp, set ramp -2V..0V, time 10.5µs, no digital values
    P2_DAC_Ramp_Write(module, dac_no, 0, 32768, 0, 0, 10500, 001b)
    ramp_active = 1

    Rem Do settings for output fifo (start output later)
    value[1] = 0C001C000h      'output 1V, dig. output 16
    value[2] = 500              ' with output time 5 µs (relative)
    value[3] = 0C002FFFFh      'output 2V, dig. output 17
    value[4] = 700              ' with output time 7 µs (relative)
    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
    P2_Digout_Fifo_Write(module, 2, value, 1)

Event:
    If (ramp_active <> 0) Then
        Rem As Long as ramp was running: check status
        ramp_active = P2_DAC_Ramp_Status(module, dac_no)
    Else
        Rem ramp has finished -> start output FIFO
        P2_Digout_Fifo_Start(Shift_Left(1, module-1))

        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
    EndIf
```

P2_DAC_Ramp_Buffer_Free

P2_DAC_Ramp_Buffer_Free gibt an, ob der Zwischenpuffer für die Rampenausgabe frei ist.

Syntax

```
#Include ADwinPro_All.Inc
```

```
ret_val = P2_DAC_Ramp_Buffer_Free(module, dac_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dac_no	Nummer (1) des Ausgangs.	LONG
ret_val	Status des Zwischenpuffers zur Rampenausgabe: >=0: Zwischenpuffer ist frei. <0: Zwischenpuffer ist belegt.	LONG

Bemerkungen

Durch mehrere Rampen hintereinander kann eine frei definierbare Spannungskurve erzeugt werden. Während eine Rampe ausgegeben wird, kann die nächste Rampe mit **P2_DAC_Ramp_Write** in einen Zwischenpuffer geschrieben werden. Es kann jeweils nur eine Rampe zwischengepuffert werden.

Fragen Sie erst mit **P2_DAC_Ramp_Buffer_Free** ab, ob Platz im Zwischenpuffer frei ist, und schreiben Sie erst dann die nächste Rampe mit **P2_DAC_Ramp_Write** in den Zwischenpuffer.

Mit **P2_DAC_Ramp_Status** fragen Sie ab, ob aktuell eine Rampe ausgegeben wird.

Siehe auch

[P2_DAC](#), [P2_DAC_Ramp_Write](#), [P2_DAC_Ramp_Status](#), [P2_DAC_Ramp_Stop](#), [P2_DAC1_DIO](#), [P2_Dig_Fifo_Mode](#)

Gültig für

AOut-1/16 Rev. E

Beispiel

siehe [P2_DAC_Ramp_Write](#)

P2_DAC_Ramp_Stop stoppt eine Rampenausgabe sofort.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_DAC_Ramp_Stop(module, dac_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dac_no	Nummer (1) des Ausgangs.	LONG

Bemerkungen

P2_DAC_Ramp_Stop stoppt nicht nur die Rampenausgabe, sondern löscht auch den Zwischenpuffer für die Rampenausgabe.

Wir empfehlen, die Rampenausgabe mit **P2_DAC_Ramp_Stop** im Abschnitt **Init:** zu initialisieren.

Siehe auch

[P2_DAC](#), [P2_DAC_Ramp_Write](#), [P2_DAC_Ramp_Status](#), [P2_DAC_Ramp_Buffer_Free](#), [P2_DAC1_DIO](#), [P2_Dig_Fifo_Mode](#)

Gültig für

[AOut-1/16 Rev. E](#)

Beispiel

siehe [P2_DAC_Ramp_Write](#)

P2_DAC_Ramp_Stop

3.6 Pro II: Digitale Ein-/Ausgänge

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

- [P2_Comp_Init \(Seite 145\)](#)
- [P2_Comp_Filter_Init \(Seite 147\)](#)
- [P2_Comp_Set \(Seite 148\)](#)
- [P2_Comp_Set_Voltage \(Seite 149\)](#)
- [P2_Dig_Fifo_Mode \(Seite 150\)](#)
- [P2_Dig_Latch \(Seite 152\)](#)
- [P2_Dig_Read_Latch \(Seite 153\)](#)
- [P2_Dig_Write_Latch \(Seite 154\)](#)
- [P2_Digin_Edge \(Seite 155\)](#)
- [P2_Digin_Fifo_Clear \(Seite 157\)](#)
- [P2_Digin_Fifo_Enable \(Seite 158\)](#)
- [P2_Digin_Fifo_Full \(Seite 159\)](#)
- [P2_Digin_Fifo_Read \(Seite 160\)](#)
- [P2_Digin_Fifo_Read_Fast \(Seite 162\)](#)
- [P2_Digin_Fifo_Read_Timer \(Seite 164\)](#)
- [P2_Digin_Filter_Init \(Seite 165\)](#)
- [P2_Digin_Long \(Seite 167\)](#)
- [P2_Digout \(Seite 168\)](#)
- [P2_Digout_Bits \(Seite 169\)](#)
- [P2_Digout_Fifo_Clear \(Seite 171\)](#)
- [P2_Digout_Fifo_Empty \(Seite 172\)](#)
- [P2_Digout_Fifo_Enable \(Seite 173\)](#)
- [P2_Digout_Fifo_Read_Timer \(Seite 174\)](#)
- [P2_Digout_Fifo_Start \(Seite 175\)](#)
- [P2_Digout_Fifo_Write \(Seite 176\)](#)
- [P2_Digout_Long \(Seite 179\)](#)
- [P2_Digout_Reset \(Seite 180\)](#)
- [P2_Digout_Set \(Seite 181\)](#)
- [P2_DigProg \(Seite 182\)](#)
- [P2_DigProg_Set_IO_Level \(Seite 184\)](#)
- [P2_Get_Digout_Long \(Seite 185\)](#)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

P2_Comp_Init stellt den Betriebsmodus für die Komparatoren einer Kanalgruppe auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.inc

P2_Comp_Init(module, ch_group, mode)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ch_group	Kanalgruppe (1...4): 1: Kanäle 0, 1, 2, 3 2: Kanäle 4, 5, 6, 7 3: Kanäle 8, 9, 10, 11 4: Kanäle 12, 13, 14, 15	LONG
mode	Betriebsmodus (0...7) der Komparatoren: 0: Weder Hysterese noch Halten. 1: Halten 1µs. 2: Halten 10µs. 3: Halten 100µs. 4: Hysterese ±34mV um die Schaltschwelle (bis E02: ±17mV) 5: Hysterese ±66mV um die Schaltschwelle (bis E02: ±33mV) 6: Hysterese ±110mV um die Schaltschwelle (bis E02: ±55mV) 7: Hysterese ±200mV um die Schaltschwelle (bis E02: ±100mV)	LONG

Bemerkungen

Der Betriebsmodus gilt für die Kanäle einer Kanalgruppe. Die Einstellung ist auch dann wirksam, wenn eine Kanalgruppe für die freie Hysterese genutzt wird.

Sie lesen die Komparator-Bits mit Befehlen für Digitaleingänge wie z.B. **P2_Digin_Long**.

Sobald der anliegende Analogwert die Schaltschwelle erreicht, wird der Komparator für eine definierte Haltezeit deaktiviert und arbeitet erst danach weiter.

Die Hysterese bezieht sich auf die mit **P2_Comp_Set** eingestellte Schaltschwelle. Das Bit des Kanals wird auf 1 gesetzt, wenn die Schaltschwelle plus Hysterese überschritten wird, und wieder auf 0 gesetzt, wenn die Schaltschwelle minus Hysterese unterschritten wird.

Siehe auch

[P2_Comp_Filter_Init](#), [P2_Comp_Set](#), [P2_Comp_Set_Voltage](#), [P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Digin_Edge](#), [P2_Digin_Fifo_Enable](#), [P2_Digin_Long](#)

Gültig für

Comp-16 Rev. E

P2_Comp_Init

Komparator halten

Hysterese

Beispiel

```
#Include ADwinPro_All.inc
#Define module 2
Dim Bit_Ch_0, Bit_Ch_0_1 As Long

Init:
  P2_Comp_Init(module,1,3) 'set channels 0..3 to hold 200µs
  Rem Set thresholds: channel group 1 to 0V, channel group 2 to 10V
  P2_Comp_Set(module,1,Volt2Digits(0.0))
  P2_Comp_Set(module,2,Volt2Digits(10.0))
  Rem Set thresholds in volts directly
  'P2_Comp_Set_Voltage(module, 1, 0.0)
  'P2_Comp_Set_Voltage(module, 1, 10.0)
  Rem Set filter to length 200 ns
  P2_Comp_Filter_Init(module, 10)

Event:
  Rem check comparator bits
  Par_1 = P2_Digin_Long(module)
  Rem channel 1
  Bit_Ch_0 = Par_1 And 1b 'threshold of channel 0
  Rem free hysteresis determined by channels 0, 1
  Bit_Ch_0_1 = Shift_Right(Par_1,16) And 1

  If (Bit_Ch_0 = 1) Then
    Rem channel 0 > 0 V
  Else ' (Bit_Ch_0 = 0)
    Rem channel 0 < 0 V
  EndIf

  If (Bit_Ch_0_1 = 1) Then
    Rem both channels 0, 1 > 10 V
  Else ' (Bit_Ch_0_1 = 0)
    Rem both channels 0, 1 < 0 V
  EndIf

Function Volt2Digits(volt) As Long
  Volt2Digits = volt * (65536 / 60) + 32768
EndFunction
```

P2_Comp_Filter_Init stellt die Filter-Prüfdauer für alle Komparatoren auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.inc

P2_Comp_Filter_Init(module, filter_value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
filter_value	Prüfdauer des Filters, angegeben in Takten (1...65535) mit der Länge 20ns.	LONG
Der Wert 0 (Null) deaktiviert den Filter.		

Bemerkungen

Der Filter unterdrückt einzelne Fehlpulse (Spikes) eines Signals. Die Anzahl der Fehlpulse sollte im Verhältnis zur Pulsbreite des Signals klein sein. Die Prüfdauer des Filters sollte etwas länger sein als die erwartete Breite der Fehlpulse.

Die Filtereinstellungen gelten für alle Kanäle gleichermaßen. Jeder Kanal hat seinen eigenen Filter. Nach dem Einschalten sind die Filter deaktiviert.

Beachten Sie: Der Filter verzögert Flanken des resultierenden Signals um die eingestellte Prüfdauer. Falls Spikes auftreten, verzögern sich die Flanken zusätzlich geringfügig.

Der Filter ist nicht geeignet für die Kombination mit dem Komparator-Modus „Hold“.

Siehe auch

[P2_Comp_Init](#), [P2_Comp_Set](#), [P2_Comp_Set_Voltage](#), [P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Digin_Edge](#), [P2_Digin_Fifo_Enable](#), [P2_Digin_Long](#)

Gültig für

[Comp-16 Rev. E](#)

Beispiel

siehe [P2_Comp_Init](#)

P2_Comp_Filter_Init

P2_Comp_Set

P2_Comp_Set stellt die Schaltschwelle für die Komparatoren einer Kanalgruppe auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_Comp_Set(module, ch_group, value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ch_group	Kanalgruppe (1...4): 1: Kanäle 0, 2, 4, 6 2: Kanäle 1, 3, 5, 7 3: Kanäle 8, 10, 12, 14 4: Kanäle 9, 11, 13, 15	LONG
value	Auszugebender Wert (31676...65535) in Digits für die Schaltschwelle von -1V...30V.	LONG

Bemerkungen

Mit dem Befehl **P2_Comp_Set_Voltage** geben Sie die Schaltschwelle in Volt an.

Die Umrechnung zwischen Digits und Spannung ist wie folgt:

$$\text{Digits} = 32768 + \frac{\text{Spannung} \cdot 65536}{60 \text{ V}}$$

$$\text{Spannung} = \frac{60 \text{ V}}{65536} \cdot (\text{Digits} - 32768)$$

Sie lesen die Komparator-Bits mit Befehlen für Digitaleingänge wie z.B. **P2_Digin_Long**.

Bei Halten / Standard-Hysteresis ist die Zuordnung der Bits zu den Einzelkanälen wie folgt:

Bitnr.	31:16	15	14	...	2	1	0
Kanalnr.	–	15	14	...	2	1	0

Die Komparator-Bits für die Kanalpaare der frei wählbaren Hysteresis sind die Bits 23:16; die Zuordnung ist wie folgt:

Bitnr.	31:24	23	22	...	17	16	15:0
Kanalpaar	–	14, 15	12, 13	...	2, 3	0, 1	–

Siehe auch

[P2_Comp_Init](#), [P2_Comp_Filter_Init](#), [P2_Comp_Set_Voltage](#), [P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Digin_Edge](#), [P2_Digin_Fifo_Enable](#), [P2_Digin_Long](#)

Gültig für

Comp-16 Rev. E

Beispiel

siehe [P2_Comp_Init](#)

P2_Comp_Set_Voltage stellt die Schaltschwelle für die Komparatoren einer Kanalgruppe auf dem angegebenen Modul ein, angegeben in Volt.

Syntax

```
#Include ADwinPro_All.inc

P2_Comp_Set_Voltage(module, ch_group, volts)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ch_group	Kanalgruppe (1...4): 1: Kanäle 0, 2, 4, 6 2: Kanäle 1, 3, 5, 7 3: Kanäle 8, 10, 12, 14 4: Kanäle 9, 11, 13, 15	LONG
volts	Auszugebender Wert für die Schaltschwelle in Volt (-1V...30V).	FLOAT

Bemerkungen

Mit dem Befehl **P2_Comp_Set** geben Sie die Schaltschwelle in Volt an.

Die Umrechnung zwischen Digits und Spannung ist wie folgt:

$$\text{Digits} = 32768 + \frac{\text{Spannung} \cdot 65536}{60 \text{ V}}$$

$$\text{Spannung} = \frac{60 \text{ V}}{65536} \cdot (\text{Digits} - 32768)$$

Sie lesen die Komparator-Bits mit Befehlen für Digitaleingänge wie z.B. **P2_Digin_Long**.

Bei Halten / Standard-Hysteresis ist die Zuordnung der Bits zu den Einzelkanälen wie folgt:

Bitnr.	31:16	15	14	...	2	1	0
Kanalnr.	–	15	14	...	2	1	0

Die Komparator-Bits für die Kanalpaare der frei wählbaren Hysteresis sind die Bits 23:16; die Zuordnung ist wie folgt:

Bitnr.	31:24	23	22	...	17	16	15:0
Kanalpaar	–	14, 15	12, 13	...	2, 3	0, 1	–

Siehe auch

[P2_Comp_Init](#), [P2_Comp_Filter_Init](#), [P2_Comp_Set](#), [P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Digin_Edge](#), [P2_Digin_Fifo_Enable](#), [P2_Digin_Long](#)

Gültig für

Comp-16 Rev. E

Beispiel

siehe [P2_Comp_Init](#)

P2_Comp_Set_Voltage

P2_Dig_Fifo_Mode

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, einfache Ausgabe. 3: Ausgangs-FIFO zur Flankenausgabe, Zeitwerte relativ, einfache Ausgabe. 5: nur für TiCo2 Ausgangs-FIFO zur Flankenausgabe, Zeitwerte absolut, kontinuierliche Ausgabe. 7: nur für TiCo2 Ausgangs-FIFO zur Flankenausgabe, Zeitwerte relativ, kontinuierliche Ausgabe.	LONG

Bemerkungen

Der Ausgangs-FIFO steht auf dem Modul DIO-32-TiCo seit Revision 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.

Mit dem Prozessor TiCo2 können die Werte im Ausgangs-FIFO entweder einfach oder kontinuierlich ausgegeben werden:

- einfache Ausgabe: Die im FIFO enthaltene Liste an Wertepaaren wird einmal ausgegeben, danach stoppt die Ausgabe.
- kontinuierliche Ausgabe: Die im FIFO enthaltene Liste an Wertepaaren wird so lange ausgegeben und wiederholt, bis die Ausgabe mit **P2_Digout_Fifo_Clear** gestoppt wird.

Bei einfacher Ausgabe (Modi 1 und 3) kann – solange Wertepaare im FIFO vorhanden sind – die Liste an Wertepaaren im FIFO aufgefüllt werden.

Beim Modul AOut-1/16 können nur die Modi 1 und 3 eingestellt werden.

Siehe auch

[P2_Digin_Fifo_Enable](#), [P2_Digout_Fifo_Read_Timer](#), [P2_Digout_Fifo_Start](#), [P2_Digout_Fifo_Clear](#), [P2_Digout_Fifo_Write](#), [P2_DigProg_Set_IO_Level](#), [P2_DAC_Ramp_Write](#)

Gültig für

[AOut-1/16 Rev. E](#), [DIO-32-TiCo Rev. E](#), [DIO-32-TiCo2 Rev. E](#), [DIO-32/1-TiCo Rev. E](#), [DIO-8-D12 Rev. E](#), [MIO-D12 Rev. E](#), [SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

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)
    Rem with AOUT-1/16: delete line with P2_DigProg
    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))

    Rem for DIO-32-TiCo2 only: set voltage level
    Rem P2_DigProg_Set_IO_Level(module, 0, 160)

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

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. Lesen Sie die Werte mit **P2_Dig_Read_Latch**.

Bei digitalen Ausgängen überträgt die Anweisung die Werte der Ausgangs-Latches an die Ausgänge. Schreiben Sie Werte mit **P2_Dig_Write_Latch** in die Ausgangs-Latches.

Das Latchen kann synchron mit Aktionen auf anderen Modulen gestartet werden. Verwenden Sie hierzu den Befehl **P2_Sync_All**.

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

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, REL-16 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Bei AOUT-1/16 und SENT: Zeile mit P2_DigProg löschen  
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 latches, 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_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

Modul AOut-1/16:

Bitnr.	31:16	15	...	0
Eingang	–	15	...	0

Modul SENT-4-Out, SENT-6:

Bitnr.	31:20	19	...	16	15:0
Eingang	–	19	...	16	–

Andere Module:

Bitnr.	31	30	29	...	2	1	0
Eingang	31	30	29	...	2	1	0

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **P2_DigProg** als Eingänge zu programmieren. Das gilt nicht für das Modul AOut-1/16.

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

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

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

Init:

```
Rem Bei AOUT-1/16 und SENT: Zeilen mit P2_DigProg löschen
Rem DIO31:00 der Module 1+2 als Eingänge setzen
```

```
P2_DigProg(1,0000b)
P2_DigProg(2,0000b)
```

Event:

```
REM Pegel an den digitalen Eingängen von beiden Modulen synchron
REM in die Zwischenregister übernehmen
```

```
P2_Sync_All(11b)
Par_1 = P2_Dig_Read_Latch(1) 'Zwischenregister Modul 1 auslesen
Par_2 = P2_Dig_Read_Latch(2) 'Zwischenregister Modul 2 auslesen
```

P2_Dig_Read_Latch

P2_Dig_Write_Latch

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

Modul AOut-1/16:

Bitnr.	31	...	16	15:0
Ausgang	31	...	16	–

Modul SENT-4-Out, SENT-6:

Bitnr.	31:28	27	...	24	23:0
Ausgang	–	19	...	16	–

Andere Module:

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. Das gilt nicht für das Modul AOut-1/16.

Sie können digitale Ausgänge mit der Anweisung **P2_Digout** direkt 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](#), [P2_Digout_Bits](#)

Gültig für

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, REL-16 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Bei AOUT-1/16 und SENT: Zeile mit P2_DigProg löschen
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_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

Modul AOut-1/16:

Bitnr.	31:16	15	...	0
Eingang	–	15	...	0

Modul SENT-4-Out, SENT-6:

Bitnr.	31:20	19	...	16	15:0
Eingang	–	19	...	16	–

Andere Module:

Bitnr.	31	30	...	2	1	0
Eingang	31	30	...	2	1	0

Bemerkungen

Der Aufruf von **P2_Digin_Edge** setzt alle Bits zurück auf 0.

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.

Bei Pro II-MIO-D12 Rev. E können nur die Bits 11:0 (entspricht OPT11:OPT0) sinnvoll genutzt werden.

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

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

P2_Digin_Edge

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
P2_DigProg(1,1100b)      'Kanäle 15:0 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_Clear_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

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

siehe [P2_Digin_Fifo_Read](#)

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. Bit = 0: Überwachung ausgeschaltet. Bit = 1: Überwachung eingeschaltet.	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; davon ausgenommen sind OPT-Kanäle und das Modul AOut-1/16.

Bei Pro II-MIO-D12 Rev. E können nur die Bits 11:0 (entspricht OPT11:OPT0) genutzt werden.

Der FIFO sollte mit **P2_Dig_Fifo_Mode** für die Flankenüberwachung aktiviert werden.

Die Flankenüberwachung prüft bei jedem Zählertakt (alle 10ns / 5ns), 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 Eingänge als Bitmuster. Bei Ausgängen ist der Status undefiniert.
- Wert 2 enthält einen Zeitstempel, den aktuellen Stand eines Zählers.
DIO-32-TiCo2, DIO-8-D12: Taktrate 200MHz
alle anderen Module: Taktrate 100MHz

Das FIFO-Feld kann maximal 511 bzw. 2047 Wertepaare (Pegelzustand und Zeitstempel) enthalten, je nach Modultyp. 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

[AOut-1/16 Rev. E](#), [Comp-16 Rev. E](#), [DIO-32 Rev. E](#), [DIO-32-TiCo Rev. E](#), [DIO-32-TiCo2 Rev. E](#), [DIO-32/1-TiCo Rev. E](#), [DIO-8-D12 Rev. E](#), [MIO-D12 Rev. E](#), [OPT-16 Rev. E](#), [OPT-32-24V Rev. E](#), [SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

siehe [P2_Digin_Fifo_Read](#)

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 der belegten Wertepaare im FIFO: DIO-32-TiCo2, DIO-8-D12: 0...2047 alle anderen Module: 0...511	LONG

Bemerkungen

Das FIFO-Feld kann maximal 511 bzw. 2047 Wertepaare (Pegelzustand und Zeitstempel) enthalten, je nach Modultyp. 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

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

siehe [P2_Digin_Fifo_Read](#)

P2_Digin_Fifo_Full

P2_Digin_Fifo_Read

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 der zu lesenden Wertepaare: DIO-32-TiCo2, DIO-8-D12: 0...2047 alle anderen Module: 0...511	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

Bei Pro II-MIO-D12 Rev. E können nur die Bits 11:0 (entspricht OPT11:OPT0) genutzt werden.

Es 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.

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 Zählertakten. Die Dauer eines Zählertakts hängt vom Modultyp ab.

$$\Delta t = \text{Taktdauer} \cdot (\text{stamp}_1 - \text{stamp}_2)$$

Modul	Taktrate	Taktdauer
DIO-32-TiCo2, DIO-8-D12	200MHz	5ns
alle anderen Module	100MHz	10ns

Siehe auch

[P2_Digin_Fifo_Clear](#), [P2_Digin_Fifo_Enable](#), [P2_Digin_Fifo_Full](#), [P2_Digin_Fifo_Read_Timer](#), [P2_Digin_Edge](#)

Gültig für

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

```
Dim Data_1[10000], Data_2[10000] As Long
```

```
Dim num, index As Long
```

Init:

```
Rem Bei AOUT-1/16: Zeile mit P2_DigProg löschen
```

```
P2_DigProg(1,1100b) 'Kanäle 15:0 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

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 der zu lesenden Wertepaare: DIO-32-TiCo2, DIO-8-D12: 0...2047 alle anderen Module: 0...511	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

Bei Pro II-MIO-D12 Rev. E können nur die Bits 11:0 (entspricht OPT11:OPT0) genutzt werden.

Es 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 **valuepairs[]** werden die Wertepaare aus Pegelzustand und zugehörigem Zeitstempel abgelegt:

- Ein Feldelement enthält den Pegelzustand der Eingänge 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 Zählertakten. Die Taktdauer ist abhängig vom Modultyp.

$$\Delta t = \text{Taktdauer} \cdot (\text{stamp}_1 - \text{stamp}_2)$$

Modul	Taktrate	Taktdauer
DIO-32-TiCo2, DIO-8-D12	200MHz	5ns
alle anderen Module	100MHz	10ns

Siehe auch

[P2_Digin_Fifo_Clear](#), [P2_Digin_Fifo_Enable](#), [P2_Digin_Fifo_Full](#), [P2_Digin_Fifo_Read_Timer](#), [P2_Digin_Edge](#)

Gültig für

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

```
Dim Data_1[20000] As Long
```

```
Dim num, index As Long
```

Init:

```
Rem Bei AOUT-1/16: Zeile mit P2_DigProg löschen
```

```
P2_DigProg(1,1100b) 'Kanäle 15:0 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 + 2 * num
```

```
If (index > 10000) Then index = 1
```

```
EndIf
```

P2_Digin_Fifo_Read_Timer

P2_Digin_Fifo_Read_Timer gibt den aktuellen Stand des 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 ($-2^{31}-1$... 2^{31}) des Zählers.	LONG

Bemerkungen

Der Modulzähler wird für das Erzeugen der Zeitstempel bei der Flankenüberwachung benutzt, siehe **P2_Digin_Fifo_Enable**.

Der Zähler wird regelmäßig um 1 erhöht, so dass der Zähler nach 2^{32} Takten seinen ursprünglichen Wert erneut erreicht. Bei Zeitvergleichen muss dieser „Überlauf“ berücksichtigt werden, der Zählerstand muss daher im Programm regelmäßig vor dem Überlauf abgefragt werden. Der Zähler arbeitet je nach Modultyp mit unterschiedlicher Taktrate:

Modul	Taktrate	Taktdauer	Überlaufzeit
DIO-32-TiCo2, DIO-8-D12	200MHz	5ns	$21s = 5ns \times 2^{32}$
alle anderen Module	100MHz	10ns	$43s = 10ns \times 2^{32}$

Siehe auch

[P2_Digin_Fifo_Enable](#), [P2_Digin_Fifo_Read](#)

Gültig für

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define count_overflow Par_1
Dim t_start, diff_new, diff_old As Long

Init:
count_overflow = 0          'overflow occurs every 43 seconds
t_start = P2_Digin_Fifo_Read_Timer()
diff_old = 0

Event:
Rem Event section must be run at least once every 20 seconds.
Rem Else you will miss counter overflows.

Rem get timer difference
diff_new = P2_Digin_Fifo_Read_Timer() - t_start
If ((diff_new > 0) And (diff_old < 0)) Then
    Inc(count_overflow)      'increase number of counter overflows
EndIf
diff_old = diff_new
```

ähnliche Beispiele siehe

- **ADbasic**-Beispiel im Ordner C:\ADwin\ADbasic\samples_ADwin:
[seconds_timer.bas](#)
- **TiCoBasic**-Beispiel [seconds_timer_TiCo.bas](#) im Ordner
C:\ADwin\TiCoBasic\samples_ADwin

P2_Digin_Filter_Init stellt die Filter-Prüfdauer für alle Eingänge auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.inc

P2_Digin_Filter_Init(module, filter_value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
filter_value	Prüfdauer des Filters, angegeben in Einheiten (1...65535) von 20ns, bei DIO-32-TiCo2 und DIO-8-D12 in Einheiten von 5ns.	LONG

Der Wert 0 (Null) deaktiviert den Filter.

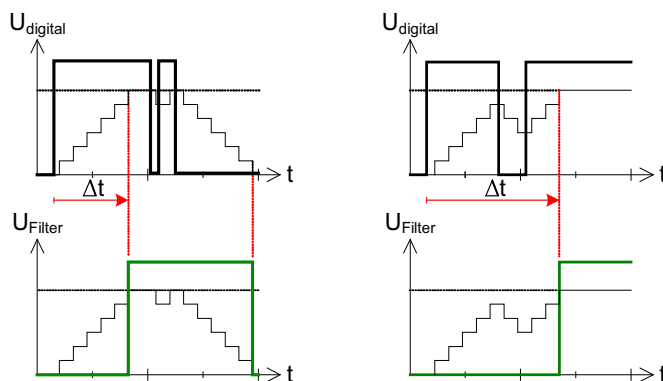
Bemerkungen

Der Filter unterdrückt einzelne Fehlpulse (Spikes) eines Signals. Die Anzahl der Fehlpulse sollte im Verhältnis zur Pulsbreite des Signals klein sein. Die Prüfdauer des Filters sollte etwas länger sein als die erwartete Breite der Fehlpulse.

Die Filtereinstellungen gelten für alle Kanäle gleichermaßen. Jeder Kanal hat seinen eigenen Filter. Nach dem Einschalten sind die Filter deaktiviert.

Der Filter überträgt einen Flankenwechsel des Eingangssignals nicht direkt auf das Ausgangssignal. Je nach Eingangssignal wird alle 20ns ein Zähler erhöht (High-Signal) oder erniedrigt (Low-Signal), in den Grenzen von 0... **filter_value**. Hat der Zähler den Wert 0, hat das Ausgangssignal den Pegel Low, bei **filter_value** ist es Pegel High.

Beachten Sie: Der Filter verzögert Flanken des resultierenden Signals um die eingestellte Prüfdauer. Falls Fehlpulse auftreten, könne sich die Flanken zusätzlich geringfügig verzögern.



Die Abbildung zeigt das Filtern von 2 Beispielsignalen (schwarzes Signal, oben) mit Fehlpulsen. Das Treppennmuster zeigt den Wert des Filterzählers. Im rechten Beispiel verzögert sich die resultierende Flanke durch den Fehlpuls. Der Filter (hier mit **filter_value=6**) verzögert die Flanken des resultierenden Signals; die Zeitverzögerung Δt kann größer werden, wenn mehrere Fehlpulse vorkommen.

Siehe auch

[P2_Comp_Init](#), [P2_Comp_Filter_Init](#), [P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Digin_Edge](#), [P2_Digin_Fifo_Enable](#), [P2_Digin_Long](#)

Gültig für

DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E

P2_Digin_Filter_Init

Beispiel

Rem example for DIO-32-TiCo

```
#Include ADwinPro_All.Inc
```

```
#Define module 2
```

Init:

```
P2_DigProg(module,0)      'Set DIO31:00 as inputs
```

```
P2_Digin_Filter_Init(module, 5) 'set spike filter to 100ns
```

Event:

```
Par_1 = P2_Digin_Long(module) 'Read all inputs
```

P2_Digin_Long gibt den Zustand der Eingänge (Bits 31:0) 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

Modul AOut-1/16:

Bitnr.	31:16	15	...	0
Eingang	–	15	...	0

Modul SENT-4-Out, SENT-6:

Bitnr.	31:20	19	...	16	15:0
Eingang	–	19	...	16	–

Andere Module:

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. Das gilt nicht für das Modul AOut-1/16.

Siehe auch

[P2_Comp_Init](#), [P2_Comp_Set](#), [P2_Dig_Latch](#), [P2_DigProg](#), [P2_Digout_Long](#)

Gültig für

AOut-1/16 Rev. E, Comp-16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, OPT-16 Rev. E, OPT-32-24V Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Bei AOUT-1/16 und SENT: Zeile mit P2_DigProg löschen
P2_DigProg(1,0000b) 'DIO 31:00 als Eingang
```

Event:

```
Par_1 = P2_Digin_Long(1) 'Alle Eingänge einlesen
```

P2_Digin_Long

P2_Digout

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, bei Modul AOut-1/16: 16...31, bei SENT-x: 16...19).	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. Das gilt nicht für die Module AOut-1/16, SENT-4-Out und SENT-6.

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

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, REL-16 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Bei AOUT-1/16: Zeile mit P2_DigProg löschen
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 = 8000h) 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_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

Modul AOut-1/16:

Bitnr.	31	...	16	15:0
Ausgang	31	...	16	–

Modul SENT-4-Out, SENT-6:

Bitnr.	31:28	27	...	24	23:0
Ausgang	–	19	...	16	–

Andere Module:

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. Das gilt nicht für das Modul AOut-1/16.

Mit **P2_Digout_Bits** 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

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, REL-16 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

P2_Digout_Bits

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Bei AOUT-1/16: Zeile mit P2_DigProg löschen
```

```
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,0AAAAh)
```

```
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

Auf dem Modul DIO-32-TiCo steht der Ausgangs-FIFO seit Revision E03 zur Verfügung.

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

[AOut-1/16 Rev. E](#), [DIO-32-TiCo Rev. E](#), [DIO-32-TiCo2 Rev. E](#), [DIO-32/1-TiCo Rev. E](#), [DIO-8-D12 Rev. E](#), [MIO-D12 Rev. E](#), [SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

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 Flanken-
ausgabe zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_value = P2_Digout_Fifo_Empty(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
ret_value	Anzahl der freien Wertepaare im FIFO: DIO-32-TiCo2, DIO-8-D12: 0...2047 alle anderen Module: 0...511	LONG

Bemerkungen

Das FIFO-Feld kann maximal 511 bzw. 2047 Wertepaare (Pegelzustand und
Zeitstempel) enthalten, je nach Modultyp.

Siehe auch

[P2_Dig_Fifo_Mode](#), [P2_Digout_Fifo_Read_Timer](#), [P2_Digout_Fifo_Start](#), [P2_Digout_Fifo_Write](#)

Gültig für

[AOut-1/16 Rev. E](#), [DIO-32-TiCo Rev. E](#), [DIO-32-TiCo2 Rev. E](#), [DIO-32/1-TiCo Rev. E](#), [DIO-8-D12 Rev. E](#), [MIO-D12 Rev. E](#), [SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

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

Modul AOut-1/16:

Bitnr.	31	30	29	...	16	15:0
Ausgang	–	–	29	...	16	–

Bemerkungen

Auf dem Modul DIO-32-TiCo steht der Ausgangs-FIFO seit Revision E03 zur Verfügung.

Flanken können nur auf Ausgangskanäle ausgegeben werden. Sie programmieren die Kanäle mit **P2_DigProg** als Eingänge oder Ausgänge; davon ausgenommen sind TRA-Kanäle und das Modul AOut-1/16.

Der FIFO muss mit **P2_Dig_Fifo_Mode** als Ausgangs-FIFO eingestellt werden.

P2_Digout_Fifo_Enable wählt Kanäle zur Flankenausgabe über den Ausgangs-FIFO aus. An den übrigen Ausgangskanälen – und zwar nur an diesen – können Sie die Pegel einzeln mit Befehlen wie **P2_Digout_Long** setzen.

Die Pegel und Zeitpunkte für die Flankenausgabe werden mit **P2_Digout_Fifo_Write** festgelegt.

Bei einem Modul mit TiCo2-Prozessor sind nach dem Einschalten die vorher eingestellten Spannungspegel aktiv. Aus Sicherheitsgründen sollten Sie die Kanalpegel im Abschnitt **Init**: noch einmal mit

P2_DigProg_Set_IO_Level setzen, bevor Sie die Ausgabe starten.

Siehe auch

[P2_Digout_Fifo_Clear](#), [P2_Dig_Fifo_Mode](#), [P2_Digout_Fifo_Start](#), [P2_Digout_Fifo_Enable](#), [P2_Digout_Fifo_Write](#), [P2_DigProg](#), [P2_DigProg_Set_IO_Level](#), [P2_Digout_Long](#)

Gültig für

[AOut-1/16 Rev. E](#), [DIO-32-TiCo Rev. E](#), [DIO-32-TiCo2 Rev. E](#), [DIO-32/1-TiCo Rev. E](#), [DIO-8-D12 Rev. E](#), [MIO-D12 Rev. E](#), [SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

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 Zählers auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Digout_Fifo_Read_Timer(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Aktueller Stand ($-2^{31}-1 \dots 2^{31}$) des 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 – und bei Modulen mit TiCo2 nur mit einfacher Ausgabe – verwendet werden, d. h. Parameter **mode** = 1 bei **P2_Dig_Fifo_Mode**.

Der Zähler wird regelmäßig um 1 erhöht, so dass der Zähler nach 2^{32} Takten seinen ursprünglichen Wert erneut erreicht. Eine „verpasste“ Flankenausgabe wird erst nach diesem „Überlauf“ ausgegeben. Der Zähler arbeitet je nach Modultyp mit unterschiedlicher Taktrate:

Modul	Taktrate	Taktdauer	Überlaufzeit
DIO-32-TiCo ab Rev. E03 MIO-D12, AOut-1/16	100MHz	10ns	$43s = 10ns \times 2^{32}$
DIO-32-TiCo2, DIO-8-D12	200MHz	5ns	$21s = 5ns \times 2^{32}$

Der Zähler wird mit **Digout_Fifo_Clear** auf Null gesetzt.

Siehe auch

[P2_Dig_Fifo_Mode](#), [P2_Digout_Fifo_Clear](#), [P2_Digout_Fifo_Empty](#), [P2_Digout_Fifo_Start](#), [P2_Digout_Fifo_Write](#), [P2_DigProg](#)

Gültig für

[AOut-1/16 Rev. E](#), [DIO-32-TiCo Rev. E](#), [DIO-32-TiCo2 Rev. E](#), [DIO-32/1-TiCo Rev. E](#), [DIO-8-D12 Rev. E](#), [MIO-D12 Rev. E](#), [SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

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_pattern Bitmuster zum Ansprechen der Module: _LONG |
 Bit = 0: Modul ignorieren.
 Bit = 1: Flankenausgabe auf dem Modul starten.

Bits in module_pattern	31:15	14	13	...	1	0
Moduladresse	–	15	14	...	2	1

Bemerkungen

Auf dem Modul DIO-32-TiCo steht der Ausgangs-FIFO seit Revision E03 zur Verfügung.

Bei einem Modul mit TiCo2-Prozessor sind nach dem Einschalten die vorher eingestellten Spannungspegel aktiv. Aus Sicherheitsgründen sollten Sie die Kanalpegel im Abschnitt **Init**: noch einmal mit

P2_DigProg_Set_IO_Level setzen, bevor Sie die Ausgabe starten.

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 regelmäßig um 1 erhöht, so dass der Zähler nach 2^{32} Takten seinen ursprünglichen Wert erneut erreicht. Bei Zeitvergleichen muss dieser „Überlauf“ berücksichtigt werden, der Zählerstand muss daher im Programm regelmäßig vor dem Überlauf abgefragt werden. Der Zähler arbeitet je nach Modultyp mit unterschiedlicher Taktrate:

Modul	Taktrate	Taktdauer	Überlaufzeit
DIO-32-TiCo ab Rev. E03, MIO-D12, AOut-1/16	100MHz	10ns	$43s = 10ns \times 2^{32}$
DIO-32-TiCo2, DIO-8-D12	200MHz	5ns	$21s = 5ns \times 2^{32}$

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

AOut-1/16 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, MIO-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E

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_Write(module, count, values[],
start_index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl der zu schreibenden Wertepaare: DIO-32-TiCo2, DIO-8-D12: 1...2047 alle anderen Module: 1...511	LONG
values[]	Feld, das abwechselnd Bitmuster der Pegelzustände und Zeitstempel für Ausgabezeitpunkte enthält. Die Zuordnung der Bits zu den Ausgängen bzw. deren Funktion ist unten dargestellt.	LONG ARRAY
start_index	Startindex für das Feld values[] , ab dem die Daten gelesen werden.	LONG

Bitnr.	31	30	...	2	1	0
Ausgang	31	30	...	2	1	0

Modul AOut-1/16:

Bitnr.	31	30	29	...	16	15:0
Ausgang / Funktion	Status TTL-Ausgabe	Status DAC-Ausgabe	29	...	16	DAC-Wert

Bemerkungen

Es dürfen nicht mehr Wertepaare geschrieben werden als im FIFO frei sind. Die Anzahl der freien Wertepaare wird mit **P2_Digout_Fifo_Empty** bestimmt.

Das FIFO-Feld kann maximal 511 bzw. 2047 Wertepaare (Pegelzustand und Zeitstempel) enthalten, je nach Modultyp. Wenn das FIFO-Feld voll ist, können keine weiteren Wertepaare hineingeschrieben werden.

Im Feld **values[]** müssen Wertepaare aus Pegelzustand und zugehörigem Zeitstempel abgelegt sein. Doppelte Zeitangaben beziehen sich (in dieser Reihenfolge) auf die Module a) DIO32-TiCo, MIO-D12 und AOut-1/16 und b) und DIO32-TiCo2:

- Ein Feldelement mit ungeradem Index enthält den Pegelzustand der Kanäle 0...31 als Bitmuster.
Beim Modul AOut-1/16 enthält ein Feldelement zwei Ausgabewerte und 2 Statusbits:
Bits 15:0 enthalten einen Digitalwert für die DAC-Ausgabe.
Bits 29:16 enthalten den Pegel der Ausgänge 16...29 als Bitmuster.
Bits 31:30 legen fest, welche Werte ausgegeben werden. Mit Bit = 1 wird der Wert ausgegeben, mit Bit = 0 bleibt der bisherige Wert unverändert bestehen:
- Bit 30: Bits 15:0 als DAC-Wert ausgeben.
- Bit 31: Bits 29:16 auf TTL-Ausgänge ausgeben.
- Ein Feldelement mit geradem Index enthält einen Zeitstempel (absolut oder relativ, siehe **P2_Dig_Fifo_Mode**). Der Abstand zwischen zwei Zeitstempeln muss mindestens 20ns/10ns betragen. Der Wert eines Zeitstempels wird in Prozessortakten gezählt, also in Einheiten von 10ns / 5ns.
- Beachten Sie: Wenn auf einem Modul mit TiCo2-Prozessor die kontinuierliche Ausgabe aktiviert ist, muss der erste Zeitstempel im Feld **values[]** einen Wert größer gleich 2 (=10ns) haben.

Solange die kontinuierliche Ausgabe läuft, können keine weiteren Werte in den Ausgangs-FIFO geschrieben werden.

Die Ausgabe läuft ab wie folgt:

- Der Modulzähler wird alle 10ns/5ns 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/5ns. 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 152 wird das zugehörige Bitmuster genau 1,52µs/0,76µs nach Start des Modulzählers ausgegeben.

- als Relativwert mit Bezug zum vorherigen Zeitstempel; dazu wird der Modulzähler bei der Bitmuster-Ausgabe auf Null gesetzt.
Bei einem Zeitstempel von 152 wird das zugehörige Bitmuster genau 1,52µs/0,76µs 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. Wenn der FIFO jedoch einmal leer läuft, ist Folgendes zu beachten:

- 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 / 21 Sekunden ausgeführt.
- Bei Relativwerten muss der Zeitstempel größer sein als der Zeitraum seit dem vorherigen Ausgabezeitpunkt (als der FIFO leer lief). Gelingt das nicht, wird das Bitmuster sofort ausgegeben (jedoch offensichtlich 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_Dig-Prog](#)

Gültig für

[AOut-1/16 Rev. E](#), [DIO-32-TiCo Rev. E](#), [DIO-32-TiCo2 Rev. E](#), [DIO-32/1-TiCo Rev. E](#), [DIO-8-D12 Rev. E](#), [MIO-D12 Rev. E](#), [SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

siehe [P2_Dig_Fifo_Mode](#)

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

Modul AOut-1/16:

Bitnr.	31	...	16	15:0
Ausgang	31	...	16	–

Modul SENT-4-Out, SENT-6:

Bitnr.	31:28	27	...	24	23:0
Ausgang	–	19	...	16	–

Andere Module:

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. Das gilt nicht für das Modul AOut-1/16.

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

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, REL-16 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Bei AOUT-1/16: Zeile mit P2_DigProg löschen
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_Long

P2_Digout_Reset

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

Modul AOut-1/16:

Bitnr.	31	...	16	15:0
Ausgang	31	...	16	–

Modul SENT-4-Out, SENT-6:

Bitnr.	31:28	27	...	24	23:0
Ausgang	–	19	...	16	–

Andere Module:

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. Das gilt nicht für das Modul AOut-1/16.

Mit **P2_Digout_Reset** 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

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, REL-16 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Bei AOUT-1/16: Zeile mit P2_DigProg löschen
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_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

Modul AOut-1/16:

Bitnr.	31	...	16	15:0
Ausgang	31	...	16	–

Modul SENT-4-Out, SENT-6:

Bitnr.	31:28	27	...	24	23:0
Ausgang	–	19	...	16	–

Andere Module:

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. Das gilt nicht für das Modul AOut-1/16.

Mit **P2_Digout_Set** können beliebige Ausgänge 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](#), [P2_Digout_Bits](#), [P2_Digout_Long](#), [P2_Digout_Reset](#), [P2_DigProg](#)

Gültig für

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, REL-16 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Bei AOUT-1/16: Zeile mit P2_DigProg löschen
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_Digout_Set

P2_DigProg

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	15:8	7:0

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).

Bei den Modulen DIO-8-D12 und SPI-2-D können nur single-ended-Kanäle konfiguriert werden. Für differentielle Kanäle ist **P2_DigProg_Bits** erforderlich.

Siehe auch

[P2_DigProg_Bits](#), [P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Dig_Write_Latch](#), [P2_DigProg_Set_IO_Level](#)

[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, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, SPI-2-D Rev. E, SPI-2-T 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_Bits programmiert die digitalen Kanäle des angegebenen Moduls einzeln als Ein- oder Ausgang.

Syntax

```
#Include ADwinPro_All.inc

P2_DigProg_Bits (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	30	...	1	0
Kanalnr.	31	30	...	1	0

Bemerkungen

Nach dem Einschalten des Systems sind alle Kanäle als Eingänge konfiguriert. Bei den Modulen DIO-8-D12 und SPI-2-D können nur die differentiellen digitalen Kanäle 0...11 konfiguriert werden. Für single-ended-Kanäle ist **P2_DigProg** erforderlich.

Siehe auch

[P2_DigProg](#), [P2_Dig_Latch](#), [P2_Dig_Read_Latch](#), [P2_Dig_Write_Latch](#), [P2_DigProg_Set_IO_Level](#), [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-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, SPI-2-D Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

```
Rem Kanäle 0...11 des Moduls Nr. 1 als Eingang konfigurieren
Rem und Kanäle 12...31 als Ausgang
P2_DigProg_Bits(1, 0Bh)
```

P2_DigProg_Bits

P2_DigProg_Set_IO_Level

P2_DigProg_Set_IO_Level setzt die Spannungspegel in Gruppen zu je 8 auf einen definierten Wert.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_DigProg_Set_IO_Level(module, group, level)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
group	Nummer (0...3) der Kanalgruppe, deren Spannungspegel gesetzt wird: 0: Kanäle 0...7 1: Kanäle 8...15 2: Kanäle 16...23 3: Kanäle 24...31	LONG
level	Wert in Digits (0...255), um den Spannungspegel der Digitalkanäle einzustellen. Zuordnung siehe Tabelle.	LONG

Digits	Spannungspegel
0	1,6V
36	1,8V
100	2,2V
132	2,5V
191	3,3V
255	4,7V

Bemerkungen

Nach dem Einschalten sind die vorher eingestellten Spannungspegel aktiv. Wir empfehlen aber aus Sicherheitsgründen, alle verwendeten Kanalpegel im Abschnitt **Init:** zu setzen, bevor die Ausgabe gestartet wird.

Die Spannungspegel können nur für Kanalgruppen zu je 8 gesetzt werden (nur 4 relevante Bits, die anderen Bits werden ignoriert).

Die angegebenen Spannungspegel gelten für den High-Pegel und werden mit einer Toleranz von $\pm 0,1V$ eingehalten. Die Zuordnung zwischen Digitalwert und Spannungspegel ist nicht linear.

Beachten Sie: Das Umstellen auf einen anderen Spannungspegel kann – je nach Spannungsdifferenz und Umschaltrichtung – bis zu vier Millisekunden dauern. Starten Sie die Ausgabe erst nach einer entsprechenden Wartezeit mit **P2_Digout_Fifo_Start**.

Siehe auch

[P2_Digin_Long](#), [P2_Digout](#), [P2_Digout_Long](#), [P2_Digin_Fifo_Enable](#), [P2_Digout_Fifo_Start](#), [P2_DigProg](#), [P2_Get_Digout_Long](#)

Gültig für

DIO-32-TiCo2 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Init:

Rem Spannungspegel 2,8 V für die Kanäle 0..15 einstellen

```
P2_DigProg_Set_IO_Level(1, 0, 160)
```

```
P2_DigProg_Set_IO_Level(1, 1, 160)
```

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 31:0).	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

AOut-1/16 Rev. E, DIO-32 Rev. E, DIO-32-TiCo Rev. E, DIO-32-TiCo2 Rev. E, DIO-32/1-TiCo Rev. E, DIO-8-D12 Rev. E, REL-16 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E, SPI-2-D Rev. E, SPI-2-T Rev. E, TRA-16 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

Event:

```
Rem Bits 31:0 aus dem Latch zurücklesen
Par_1 = P2_Get_Digout_Long(1)
```

P2_Get_Digout_Long

3.7 Pro II: Zähler

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit Zählern gelten:

- [P2_Cnt_Clear](#) (Seite 187)
- [P2_Cnt_Enable](#) (Seite 188)
- [P2_Cnt_PW_Enable](#) (Seite 189)
- [P2_Cnt_Get_Status](#) (Seite 190)
- [P2_Cnt_Get_PW](#) (Seite 192)
- [P2_Cnt_Get_PW_HL](#) (Seite 193)
- [P2_Cnt_Latch](#) (Seite 194)
- [P2_Cnt_Mode](#) (Seite 195)
- [P2_Cnt_PW_Latch](#) (Seite 197)
- [P2_Cnt_Read](#) (Seite 198)
- [P2_Cnt_Read4](#) (Seite 199)
- [P2_Cnt_Read_Int_Register](#) (Seite 200)
- [P2_Cnt_Read_Latch](#) (Seite 201)
- [P2_Cnt_Read_Latch4](#) (Seite 202)
- [P2_Cnt_Sync_Latch](#) (Seite 203)
- [P2_SSI_Mode](#) (Seite 204)
- [P2_SSI_Read](#) (Seite 205)
- [P2_SSI_Read2](#) (Seite 206)
- [P2_SSI_Set_Bits](#) (Seite 207)
- [P2_SSI_Set_Clock](#) (Seite 208)
- [P2_SSI_Set_Delay](#) (Seite 209)
- [P2_SSI_Start](#) (Seite 210)
- [P2_SSI_Status](#) (Seite 211)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

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

Bitnr.	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
#Define module 1

Init:
    P2_Cnt_Enable(module,0000b) 'alle Zähler stoppen
    P2_Cnt_Mode(module,1,0b)    'Zähler 1 Takt-Richtung
    P2_Cnt_Mode(module,2,0b)    'Zähler 2 Takt-Richtung
    P2_Cnt_Clear(module,11b)    'Zähler 1+2 auf 0 zurücksetzen
    P2_Cnt_Enable(module,11b)   'Zähler 1+2 starten

Event:
    P2_Cnt_Latch(module,11b)    'Zähler 1+2 gleichzeitig latchen
    Par_1 = P2_Cnt_Read_Latch(module,1) 'Latch Zähler 1 und ...
    Par_2 = P2_Cnt_Read_Latch(module,2) 'Latch Zähler 2 auslesen
```

P2_Cnt_Clear

P2_Cnt_Enable

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

Bitnr.	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](#), [MIO-D12 Rev. E](#)

Beispiel

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

Init:
    P2_Cnt_Enable(module, 0000b) 'alle Zähler stoppen
    P2_Cnt_Mode(module, 1, 0b)   'Zähler 1 Takt-Richtung
    P2_Cnt_Mode(module, 2, 0b)   'Zähler 2 Takt-Richtung
    P2_Cnt_Clear(module, 11b)    'Zähler 1+2 auf 0 zurücksetzen
    P2_Cnt_Enable(module, 11b)   'Zähler 1+2 starten

Event:
    P2_Cnt_Latch(module, 11b)    'Zähler 1+2 gleichzeitig latchen
    Par_1 = P2_Cnt_Read_Latch(module, 1) 'Latch Zähler 1 und ...
    Par_2 = P2_Cnt_Read_Latch(module, 2) 'Latch Zähler 2 auslesen
```


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

Bitnr.	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.

PWM-Zähler können zwar angehalten, aber nicht gelöscht werden; nur beim Einschalten der ADwin-Hardware werden die Zähler zurückgesetzt. Wenn Sie die Stände verschiedener Zähler miteinander vergleichen wollen, gehen Sie folgendermaßen vor:

- Vorbereitung, empfohlen im Abschnitt **Init:/LowInit:**
Zähler stoppen
Zähler latchen
Zählerstände auslesen, den Offset der Stände berechnen und merken
Zähler gemeinsam starten
- Nutzen z. B. im Abschnitt **Event:**
Beim Vergleich der Zählerstände den gemerkten Offset berücksichtigen

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_Read_Int_Register](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Beispiel

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

Init:
    P2_Cnt_PW_Enable(module,0000b) 'alle PW-Zähler stoppen
    Rem Zähler 1+2: Modus Takt-Richtung, PWM-Msg. am Eingang CLK
    P2_Cnt_Mode(module,1,0)
    P2_Cnt_Mode(module,2,0)
    P2_Cnt_PW_Enable(module,0011b) 'PWM-Zähler 1+2 starten

Event:
    P2_Cnt_PW_Latch(module,11b) 'Zähler 1+2 gleichzeitig latchen
    REM High-/Low-Zeit lesen
    P2_Cnt_Get_PW_HL(module,1,Par_1,Par_2)
    REM Frequenz und Taktverhältnis lesen
    P2_Cnt_Get_PW(module,1,FPar_1,FPar_2)
```

P2_Cnt_PW_Enable

P2_Cnt_Get_Status

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 4:0 siehe Tabelle.	LONG

Bitnr.	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](#), [MIO-D12 Rev. E](#)

Beispiel

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

Init:
    P2_Cnt_Enable(module,0000b) 'Zähler stoppen
    Rem Zähler 1: Modus Takt-Richtung
    P2_Cnt_Mode(module,1,0)
    P2_Cnt_Clear(module,1b)      'Zähler 1 auf 0 zurücksetzen
    P2_Cnt_Enable(module,1b)     'Zähler 1 starten
    error = 0                   'Fehlerindikator zurücksetzen

Event:
    PAR_1 = P2_Cnt_Read(module,1) 'Zähler 1 lesen
    PAR_2 = P2_Cnt_GetStatus(module,1) And 11111b 'Status
    Rem Leitungs- bzw. Kabelfehler Zähler 1?
    If (PAR_2 And 10000b = 10000b) Then
        Rem Anzahl Leitungs- bzw. Kabelfehler
        Inc PAR_3
        error = 1                'Fehlerindikator setzen
    EndIf
    Rem Korrelationsfehler Zähler 1?
    If (PAR_2 AND 01000b = 01000b) Then
        Inc PAR_4
        error = 1                'Fehlerindikator setzen
    EndIf
    Rem Zustand Eingang CLR
    PAR_5 = Shift_Right(PAR_2 And 100b,2)
    Rem Zustand Eingang A
    PAR_6 = Shift_Right(PAR_2 And 10b,1)
    Rem Zustand Eingang B
    PAR_7 = PAR_2 And 1b
```

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_Read_Int_Register](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Beispiel

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

Init:

```
P2_Cnt_PW_Enable(module, 0000b) 'alle Zähler stoppen
Rem Zähler 1+2: Modus Takt-Richtung, PWM-Msg. am Eingang CLK
P2_Cnt_Mode(module, 1, 0)
P2_Cnt_Mode(module, 2, 0)
P2_Cnt_PW_Enable(module, 0011b) 'PWM-Zähler 1+2 starten
```

Event:

```
P2_Cnt_PW_Latch(module, 11b) 'Zähler 1+2 gleichzeitig latchen
REM High-/Low-Zeit lesen
P2_Cnt_Get_PW_HL(module, 1, Par_1, Par_2)
REM Frequenz und Taktverhältnis lesen
P2_Cnt_Get_PW(module, 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 (Pegel High) des PWM-Signals in Einheiten von 10ns.	LONG CONST
lowtime	Austastzeit (Pegel Low) 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_Read_Int_Register](#), [P2_Cnt_Sync_Latch](#)

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Beispiel

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

Init:
P2_Cnt_PW_Enable(module, 0000b) 'alle Zähler stoppen
Rem Zähler 1+2: Modus Takt-Richtung, PWM-Msg. am Eingang CLK
P2_Cnt_Mode(module, 1, 0)
P2_Cnt_Mode(module, 2, 0)
P2_Cnt_PW_Enable(module, 0011b) 'PWM-Zähler 1+2 starten

Event:
P2_Cnt_PW_Latch(module, 11b) 'Zähler 1+2 gleichzeitig latches
REM High-/Low-Zeit lesen
P2_Cnt_Get_PW_HL(module, 1, Par_1, Par_2)
REM Frequenz und Taktverhältnis lesen
P2_Cnt_Get_PW(module, 1, FPar_1, FPar_2)
```

P2_Cnt_Get_PW_HL

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

Bitnr.	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.

Das Übertragen des Zählerstands kann synchron mit Aktionen auf anderen Modulen gestartet werden. Verwenden Sie hierzu den Befehl **P2_Sync_All**. Bei den Modulen MIO-4-ET1 und MIO-D12 können Sie einzelne Zähler mit **P2_Sync_Enable** für die Synchronisierung deaktivieren.

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, MIO-D12 Rev. E

Beispiel

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

Init:
    P2_Cnt_Enable(module, 0000b) 'alle Zähler stoppen
    P2_Cnt_Mode(module, 1, 0b)   'Zähler 1 Takt-Richtung
    P2_Cnt_Mode(module, 2, 0b)   'Zähler 2 Takt-Richtung
    P2_Cnt_Clear(module, 11b)    'Zähler 1+2 auf 0 zurücksetzen
    P2_Cnt_Enable(module, 11b)   'Zähler 1+2 starten

Event:
    P2_Cnt_Latch(module, 11b)    'Zähler 1+2 gleichzeitig latchen
    Par_1 = P2_Cnt_Read_Latch(module, 1) 'Latch Zähler 1 und ...
    Par_2 = P2_Cnt_Read_Latch(module, 2) 'Latch Zähler 2 auslesen
```

P2_Cnt_Mode definiert die Betriebsart eines Zählers.

Syntax

```
#Include ADwinPro_All.inc

P2_Cnt_Mode(module, cnt_no, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
cnt_no	Zählernummer: 1...4.	LONG
pattern	Bitmuster zur Einstellung des Betriebsmodus des Zählers.	LONG

Bitnr.	Bedeutung
Bit 0	Zählermodus: Bit = 0: Takt-Richtungs-Modus. Bit = 1: Vier-Flanken-Auswertung (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 mit Vier-Flanken-Auswertung.
Bit 2	Eingang A / CLK invertieren (keine Wirkung auf PWM-Zähler): Bit = 0: Eingang ist nicht invertiert. Bit = 1: Eingang ist invertiert.
Bit 3	Eingang B / DIR invertieren (keine Wirkung auf PWM-Zähler): 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 Signalflanke für PWM-Auswertung. Bit = 0: steigende Flanke. Bit = 1: fallende Flanke.
Bit 8:7	Auswahl eines Eingangs für PWM-Auswertung. 00b: Eingang A / CLK 01b: Eingang B / DIR 10b: Eingang CLR / LATCH Bei CNT-T Pin-Bindung an Zählermodus beachten (siehe unten).
Bits 31:9	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.

Beim Modul CNT-T Rev. E sind die PWM-Eingangs-Pins A und B nur in Verbindung mit dem Zählermodus Vier-Flanken-Auswertung einsetzbar und die PWM-Eingangs-Pins CLK und DIR nur in Verbindung mit dem Takt-Richtungs-Modus.

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](#), [MIO-D12 Rev. E](#)

Beispiel

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

Init:

```
P2_Cnt_Enable(module,0000b) 'alle Zähler stoppen
P2_Cnt_Mode(module,1,0b)    'Zähler 1 Takt-Richtung
P2_Cnt_Mode(module,2,0b)    'Zähler 2 Takt-Richtung
P2_Cnt_Clear(module,11b)    'Zähler 1+2 auf 0 zurücksetzen
P2_Cnt_Enable(module,11b)   'Zähler 1+2 starten
```

Event:

```
P2_Cnt_Latch(module,11b)    'Zähler 1+2 gleichzeitig latchen
Par_1 = P2_Cnt_Read_Latch(module,1) 'Latch Zähler 1 und ...
Par_2 = P2_Cnt_Read_Latch(module,2) 'Latch Zähler 2 auslesen
```


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

Bitnr.	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_Read_Int_Register**, **P2_Cnt_Sync_Latch**

Gültig für

CNT-D Rev. E, CNT-I Rev. E, CNT-T Rev. E, MIO-4-ET1 Rev. E, MIO-D12 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#define module 1

Init:
    P2_Cnt_PW_Enable(module,0000b) 'alle Zähler stoppen
    Rem Zähler 1+2: Modus Takt-Richtung, PWM-Msg. am Eingang CLK
    P2_Cnt_Mode(module,1,0)
    P2_Cnt_Mode(module,2,0)
    P2_Cnt_PW_Enable(module,0011b) 'PWM-Zähler 1+2 starten

Event:
    P2_Cnt_PW_Latch(module,11b) 'Zähler 1+2 gleichzeitig latchen
    REM High-/Low-Zeit lesen
    P2_Cnt_Get_PW_HL(module,1,Par_1,Par_2)
    REM Frequenz und Taktverhältnis lesen
    P2_Cnt_Get_PW(module,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](#), [MIO-D12 Rev. E](#)

Beispiel

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

Init:
    P2_Cnt_Enable(module,0000b) 'alle Zähler stoppen
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(module,1,100000b)
    Rem Zähler 2: Modus Takt-Richtung, LATCH freigeben
    P2_Cnt_Mode(module,2,110000b)
    P2_Cnt_Clear(module,11b) 'Zähler 1+2 auf 0 zurücksetzen
    P2_Cnt_Enable(module,11b) 'Zähler 1+2 starten

Event:
    Par_1 = P2_Cnt_Read(module,1) 'Zähler 1 und ...
    Par_2 = P2_Cnt_Read(module,2) 'Zähler 2 auslesen
```

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
#Define module 1
Dim Data_1[4] As Long
Dim old[4], new[4], i As Long

Init:
    P2_Cnt_Enable(module,0000b) 'alle Zähler stoppen
    Rem Zähler 1: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(module,1,100000b)
    Rem Zähler 2: Modus Takt-Richtung, LATCH freigeben
    P2_Cnt_Mode(module,2,100000b)
    Rem Zähler 3: Modus Takt-Richtung, CLR freigeben
    P2_Cnt_Mode(module,3,110000b)
    Rem Zähler 4: Modus Takt-Richtung, LATCH freigeben
    P2_Cnt_Mode(module,4,100000b)
    P2_Cnt_Clear(module,1111b) 'Alle Zähler auf 0 zurücksetzen
    P2_Cnt_Enable(module,1111b) 'Zähler starten

Event:
    P2_Cnt_Read4(module,new,1) 'Zählerstände in Feld new
    einlesen
    For i = 1 To 4
        Data_1[i] = new[i]-old[i] 'Differenz (f = Impulse / Zeit)
        old[i] = new[i] 'Neuen Zählerstand 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(module,
                                     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

Zu jedem PWM-Zähler gehören die oben angegebenen Register. Wenn Sie die PWM-Zähler mit den Standard-Befehlen **P2_Cnt_Get_PW** und **P2_Cnt_Get_PW_HL** auswerten, benötigen Sie keine Kenntnisse über die PWM-Register. Nur für spezielle Lösungen ist es sinnvoll, wenn Sie die PWM-Register selbst auswerten.

Registerinhalte werden mit **P2_Cnt_PW_Latch** oder **P2_Cnt_Sync_Latch** gesetzt.

Zur Auswertung der PWM-Register beachten Sie die Hinweise im Handbuch *ADwin-Pro II Hardware*, ModulPro II-CNT-x Rev. E Pro II-CNT-x Rev. E.

Siehe auch

[P2_Cnt_PW_Enable](#), [P2_Cnt_Get_PW](#), [P2_Cnt_Get_PW_HL](#), [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](#), [MIO-D12 Rev. E](#)

Beispiel

siehe [P2_Cnt_Sync_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](#), [MIO-D12 Rev. E](#)

Beispiel

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

Init:
    P2_Cnt_Enable(module,0000b) 'alle Zähler stoppen
    Rem Zähler 2 Takt-Richtung, Latch-Eingang freigeben
    P2_Cnt_Mode(module,2,110000b)
    P2_Cnt_Clear(module,10b) 'Zähler 2 auf 0 zurücksetzen
    P2_Cnt_Enable(module,10b) 'Zähler 2 starten

Event:
    P2_Cnt_Latch(module,0010b) 'Zähler 2 latchen
    Par_10 = P2_Cnt_Read_Latch(module,2) 'Latch Zähler 2 lesen
```

P2_Cnt_Read_Latch

P2_Cnt_Read_Latch4

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
#Define module 1
```

```
Dim Data_1[4] As Long
Dim old[4], new[4] As Long
Dim i As Long
```

Init:

```
P2_Cnt_Enable(module,0) 'Zähler stoppen
Rem Zähler 1..4 Modus Takt-Richtung
P2_Cnt_Mode(module,1,0b)
P2_Cnt_Mode(module,2,0b)
P2_Cnt_Mode(module,3,0b)
P2_Cnt_Mode(module,4,0b)
P2_Cnt_Clear(module,1111b) 'Alle Zähler auf 0 zurücksetzen
P2_Cnt_Enable(module,1111b) 'Zähler starten
```

Event:

```
P2_Cnt_Latch(module,1111b) 'Zähler gleichzeitig latchen
P2_Cnt_Read_Latch4(module,new,1) 'Zähler in Feld new einlesen
For i = 1 To 4
    Data_1[i] = new[i]-old[i] 'Differenz (f = Impulse / Zeit)
    old[i] = new[i] 'Neuen Zählerstand speichern
Next i
```

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

Bitnr.	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_Int_Register](#), [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, MIO-D12 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define module 1
#Define frequency PAR_1
Dim time, edges As Long
Dim pw, oldpw As Long
Dim vr, oldvr As Long

Init:
    Processdelay = 3000000 '100Hz with T11 processor
    P2_Cnt_Enable(module,0) 'counters off
    P2_Cnt_Mode(module,1,00000000b) 'mode: clock/dir
    P2_Cnt_Clear(module,0001b) 'clear counter 1
    P2_Cnt_Enable(module,0001b) 'enable V/R counter 1
    P2_Cnt_PW_Enable(module,0001b) 'enable PWM counter 1
    P2_Cnt_Sync_Latch(module,0001b) 'latch counter 1 (V/R + PWM)
    oldvr = P2_Cnt_Read_Int_Register(module,1,6) 'V/R counter 1
    oldpw = P2_Cnt_Read_Int_Register(module,1,8) 'PWM counter 1
    frequency = 0

Event:
    P2_Cnt_Sync_Latch(module,0001b) 'latch counter 1 (V/R + PWM)
    vr = P2_Cnt_Read_Int_Register(module,1,6) 'V/R counter 1
    edges = Abs(vr - oldvr) 'number of edges between events
    If (edges <> 0) Then
        Rem get positive edges latch 1
        pw = P2_Cnt_Read_Int_Register(module,1,8)
        time = pw - oldpw 'calculate time base
        Rem frequency: 100000000=timer frequency of CNT module
        frequency = edges * 100000000 / time
        oldvr = vr 'store VR counter value
        oldpw = pw 'store PW counter value
    EndIf
```

P2_Cnt_Sync_Latch

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, MIO-D12 Rev. E

Beispiel

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

Init:

```
P2_SSI_Set_Clock(module,200) 'CLK (Taktrate) = 125 kHz
P2_SSI_Set_Delay(module,1,250) 'Wartezeit Decoder 1: 5 µs
P2_SSI_Set_Delay(module,2,500) 'Wartezeit Decoder 2: 10 µs
P2_SSI_Set_Bits(module,1,23) 'Anzahl Bits = 23 (Decoder 1)
P2_SSI_Set_Bits(module,2,23) 'Anzahl Bits = 23 (Decoder 2)
P2_SSI_Mode(module,3) 'Continuous-Modus für beide Decoder
```

Event:

```
Par_1 = P2_SSI_Read(module,1) 'Positionswert Decoder 1 lesen
Par_2 = P2_SSI_Read(module,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 (= Absolutwert-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, MIO-D12 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define module 1
Dim m, n, y As Long

Init:
P2_SSI_Set_Clock(module, 50) 'CLK (Taktrate) = 500 kHz
P2_SSI_Set_Delay(module, 1, 250) 'Wartezeit Decoder: 5 µs
P2_SSI_Set_Bits(module, 1, 23) 'Anzahl Bits = 23 (Decoder 1)
P2_SSI_Mode(module, 1) 'Continuous-Mode setzen (Decoder 1)

Event:
Par_1 = P2_SSI_Read(module, 1) 'Positionswert (Decoder 1) lesen
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

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
#Define module 1
Dim Data_1[2000] As Long

Init:
P2_SSI_Set_Clock(module,50) 'CLK (clock rate) = 500 kHz
P2_SSI_Set_Delay(module,1,250) 'waiting delay decoder 1: 5 µs
P2_SSI_Set_Delay(module,2,1000) 'waiting delay decoder 2: 20 µs
P2_SSI_Set_Bits(module,1,10) '10 bits for decoder 1
P2_SSI_Set_Bits(module,2,25) '25 bits for decoder 2
P2_SSI_Mode(module,3) 'Set continuous-mode (both decoders)
Par_1 = 0

Event:
Inc Par_1
If (Par_1 > 1000) Then Par_1 = 1
P2_SSI_Read2(module,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 (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, MIO-D12 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#define module 1
```

Init:

```
P2_SSI_Set_Clock(module, 50) 'CLK (Taktrate) = 500 kHz
P2_SSI_Set_Delay(module, 1, 250) 'waiting delay decoder 1: 5 µs
P2_SSI_Set_Delay(module, 2, 1000) 'waiting delay decoder 2: 20 µs
P2_SSI_Mode(module, 3) 'set continuous mode for both
P2_SSI_Set_Bits(module, 1, 10) '10 Bits for Decoder 1
P2_SSI_Set_Bits(module, 2, 25) '25 Bits for Decoder 2
```

Event:

```
Par_1 = P2_SSI_Read(module, 1) 'read value of decoder 1
Par_2 = P2_SSI_Read(module, 2) 'read value of decoder 2
```

P2_SSI_Set_Bits



P2_SSI_Set_Clock



P2_SSI_Set_Clock stellt die Taktrate (6,1 kHz bis 12,5 MHz) 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 (2...4095) zur Einstellung der Taktrate nach der Formel: Taktrate = 25MHz / 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 250 kHz.

Bei Teilerfaktoren über 4095 werden die niederwertigsten 12 Bits als Teilerfaktor verwendet.

Die mögliche Taktfrequenz ist abhängig von Kabellänge, Kabeltyp und den jeweils verwendeten Sende- und Empfangsbausteinen des Encoders bzw. Decoders. Grundsätzlich gilt als Regel: Je höher die Taktfrequenz, desto kürzer die mögliche Kabellänge.

Siehe auch

[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, MIO-D12 Rev. E

Beispiel

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

Init:

```
P2_SSI_Set_Clock(module,10) 'CLK (clock rate) = 2.5 MHz
P2_SSI_Set_Delay(module,1,250) 'waiting delay decoder 1: 5 µs
P2_SSI_Set_Delay(module,2,1000) 'waiting delay decoder 2: 20 µs
P2_SSI_Mode(module,3) 'Continuous-Mode setzen
                        '(für beide Decoder)

P2_SSI_Set_Bits(module,1,10) 'Anzahl Bits = 10 (Decoder 1)
P2_SSI_Set_Bits(module,2,25) 'Anzahl Bits = 25 (Decoder 2)
```

Event:

```
Par_1 = P2_SSI_Read(module,1) 'Positionswert (Decoder 1)
auslesen
Par_2 = P2_SSI_Read(module,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, MIO-D12 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#define module 1
```

Init:

```
P2_SSI_Set_Clock(module, 50) 'CLK (Taktrate) = 500 kHz
P2_SSI_Set_Delay(module, 1, 400) 'Zeitabstand 8µs für Decoder 1
P2_SSI_Set_Delay(module, 2, 200) 'Zeitabstand 4µs für Decoder 2
P2_SSI_Set_Bits(module, 1, 10) '10 Bits für Decoder 1
P2_SSI_Set_Bits(module, 2, 25) '25 Bits für Decoder 2
P2_SSI_Mode(module, 3) 'Continuous-Mode für beide Decoder
```

Event:

```
Par_1 = P2_SSI_Read(module, 1) 'Positionswert (Decoder 1)
auslesen
Par_2 = P2_SSI_Read(module, 2) 'Positionswert (Decoder 2)
auslesen
```

P2_SSI_Set_Delay

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, MIO-D12 Rev. E

Beispiel

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

Init:

```
P2_SSI_Set_Clock(module,250) 'CLK (Taktrate) = 100 kHz
P2_SSI_Set_Delay(module,1,250) 'Wartezeit Decoder 1: 5 µs
P2_SSI_Set_Delay(module,2,1000) 'Wartezeit Decoder 2: 20 µs
P2_SSI_Mode(module,0) 'Single shot-Mode einstellen
                          '(beide Zähler)

P2_SSI_Set_Bits(module,1,23) 'Anzahl Bits = 23 (Decoder 1)
P2_SSI_Set_Bits(module,2,23) 'Anzahl Bits = 23 (Decoder 2)
```

Event:

```
P2_SSI_Start(module,3) 'Positionswert von Decoder 1 & 2 lesen
Do 'Für Decoder 1:
Until (P2_SSI_Status(module,1) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = P2_SSI_Read(module,1) 'Positionswert auslesen und anzeigen

Do 'Für Decoder 2:
Until (P2_SSI_Status(module,2) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = P2_SSI_Read(module,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, MIO-D12 Rev. E

Beispiel

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

Init:

```
P2_SSI_Set_Clock(module, 250) 'CLK (Taktrate) = 100 kHz
P2_SSI_Set_Delay(module, 1, 250) 'Wartezeit Decoder 1: 5 µs
P2_SSI_Set_Delay(module, 2, 1000) 'Wartezeit Decoder 2: 20 µs
P2_SSI_Mode(module, 0) 'Single shot-Mode einstellen
                          '(beide Zähler)

P2_SSI_Set_Bits(module, 1, 23) 'Anzahl Bits = 23 (Decoder 1)
P2_SSI_Set_Bits(module, 2, 23) 'Anzahl Bits = 23 (Decoder 2)
```

Event:

```
P2_SSI_Start(module, 3) 'Positionswert von Decoder 1 & 2 lesen
Do 'Für Decoder 1:
Until (P2_SSI_Status(module, 1) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = P2_SSI_Read(module, 1) 'Positionswert auslesen und
anzeigen
Do 'Für Decoder 2:
Until (P2_SSI_Status(module, 2) = 0)
Rem Wenn Positionswert komplett gelesen ist, dann ...
Par_1 = P2_SSI_Read(module, 2) 'Positionswert auslesen und
anzeigen
```

P2_SSI_Status

3.8 Pro II: CAN-Bus

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit CAN-Bus gelten:

- [CAN_Msg](#) (Seite 213)
- [P2_CAN_Interrupt_Source](#) (Seite 214)
- [P2_CAN_Set_LED](#) (Seite 216)
- [P2_En_Interrupt](#) (Seite 217)
- [P2_En_Receive](#) (Seite 218)
- [P2_En_Transmit](#) (Seite 219)
- [P2_Get_CAN_Reg](#) (Seite 220)
- [P2_Init_CAN](#) (Seite 221)
- [P2_Read_Msg](#) (Seite 222)
- [P2_Read_Msg_Con](#) (Seite 224)
- [P2_Set_CAN_Baudrate](#) (Seite 226)
- [P2_Set_CAN_Reg](#) (Seite 227)
- [P2_Transmit](#) (Seite 228)
- [P2_Transmit_Status](#) (Seite 230)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

CAN_Msg ist ein eindimensionales Feld mit 9 Elementen, in dem Message-Objekte (Nachrichten) des CAN-Busses beim Senden und Empfangen gespeichert sind oder werden.

Syntax

```
#Include ADwinPro_All.inc
```

```
CAN_Msg[n] = value
```

oder

```
value = CAN_Msg[n]
```

Parameter

n	Elementnummer im Feld CAN_Msg (1... 9)	LONG
value	Wert (8 Bit), der in das Message-Objekt geschrieben oder daraus gelesen wird.	LONG

Bemerkungen

Die Elemente des Felds **CAN_Msg** [] haben folgende Funktion:

Elementnr. in CAN_Msg	1...8	9
Inhalt	Message-Objekt(e) = Datenbyte(s)	Anzahl (0...8) belegter Datenbytes

Tragen Sie die zu übertragenden Datenbytes und ihre Anzahl in das Feld **CAN_Msg** [] ein, bevor Sie diese mit **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)
```

```
'Par_1 = Cast_Float32ToLong(pi) 'korrekte Syntax für T12
```

```
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
```

CAN_Msg

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

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Bitmuster, das die Interrupt-Quelle angibt.	LONG

Bitnr.	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 und ein extern gesteuerter Prozess verwendet wird.

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
```

```
REM to be run as externally controlled process (compiler options)
```

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

module	Eingestellte Moduladresse (1...15).	_LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	_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

Sie schalten die obere LED auf der Frontblende mit **P2_Set_LED** ein oder aus.

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-Objektes 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

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_Interrupt



P2_En_Receive

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 ($0 \dots 2^{11}$ oder $0 \dots 2^{29}$).	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](#), [P2_Transmit_Status](#)

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_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

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
id	Identifier der Nachrichten, die in diesem Message-Objekt gesendet werden sollen ($0 \dots 2^{11}$ oder $0 \dots 2^{29}$).	LONG
id_extend	Merker für die Länge des Identifiers: 0: 11 Bit Identifier 1: 29 Bit Identifier	LONG

Bemerkungen

Erst wenn ein Message-Objekt mit **En_Transmit** zum Senden freigegeben ist, kann das Objekt Nachrichten auf dem CAN-Bus senden.

Siehe auch

[CAN_Msg](#), [P2_En_Receive](#), [P2_Init_CAN](#), [P2_Read_Msg](#), [P2_Transmit](#), [P2_Transmit_Status](#)

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_En_Transmit

P2_Get_CAN_Reg

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 (Address map). 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_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_Init_CAN

P2_Read_Msg

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 Nachrichteninhalte 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 **P2_En_Receive** zum Empfangen frei.
- Sooft erforderlich: Prüfen Sie auf eine neue Nachricht und – falls vorhanden – speichern die Nachricht in **CAN_Msg** mit **P2_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](#), [P2_Transmit_Status](#)

Gültig für

[CAN-2 Rev. E](#)

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)
```

'FPar_1 = Cast_LongToFloat32(Par_1) 'korrekte Syntax für T12

```
EndIf
```

P2_Read_Msg_Con

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 **P2_Read_Msg** stellt **P2_Read_Msg_Con** sicher, dass die Nachricht konsistent ist: Wenn während des Auslesens eine neue Nachricht eintrifft, wird die neuere Nachricht zurückgegeben.

Um eine Nachricht zu empfangen, müssen Sie folgende Reihenfolge einhalten:

- Einmal: Geben Sie das Message-Objekt mit **P2_En_Receive** zum Empfangen frei.
- So oft erforderlich: Prüfen Sie auf eine neue Nachricht und – falls vorhanden – speichern die Nachricht in **CAN_Msg** mit **P2_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](#)

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)
```

'FPar_1 = Cast_LongToFloat32(Par_1) 'korrekte Syntax für T12

```
EndIf
```

P2_Set_CAN_Baudrate

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 (1 MBit/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_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

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
regno	Register-Nummer (0...255) des CAN-Controllers	LONG
value	Wert (0...255), der in das Controller-Register geschrieben werden soll.	LONG

Bemerkungen

Sie finden die Registernummern des CAN-Controllers AN82527 im Intel®-Datenblatt.

Siehe auch

[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_Set_CAN_Reg

P2_Transmit

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 **P2_En_Transmit** zum Senden frei.
- So oft erforderlich: Geben Sie die Nachricht in das Feld **CAN_Msg** ein: Die Datenbytes und die Anzahl der Datenbytes.
- Vor dem Senden: Fragen Sie mit **P2_Transmit_Status** ab, ob das Message-Objekt zum Senden bereit ist.
- Senden Sie die Nachricht mit **P2_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](#), [P2_Transmit_Status](#)

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)
    'Par_1 = Cast_Float32ToLong(pi) 'korrekte Syntax für T12

    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_Transmit_Status

P2_Transmit_Status gibt zurück, ob ein Message-Objekt bereit ist zum Senden.

Syntax

```
#Include ADwinPro_All.inc
```

```
ret_val = P2_Transmit_Status(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
ret_val	Status des Message-Objekts. 0: Bereit zum Senden. 1: Nicht bereit zum Senden.	LONG

Bemerkungen

Der Rückgabewert ist nur für solche Message-Objekte sinnvoll, die zum Senden konfiguriert sind.

Ein Message-Objekt, das nicht bereit zum Senden ist, enthält noch eine Nachricht, die gesendet werden soll oder gerade gesendet wird.

Die CAN-Schnittstelle sendet die Nachricht, sobald das Message-Objekt Zugriffsrecht auf den CAN-Bus hat.

Sie können Nachrichten auch verschicken, ohne vorher den Status des Message-Objekts abzufragen. Wenn Sie jedoch Nachrichten schneller bereitstellen als der CAN-Controller sie verschicken kann, gehen einzelne Nachrichten verloren.

Siehe auch

[CAN_Msg](#), [P2_En_Receive](#), [P2_En_Transmit](#), [P2_Read_Msg](#), [P2_Transmit](#)

Gültig für

CAN-2 Rev. E

Beispiel

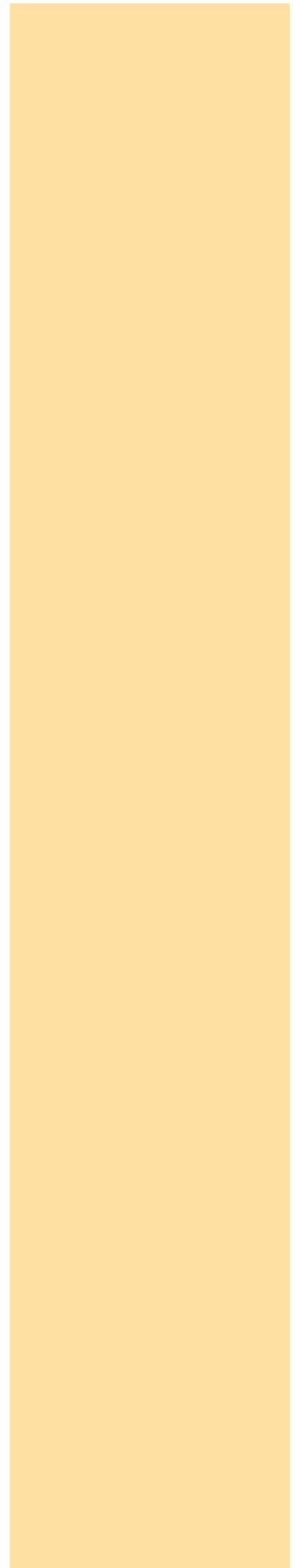
```
#Include ADwinPro_All.inc
```

Init:

```
P2_Init_CAN(1,1)           'CAN-Controller initialisieren
P2_En_Transmit(1,1,6,40,0) 'Message-Objekt 6 initialisieren
Par_1 = 0
CAN_Msg[1] = Par_1         'Wert setzen
CAN_Msg[9] = 1             'Länge der Nachricht in Bytes
```

Event:

```
Inc(Par_1)
CAN_Msg[1] = Par_1         'Wert setzen
If (P2_Transmit_Status(1,1,6) = 0) Then 'bereit zum Senden?
    P2_Transmit(1,1,6)      'Message-Objekt 6 senden
EndIf
If (Par_1 = 255) Then Par_1 = 0
```



3.9 Pro II: CAN FD-Bus

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit CAN FD-Bus gelten:

- [P2_CANFD_Set_LED](#)
- [P2_CANFD_Init_Datatable](#)
- [P2_CANFD_Init_Controller](#)
 - [P2_CANFD_Enable_Receive_Fifo](#)
 - [P2_CANFD_Enable_Transmit_Fifo](#)
 - [P2_CANFD_Enable_Transmit_Queue](#)
 - [P2_CANFD_Enable_Transmit_Event_Fifo](#)
 - [P2_CANFD_Set_Baudrate_Nominal](#)
 - [P2_CANFD_Set_Baudrate_Data](#)
 - [P2_CANFD_Set_SID11](#)
 - [P2_CANFD_Set_TDC](#)
 - [P2_CANFD_Set_Mode](#)
- [P2_CANFD_Get_Fifo_State](#)
- [P2_CANFD_Read_RMO](#)
- [P2_CANFD_Get_Header_Parts](#)
 - [P2_CANFD_Get_ID](#)
 - [P2_CANFD_Get_ESI](#)
 - [P2_CANFD_Get_FDF](#)
 - [P2_CANFD_Get_BRS](#)
 - [P2_CANFD_Get_RTR](#)
 - [P2_CANFD_Get_IDE](#)
 - [P2_CANFD_Get_DLC](#)
 - [P2_CANFD_Get_SEQ](#)
- [P2_CANFD_Read_EFO](#)
- [P2_CANFD_Write_TMO](#)
- [P2_CANFD_Transmit_Msg](#)
- [P2_CANFD_Transmit_Multi_Msg](#)
- [P2_CANFD_Get_TREC](#)
- [P2_CANFD_Get_BDIAG0](#)
- [P2_CANFD_Get_BDIAG1](#)

P2_CANFD_Set_LED schaltet die Zusatz-LED für einen CAN FD-Kanal auf dem Modul ein oder aus.

Syntax

```
#Include ADwinPro_All.inc

P2_CANFD_Set_LED (module, channel, led)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des CAN-Kanals.	LONG
led	Status der Zusatz-LED: 0: LED aus. 1: LED ein.	LONG

Bemerkungen

Sie schalten die obere LED auf der Frontblende mit **P2_Set_LED** ein oder aus.

Siehe auch

[P2_Set_LED](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.adl` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas` im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

```
#Include ADwinPro_All.inc
```

```
Init:
```

```
P2_CANFD_Set_LED(1,2,1) 'Schalte LED 2 ein
```

P2_CANFD_Set_LED

P2_CANFD_Init_Datatable

P2_CANFD_Init_Datatable initialisiert ein Datenfeld für Informationen des CAN FD-Controllers.

Syntax

```
#Include ADwinPro_All.inc

P2_CANFD_Init_Datatable(module, channel,
    canfd_data[])
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>channel</code>	Nummer (1, 2) des CAN-Kanals.	LONG
<code>canfd_data[]</code>	Feld, das CAN-Informationen und -Einstellungen aufnimmt.	ARRAY
	Das Feld muss mindestens 512 Elemente haben.	LONG

Bemerkungen

Verwenden Sie **P2_CANFD_Init_Datatable**, bevor Sie mit einem anderen Befehl auf den CAN-Controller zugreifen. Mit dem Befehl werden die Modulnummer und die Nummer des CAN-Kanals für alle nachfolgenden Befehle festgelegt.

Folgende Befehle ändern Einstellungen im Datenfeld `canfd_data[]`:

- P2_CANFD_Enable_Receive_Fifo
- P2_CANFD_Enable_Transmit_Fifo
- P2_CANFD_Enable_Transmit_Queue
- P2_CANFD_Enable_Transmit_Event_Fifo
- P2_CANFD_Set_Baudrate_Nominal
- P2_CANFD_Set_Baudrate_Data
- P2_CANFD_Set_SID11
- P2_CANFD_Set_TDC
- P2_CANFD_Set_Mode

Siehe auch

[P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#), [P2_CANFD_Set_Mode](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche ADwinCANFDUserInterface.adb und zugehöriges Programm Pro2_CANFD_sample_Code.bas im Ordner C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD.

```
#Include ADwinPro_All.inc
#Define module 1
#Define channel 2          'no. of CANFD controller
#Define IDE 0              'identifier extention flag
#Define DataCANFD2 Data_3
#Define err Par_10
Dim DataCANFD2[512] As Long

Init:
P2_CANFD_Init_Datatable(module, channel, DataCANFD2)
REM Fifo 1: receive messages
err = P2_CANFD_Enable_Receive_Fifo(1, 07H, 004H, 1, 005H,
01FFFFFFFH, IDE, DataCANFD2)
REM Fifo 3: send messages
err = P2_CANFD_Enable_Transmit_Fifo(3, 07H, 004H,
DataCANFD2)
REM transmit queue = Fifo 0
err = P2_CANFD_Enable_Transmit_Queue(07H, 004H, DataCANFD2)
err = P2_CANFD_Enable_Transmit_Event_Fifo(004H, 1, DataCANFD2)
err = P2_CANFD_Set_Baudrate_Nominal(1000000, DataCANFD2)
err = P2_CANFD_Set_Baudrate_Data(2000000, DataCANFD2)
P2_CANFD_Set_TDC(1, 0, 010h, DataCANFD2)
P2_CANFD_Set_Mode(CANFD_MODE_CANFD, DataCANFD2)
If (P2_CANFD_Init_Controller(DataCANFD2) = CANFD_INIT_NIO)
Then err = 1
```

P2_CANFD_Init_Controller

P2_CANFD_Init_Controller initialisiert einen CAN-Controller auf dem angegebenen Modul und setzt die vorbereiteten Einstellungen.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val =  
    P2_CANFD_Init_Controller(canfd_data[])
```

Parameter

<code>canfd_data[]</code>	Feld mit vorbereiteten CANFD-Einstellungen.	ARRAY LONG
<code>ret_val</code>	Status der Befehlsausführung. 0: OK, Controller ist initialisiert. <>0:Fehler bei der Initialisierung.	LONG

Bemerkungen:

Verwenden Sie erst **P2_CANFD_Init_Datatable**, um ein Datenfeld `canfd_data[]` einzurichten, und beenden Sie alle Einstellungen, bevor Sie den Controller initialisieren.

Folgende Befehle ändern Einstellungen im Datenfeld `canfd_data[]`:

- P2_CANFD_Enable_Receive_Fifo
- P2_CANFD_Enable_Transmit_Fifo
- P2_CANFD_Enable_Transmit_Queue
- P2_CANFD_Enable_Transmit_Event_Fifo
- P2_CANFD_Set_Baudrate_Nominal
- P2_CANFD_Set_Baudrate_Data
- P2_CANFD_Set_SID11
- P2_CANFD_Set_TDC
- P2_CANFD_Set_Mode

Siehe auch

[P2_CANFD_Init_Datatable](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#), [P2_CANFD_Set_Mode](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Enable_Receive_Fifo definiert einen Fifo für eingehende CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Enable_Receive_Fifo(fifo_no, pls,
    fifo_size, timestamp, id, mask, ide,
    canfd_data[])
```

Parameter

fifo_no	Nummer (1...31) des Fifo.	LONG
pls	Reservierter Platz pro CAN-Nachricht (payload size): 0: 8 Bytes 1: 12 Bytes 2: 16 Bytes 3: 20 Bytes 4: 24 Bytes 5: 32 Bytes 6: 48 Bytes 7: 64 Bytes	LONG
fifo_size	Anzahl (1...32) der CAN-Nachrichten, die der Fifo speichern kann.	LONG
timestamp	Empfangs-Zeitstempel: 0: kein Zeitstempel. 1: Zeitstempel zur CAN-Nachricht wird gespeichert.	LONG
id	Identifizier der Nachrichten, die mit diesem Fifo empfangen werden ($0 \dots 2^{11}-1$ oder $0 \dots 2^{29}-1$). Mit mask werden signifikante Bits festgelegt.	LONG
mask	Maske, die signifikante Bits für den Identifizier id festlegt: 0: Bit muss nicht passen (don't care). 1: Bit muss korrekt sein (must match).	LONG
ide	Merker für die Länge des Identifiziers: 0: 11 Bit Identifizier 1: 29 Bit Identifizier Für 12 Bit-Identifizier siehe P2_CANFD_Set_SID11 .	LONG
canfd_data[]	Feld, das CAN-Informationen und -Einstellungen aufnimmt.	ARRAY LONG
ret_val	Befehlsstatus: 0: Fifo ist angelegt. -1: Speicher ist voll, Fifo konnte nicht angelegt werden.	LONG

Bemerkungen

Die Modulnummer und der CAN FD-Controller werden mit **P2_CANFD_Init_Datatable** festgelegt und sind im Feld **canfd_data[]** abgelegt. Alle Einstellungen werden mit **P2_CANFD_Init_Controller** in den Controller übertragen.

Im CANFD-Controller stehen insgesamt 2kB Speicher für Eingangs-, Ausgangs- und Überwachungs-Fifo sowie den Ausgabepuffer zur Verfügung. Die Größe eines Eingangs-Fifo ergibt sich aus der Anzahl der Nachrichten **fifo_size**; eine Nachricht setzt sich zusammen aus dem Header (8 Byte), aus den Nachrichten-Bytes (**pls**: 0...64 Bytes) und ggf. aus dem Zeitstempel (4 Byte):

$$\text{Größe Eingangs-Fifo} = \text{fifo_size} * (8 + \text{pls} + \text{timestamp} * 4)$$

In einem Eingangs-Fifo werden CANFD-Nachrichten in der Reihenfolge gespeichert bzw. abgerufen, wie sie empfangen wurden.

P2_CANFD_Enable_Receive_Fifo

Achten Sie darauf, die Nachrichten rechtzeitig aus dem Fifo abzurufen. Wenn der Fifo voll ist und eine Nachricht empfangen wird, dann wird die älteste Nachricht überschrieben und geht damit verloren.

Siehe auch

[P2_CANFD_Get_Fifo_State](#), [P2_CANFD_Read_RMO](#), [P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#), [P2_CANFD_Set_Mode](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Enable_Transmit_Fifo definiert einen Fifo für ausgehende CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Enable_Transmit_Fifo(fifo_no,
    pls, fifo_size, canfd_data[])
```

Parameter

fifo_no	Nummer (1...31) des Fifo.	LONG
pls	Reservierter Platz pro CAN-Nachricht (payload size): 0: 8 Bytes 1: 12 Bytes 2: 16 Bytes 3: 20 Bytes 4: 24 Bytes 5: 32 Bytes 6: 48 Bytes 7: 64 Bytes	LONG
fifo_size	Anzahl (1...32) der CAN-Nachrichten, die der Fifo speichern kann.	LONG
canfd_data[]	Feld, das CAN-Informationen aufnimmt.	ARRAY LONG
ret_val	Befehlsstatus: 0: Fifo ist angelegt. -1: Speicher ist voll, Fifo konnte nicht angelegt werden.	LONG

Bemerkungen

Die Modulnummer und der CAN FD-Controller werden mit **P2_CANFD_Init_Datatable** festgelegt und sind im Feld **canfd_data[]** abgelegt. Alle Einstellungen werden mit **P2_CANFD_Init_Controller** in den Controller übertragen.

Im CANFD-Controller stehen insgesamt 2kB Speicher für Eingangs-, Ausgangs- und Überwachungs-Fifo sowie den Ausgabepuffer zur Verfügung. Die Größe eines Ausgangs-Fifo ergibt sich aus der Anzahl der Nachrichten **fifo_size**; eine Nachricht setzt sich zusammen aus dem Header (8 Byte) und aus den Nachrichten-Bytes (**pls**: 0...64 Bytes):

$$\text{Größe Ausgangs-Fifo} = \text{fifo_size} * (8 + \text{pls})$$

Aus einem Ausgangs-Fifo werden CANFD-Nachrichten in der Reihenfolge gesendet, wie sie eingestellt wurden.

Siehe auch

[P2_CANFD_Get_Fifo_State](#), [P2_CANFD_Write_TMO](#), [P2_CANFD_Transmit_Msg](#), [P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#), [P2_CANFD_Set_Mode](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Enable_Transmit_Fifo

P2_CANFD_Enable_Transmit_Queue

P2_CANFD_Enable_Transmit_Queue definiert einen Ausgabepuffer für ausgehende CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Enable_Transmit_Queue(pls,
    queue_size, canfd_data[])
```

Parameter

pls	Reservierter Platz pro CAN-Nachricht (payload size): 0: 8 Bytes 1: 12 Bytes 2: 16 Bytes 3: 20 Bytes 4: 24 Bytes 5: 32 Bytes 6: 48 Bytes 7: 64 Bytes	LONG
queue_size	Anzahl (1...32) der CAN-Nachrichten, die der Ausgabepuffer speichern kann.	LONG
canfd_data[]	Feld, das CAN-Informationen aufnimmt.	ARRAY LONG
ret_val	Befehlsstatus: 0: Ausgabepuffer ist angelegt. -1: Speicher ist voll, Ausgabepuffer konnte nicht angelegt werden.	LONG

Bemerkungen

Die Modulnummer und der CAN FD-Controller werden mit P2_CANFD_Init_Datatable festgelegt und sind im Feld canfd_data[] abgelegt. Alle Einstellungen werden mit P2_CANFD_Init_Controller in den Controller übertragen.

Im CANFD-Controller stehen insgesamt 2kB Speicher für Eingangs-, Ausgangs- und Überwachungs-Fifo sowie den Ausgabepuffer zur Verfügung. Die Größe des Ausgabepuffers ergibt sich aus der Anzahl der Nachrichten **queue_size**; eine Nachricht setzt sich zusammen aus dem Header (8 Byte) und aus den Nachrichten-Bytes (**pls**: 0...64 Bytes):

$$\text{Größe Ausgabepuffer} = \text{queue_size} * (8 + \text{pls})$$

Aus dem Ausgabepuffer werden CANFD-Nachrichten in der Reihenfolge ihrer Priorität gesendet, d.h. die Nachricht mit der jeweils kleinsten ID wird zuerst gesendet. Danach entscheidet die Reihenfolge, in der die Nachrichten eingestellt wurden.

Siehe auch

[P2_CANFD_Get_Fifo_State](#), [P2_CANFD_Write_TMO](#), [P2_CANFD_Transmit_Msg](#), [P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#), [P2_CANFD_Set_Mode](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Enable_Transmit_Event_Fifo definiert einen Überwachungs-Fifo für gesendete CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Enable_Transmit_Event_Fifo(
    fifo_size, timestamp, canfd_data[])
```

Parameter

fifo_size	Anzahl (1...32) der gesendeten CAN-Nachrichten, die der Fifo speichern kann.	LONG
timestamp	Sende-Zeitstempel: 0: kein Zeitstempel. 1: Zeitstempel wird in die CAN-Nachricht eingefügt.	LONG
canfd_data[]	Feld, das CAN-Informationen aufnimmt.	ARRAY LONG
ret_val	Befehlsstatus: 0: Fifo ist angelegt. -1: Speicher ist voll, Überwachungs-Fifo konnte nicht angelegt werden.	LONG

Bemerkungen

Die Modulnummer und der CAN FD-Controller werden mit P2_CANFD_Init_Datatable festgelegt und sind im Feld canfd_data[] abgelegt. Alle Einstellungen werden mit P2_CANFD_Init_Controller in den Controller übertragen.

Im CANFD-Controller stehen insgesamt 2kB Speicher für Eingangs-, Ausgangs- und Überwachungs-Fifo sowie den Ausgabepuffer zur Verfügung. Die Größe des Ausgabepuffers ergibt sich aus der Anzahl der Nachrichten **fifo_size**; eine Nachricht setzt sich zusammen aus dem Header (8 Byte) und ggf. aus dem Zeitstempel (**timestamp**: 4 Bytes):

$$\text{Größe Überwachungs-Fifo} = \text{fifo_size} * (8 + \text{timestamp} * 4)$$

Im Überwachungs-Fifo werden für alle gesendeten Nachrichten (aus allen Ausgangs-Fifos und aus dem Ausgabepuffer) jeweils der Header sowie ggf. der zugehörige Zeitstempel gespeichert.

Siehe auch

[P2_CANFD_Get_Fifo_State](#), [P2_CANFD_Read_EFO](#), [P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#), [P2_CANFD_Set_Mode](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Enable_Transmit_Event_Fifo

P2_CANFD_Set_Baudrate_Nominal

P2_CANFD_Set_Baudrate_Nominal stellt die nominale Baudrate ein und speichert die Informationen in einem CANFD-Datenfeld.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_CANFD_Set_Baudrate_Nominal(baud,  
    canfd_data[])
```

Parameter

baud	Nominale Baudrate des CAN-Controllers: 125...1000000 Bit/s.	LONG
canfd_data[]	Feld, das CAN-Informationen aufnimmt.	ARRAY LONG
ret_val	Status der Befehlsausführung: 0: Baudrate ist gesetzt. 1: Baudrate ist unzulässig und kann nicht gesetzt werden.	LONG

Bemerkungen

Mit **P2_CANFD_Set_Baudrate_Data** stellen Sie die Daten-Baudrate ein.

Die Modulnummer und der CAN FD-Controller werden mit **P2_CANFD_Init_Datatable** festgelegt und sind im Feld **canfd_data[]** abgelegt. Alle Einstellungen werden mit **P2_CANFD_Init_Controller** in den Controller übertragen.

Siehe auch

[P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#), [P2_CANFD_Set_Mode](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Set_Baudrate_Data stellt die Daten-Baudrate ein und speichert die Informationen in einem CANFD-Datenfeld.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Set_Baudrate_Data(baud,
    canfd_data[])
```

Parameter

baud	Daten-Baudrate des CAN-Controllers: 2000...8000000 Bit/s.	LONG
canfd_data[]	Feld, das CAN-Informationen aufnimmt.	ARRAY LONG
ret_val	Status der Befehlsausführung: 0: Baudrate ist gesetzt. 1: Baudrate ist unzulässig und kann nicht gesetzt werden.	LONG

Bemerkungen

Mit **P2_CANFD_Set_Baudrate_Nominal** stellen Sie die nominale Baudrate ein.

Die Modulnummer und der CAN FD-Controller werden mit **P2_CANFD_Init_Datatable** festgelegt und sind im Feld **canfd_data[]** abgelegt. Alle Einstellungen werden mit **P2_CANFD_Init_Controller** in den Controller übertragen.

Siehe auch

[P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#), [P2_CANFD_Set_Mode](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Set_Baudrate_Data

P2_CANFD_Set_SID11

P2_CANFD_Set_SID11 stellt ein, ob CAN FD-Nachrichten mit einem Identifier von 11 Bit oder 12 Bit Länge gesendet werden und speichert die Information in einem CANFD-Datenfeld.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_CANFD_Set_SID11(en_sid11, canfd_data[])
```

Parameter

<code>en_sid11</code>	Einstellung für die Länge des Identifiers einer CAN FD-Nachricht im Basisformat: 0: Nachricht mit 11 Bit-Identifier. 1: Nachricht mit 12 Bit-Identifier, Zusatzbit ist aktiviert.	<code>LONG</code>
<code>canfd_data[]</code>	Feld, das CAN-Informationen aufnimmt.	ARRAY <code>LONG</code>

Bemerkungen

Die Modulnummer und der CAN FD-Controller werden mit `P2_CANFD_Init_Datatable` festgelegt und sind im Feld `canfd_data[]` abgelegt. Alle Einstellungen werden mit `P2_CANFD_Init_Controller` in den Controller übertragen.

In einem Controller kann global für alle CAN FD-Nachrichten im Basisformat eingestellt werden, dass ein Identifier mit einem zusätzlichen Bit genutzt wird. Wenn jedoch Nachrichten mit erweiterter Länge (29 Bit) übertragen werden, wird die Einstellung für das Zusatzbit ignoriert.

Siehe auch

[P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_Mode](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Set_TDC stellt einen Ausgleich für Zeitverzögerungen beim Senden ein und speichert die Informationen in einem CANFD-Datenfeld.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_CANFD_Set_TDC(tdcmod, tdcv, tdcv, canfd_data[])
```

Parameter

tdcmod	Einstellung für Ausgleich: 0: Ausgleich deaktiviert. 1: Wert tdcv wird manuell eingegeben. 2: Wert tdcv wird automatisch bestimmt.	LONG
tdco	Offset (-64...63) in Einheiten von 25ns, der zur Verzögerungszeit hinzugezählt wird.	LONG
tdcv	Verzögerungszeit (0...63) in Einheiten von 25ns, wenn tdcmod = 1.	LONG
canfd_data[]	Feld, das CAN-Informationen aufnimmt.	ARRAY LONG

Bemerkungen

Die Modulnummer und der CAN FD-Controller werden mit **P2_CANFD_Init_Datatable** festgelegt und sind im Feld **canfd_data[]** abgelegt. Alle Einstellungen werden mit **P2_CANFD_Init_Controller** in den Controller übertragen.

Beim Senden liest der Controller die Daten auf dem Bus zurück, um nötigenfalls das Senden unterbrechen zu können. Beim Zurücklesen kann (bei Baudraten über 1 MB/s, also nur bei CANFD-Betrieb) eine Verzögerung gegenüber den gesendeten Daten auftreten.

Als Voreinstellung ist der Ausgleich für Zeitverzögerungen deaktiviert.

Die eingestellte Verzögerung setzt sich zusammen aus **tdcv** + **tdco**. Bei automatischem Ausgleich (**tdcmod** = 2) ermittelt der Controller **tdcv** automatisch; beim manuellen Ausgleich verwendet der Controller den übergebenen Wert **tdcv**.

Siehe auch

[P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_Mode](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Set_TDC

P2_CANFD_Set_Mode

P2_CANFD_Set_Mode stellt den CAN-Betriebsmodus eines Controllers ein.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_CANFD_Set_Mode(mode, canfd_data[])
```

Parameter

mode	Betriebsmodus des Controllers. CANFD_MODE_CANFD: CAN FD-Bus. CANFD_MODE_CAN: CAN-Bus (CAN 2.0).	LONG
canfd_data[]	Feld, das CAN-Informationen aufnimmt.	ARRAY LONG

Bemerkungen

Die Modulnummer und der CAN FD-Controller werden mit P2_CANFD_Init_Datatable festgelegt und sind im Feld canfd_data[] abgelegt. Alle Einstellungen werden mit P2_CANFD_Init_Controller in den Controller übertragen.

Siehe auch

[P2_CANFD_Init_Datatable](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Set_Baudrate_Nominal](#), [P2_CANFD_Set_Baudrate_Data](#), [P2_CANFD_Set_SID11](#), [P2_CANFD_Set_TDC](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe [P2_CANFD_Init_Datatable](#)

P2_CANFD_Get_Fifo_State gibt den Betriebszustand eines Fifo zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Get_Fifo_State(module, channel,
    fifo_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des CAN-Kanals.	LONG
fifo_no	Nummer (0...31) des Fifo. Nummer 0 steht für den Ausgabepuffer.	LONG
ret_val	Bitmuster mit Statusbits des Fifos. Gesetzte Bits geben an, dass der Status zutrifft.	LONG

Bit-Nr.	Eingangs-Fifo	Ausgangs-Fifo
0	Fifo ist nicht leer.	Fifo ist nicht voll.
1	Fifo ist weniger als halbvoll.	Fifo ist mehr als halbvoll.
2	Fifo ist leer.	Fifo ist leer.
3	Fifo ist übergelaufen.	–
5	–	Busfehler beim Senden.
6	–	Beim Senden ging der Buszugriff (Arbitration) verloren.
7	–	Senden wurde abgebrochen.
4, 31:8	–	–

Bemerkungen

Wenn beim Eingangs-Fifo Bit 3 gesetzt ist, ist mindestens eine CAN-Nachricht verloren gegangen.

Wenn beim Senden Störungen auftreten, bleibt die Nachricht im Fifo erhalten und wird später erneut gesendet. Sobald erfolgreich gesendet wurde, werden Fehler-Statusbits zurückgesetzt.

Siehe auch

P2_CANFD_Read_RMO, P2_CANFD_Write_TMO

Gültig für

CAN-FD-2 Rev. E

Beispiel

P2_CANFD_Get_Fifo_State

siehe Beispieloberfläche ADwinCANFDUserInterface.adl und zugehöriges Programm Pro2_CANFD_sample_Code.bas im Ordner C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD.

```
#Include ADwinPro_All.inc
#Define module 2
#Define ch 3
#Define InFifo 4
#Define DataCANFD3 Data_5
Dim can_msg[20] As Long
Dim err As Long
Dim DataCANFD3[512] As Long

Init:
    P2_CANFD_Init_Datatable(module, channel, DataCANFD3)
    REM Fifo: receive messages
    err = P2_CANFD_Enable_Receive_Fifo(InFifo, 07H, 004H, 1, 005H,
    01FFFFFFFH, IDE, DataCANFD2)
    P2_CANFD_Init_Controller(DataCANFD3) 'initialize channel

Event:
    REM check for fifo overflow
    If Not ((P2_CANFD_Get_Fifo_State(module, ch, inFifo) And 1000b)
    = 1000b) Then
        REM read CAN message
        Par_10 = P2_CANFD_Read_RMO(inFifo, can_msg, DataCANFD3)
    EndIf
```

P2_CANFD_Read_RMO prüft, ob in einem Fifo eine Nachricht empfangen wurde.

Falls ja, wird das Nachrichtenobjekt in einem Feld gespeichert und der Identifier der Nachricht zurückgegeben.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Read_RMO(fifo_no, canfd_data[],
                             can_msg[])
```

Parameter

<code>fifo_no</code>	Nummer (1...31) des Fifo.	LONG
<code>canfd_data[]</code>	Feld mit CAN-Informationen.	ARRAY
<code>can_msg[]</code>	Feld, das ein Nachrichtenobjekt RMO aufnimmt. Das Feld muss mindestens 20 Elemente haben.	ARRAY
<code>ret_val</code>	≥0: Eine neue Nachricht ist eingegangen, der Wert ist der Identifier des Message-Objektes. -1: Keine neue Nachricht vorhanden.	LONG

Bemerkungen

Sie können eine empfangene Nachricht nur einmal auslesen.

Achten Sie darauf, die Nachrichten rechtzeitig aus dem Fifo abzurufen. Wenn der Fifo voll ist und eine Nachricht empfangen wird, dann wird die älteste Nachricht überschrieben und geht damit verloren.

Im Feld `can_msg[]` ist die empfangene Nachricht gespeichert:

Element-Nr.	Bits 31:24	Bits 23:16	Bits 15:8	Bits 7:0
<code>can_msg[1]</code>	Nachrichten-Header, Teil 1 (Identifier)			
<code>can_msg[2]</code>	Nachrichten-Header, Teil 2			
<code>can_msg[3]</code>	Zeitstempel 32 Bit, optional			
<code>can_msg[4]</code>	CAN-Byte 4	CAN-Byte 3	CAN-Byte 2	CAN-Byte 1
<code>can_msg[5]</code>	CAN-Byte 8	CAN-Byte 7	CAN-Byte 6	CAN-Byte 5
...	...			
<code>can_msg[19]</code>	CAN-Byte 64	CAN-Byte 63	CAN-Byte 62	CAN-Byte 61
<code>can_msg[20]</code>	reserviert			

Mit **P2_CANFD_Get_Header_Parts** erhalten Sie die Bestandteile des Nachrichten-Headers aus `can_msg[]` (Header-Aufbau siehe dort).

Falls mit **P2_CANFD_Enable_Receive_Fifo** (Parameter `timestamp`) eingestellt, enthält das Element `can_msg[3]` einen Zeitstempel für den Zeitpunkt, als die Nachricht vollständig empfangen war. Bei anderer Einstellung enthält `can_msg[3]` keinen sinnvollen Wert.

Sie können mit **P2_CANFD_Get_Fifo_State** besondere Fifo-Eigenschaften prüfen, beispielsweise ob Nachrichten verloren gegangen sind.

Siehe auch

[P2_CANFD_Init_Controller](#), [P2_CANFD_Enable_Receive_Fifo](#), [P2_CANFD_Write_TMO](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

P2_CANFD_Read_RMO

siehe Beispieloberfläche

ADwinCANFDUserInterface

.adi und zugehöriges

Programm Pro2_CANFD_

sample_Code.bas

im Ordner

C:\ADwin\ADbasic\sampl

es_ADwin_ProII\CAN

FD. **P2_CANFD_**
Get_Header_Parts

P2_CANFD_Get_Header_Parts gibt aus einem CAN-Nachrichtenobjekt die Bestandteile des Nachrichten-Headers zurück.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_CANFD_Get_Header_Parts(can_msg[], canfd_data[],  
id, esi, fdf, brs, rtr, ide, dlc)
```

Parameter

can_msg []	Feld mit einem CAN-Nachrichtenobjekt.	ARRAY LONG
canfd_data []	Feld mit CAN-Informationen.	ARRAY LONG
id	Identifizier der CAN-Nachricht (0...2 ¹¹ -1, 0...2 ¹² -1 oder 0...2 ²⁹ -1).	LONG
esi	Status des Sendeknotens (nur CAN FD): 0: Sendeknoten ist fehleraktiv (error active). 1: Sendeknoten ist fehlerpassiv (error passive).	LONG
fdf	CAN-Format (nur CAN FD): 0: Format CAN (high speed). 1: Format CAN FD.	LONG
brs	Einstellung der Baudrate (nur CAN FD): 0: Nominale Baudrate. 1: Wechsel zwischen nominaler und Daten-Baudrate.	LONG
rtr	Remote transmission request (nur CAN 2.0). 0: RTR wird nicht verwendet. 1: Nachricht wird von extern angefordert (RTR).	LONG
ide	Merker für die Länge des Identifiers: 0: 11 Bit Identifier. 1: 29 Bit Identifier.	LONG
dlc	Länge der CAN-Nachricht (data length code): 0...8: 0...8 Bytes 9: 12 Bytes 10: 16 Bytes 11: 20 Bytes 12: 24 Bytes 13: 32 Bytes 14: 48 Bytes 15: 64 Bytes	LONG

Bemerkungen

Um nur einen einzelnen Parameter zu bestimmen, sind jeweils eigene Befehle vorhanden. Für den Parameter SEQ aus einem Nachrichtenobjekt (EFO) müssen Sie den Befehl **P2_CANFD_Get_SEQ** verwenden.

Der Nachrichten-Header wird in den Elementen **can_msg[1..2]** erwartet und ist zusammengesetzt wie folgt:

Bits	31 / 23 15 / 7	30 / 22 14 / 6	29 / 21 13 / 5	28 / 20 12 / 4	27 / 19 11 / 3	26 / 18 10 / 2	25 / 17 9 / 1	24 / 16 8 / 0
can_msg[1]								
31:24	–	–	SID11	EID (17:6)				
23:16	EID (12:5)							
15:8	EID (4:0)					SID (10:8)		
7:0	SID (7:0)							
can_msg[2]								

Bits	31 / 23 15 / 7	30 / 22 14 / 6	29 / 21 13 / 5	28 / 20 12 / 4	27 / 19 11 / 3	26 / 18 10 / 2	25 / 17 9 / 1	24 / 16 8 / 0
31:24	–							
23:16	–							
15:8	–							ESI
7:0	FDF	BRS	RTR	IDE	DLC (3:0)			

Der Identifier `id` wird abhängig von den Einstellungen für `ide` und das Zusatzbit (`en_sid11`, siehe `P2_CANFD_Set_SID11`) gebildet. Wenn `ide=1` ist, wird die Einstellung `en_sid11` ignoriert.

ide	en_sid11	id (29:12)	id (11)	id (10:0)
0	0	–	–	SID (10:0)
0	1	–	SID11	SID (10:0)
1	0	EID (17:0)		SID (10:0)

Siehe auch

[P2_CANFD_Read_RMO](#), [P2_CANFD_Get_ID](#), [P2_CANFD_Get_ESI](#), [P2_CANFD_Get_FDF](#), [P2_CANFD_Get_BRS](#), [P2_CANFD_Get_RTR](#), [P2_CANFD_Get_IDE](#), [P2_CANFD_Get_DLC](#), [P2_CANFD_Get_SEQ](#), [P2_CANFD_Set_SID11](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.ad` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas`
im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Get_ID

P2_CANFD_Get_ID gibt den Parameter ID aus Header eines CAN-Nachrichtenobjekts zurück.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Get_ID(can_msg[], canfd_data[],
                          ide)
```

Parameter

<code>can_msg[]</code>	Feld mit einem CAN-Nachrichtenobjekt.	ARRAY LONG
<code>canfd_data[]</code>	Feld mit CAN-Informationen.	ARRAY LONG
<code>ide</code>	Merker für die Länge des Identifiers: 0: 11 Bit Identifier. 1: 29 Bit Identifier.	LONG
<code>ret_val</code>	Parameter ID. Identifier der CAN-Nachricht ($0 \dots 2^{11}-1$, $0 \dots 2^{12}-1$ oder $0 \dots 2^{29}-1$).	LONG

Bemerkungen

Der Identifier `id` wird abhängig von den Einstellungen für `ide` (siehe **P2_CANFD_Get_IDE**) und das Zusatzbit (`en_sid11`, siehe **P2_CANFD_Set_SID11**) gebildet. Wenn `ide=1` ist, wird die Einstellung `en_sid11` ignoriert.

ide	en_sid11	id (29:12)	id (11)	id (10:0)
0	0	–	–	SID (10:0)
0	1	–	SID11	SID (10:0)
1	0	EID (17:0)		SID (10:0)

Mit **P2_CANFD_Get_Header_Parts** erhalten Sie alle Parameter des Nachrichten-Headers auf einmal.

Siehe auch

[P2_CANFD_Set_SID11](#), [P2_CANFD_Read_RMO](#), [P2_CANFD_Get_IDE](#), [P2_CANFD_Get_ESI](#), [P2_CANFD_Get_FDF](#), [P2_CANFD_Get_BRS](#), [P2_CANFD_Get_RTR](#), [P2_CANFD_Get_DLC](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
REM XXX Beispiel fehlt
```


P2_CANFD_Get_ESI gibt aus einem CAN-Nachrichtenobjekt den Parameter ESI aus dem Nachrichten-Header zurück..

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Get_ESI(can_msg[])
```

Parameter

can_msg []	Feld mit einem CAN-Nachrichtenobjekt.	ARRAY
		LONG
ret_val	Parameter ESI (nur mit CAN FD). Status des Sendeknotens: 0: Sendeknoten ist fehleraktiv (error active). 1: Sendeknoten ist fehlerpassiv (error passive).	LONG

Bemerkungen

Mit **P2_CANFD_Get_Header_Parts** erhalten Sie alle Parameter des Nachrichten-Headers auf einmal.

Siehe auch

[P2_CANFD_Read_RMO](#), siehe Beispieloberfläche [ADwinCANFDUserInterface.ad](#) und zugehöriges Programm [Pro2_CANFD_sample_Code.bas](#) im Ordner [C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD](#). [P2_CANFD_Get_Header_Parts](#), [P2_CANFD_Get_ID](#), [P2_CANFD_Get_FDF](#), [P2_CANFD_Get_BRS](#), [P2_CANFD_Get_RTR](#), [P2_CANFD_Get_IDE](#), [P2_CANFD_Get_DLC](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche [ADwinCANFDUserInterface.ad](#) und zugehöriges Programm [Pro2_CANFD_sample_Code.bas](#) im Ordner [C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD](#).

P2_CANFD_Get_ESI

P2_CANFD_Get_FDF

P2_CANFD_Get_FDF gibt aus einem CAN-Nachrichtenobjekt den Parameter FDF aus dem Nachrichten-Header zurück..

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc  
ret_val = P2_CANFD_Get_FDF(can_msg[])
```

Parameter

can_msg[]	Feld mit einem CAN-Nachrichtenobjekt.	ARRAY
		LONG
ret_val	Parameter FDF (nur mit CAN FD). CAN-Format: 0: Format CAN (high speed). 1: Format CAN FD.	LONG

Bemerkungen

Mit **P2_CANFD_Get_Header_Parts** erhalten Sie alle Parameter des Nachrichten-Headers auf einmal.

Siehe auch

[P2_CANFD_Read_RMO](#), [P2_CANFD_Get_ID](#), [P2_CANFD_Get_ESI](#), [P2_CANFD_Get_BRS](#), [P2_CANFD_Get_RTR](#), [P2_CANFD_Get_IDE](#), [P2_CANFD_Get_DLC](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc  
REM XXX Beispiel fehlt
```

P2_CANFD_Get_BRS gibt aus einem CAN-Nachrichtenobjekt den Parameter BRS aus dem Nachrichten-Header zurück.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Get_BRS(can_msg[])
```

Parameter

can_msg[]	Feld mit einem CAN-Nachrichtenobjekt.	ARRAY
		LONG
ret_val	Parameter BRS (nur mit CAN FD). Einstellung der Baudrate: 0: Nominale Baudrate. 1: Wechsel zwischen nominaler und Daten-Baudrate.	LONG

Bemerkungen

Mit **P2_CANFD_Get_Header_Parts** erhalten Sie alle Parameter des Nachrichten-Headers auf einmal.

Siehe auch

[P2_CANFD_Read_RMO](#), [P2_CANFD_Get_ID](#), [P2_CANFD_Get_ESI](#), [P2_CANFD_Get_FDF](#), [P2_CANFD_Get_RTR](#), [P2_CANFD_Get_IDE](#), [P2_CANFD_Get_DLC](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.adl` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas` im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Get_BRS

P2_CANFD_Get_RTR

P2_CANFD_Get_RTR gibt aus einem CAN-Nachrichtenobjekt den Parameter RTR aus dem Nachrichten-Header zurück.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc  
ret_val = P2_CANFD_Get_RTR(can_msg[])
```

Parameter

can_msg[]	Feld mit einem CAN-Nachrichtenobjekt.	ARRAY LONG
ret_val	Parameter RTR (nur mit CAN 2.0). Remote transmission request. 0: RTR wird nicht verwendet. 1: Nachricht wird von extern angefordert (RTR) .	LONG

Bemerkungen

Mit **P2_CANFD_Get_Header_Parts** erhalten Sie alle Parameter des Nachrichten-Headers auf einmal.

Siehe auch

[P2_CANFD_Read_RMO](#), [P2_CANFD_Get_ID](#), [P2_CANFD_Get_ESI](#), [P2_CANFD_Get_FDF](#), [P2_CANFD_Get_BRS](#), [P2_CANFD_Get_IDE](#), [P2_CANFD_Get_DLC](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc  
REM XXX Beispiel fehlt
```

P2_CANFD_Get_IDE gibt aus einem CAN-Nachrichtenobjekt den Parameter IDE aus dem Nachrichten-Header zurück.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Get_IDE(can_msg[])
```

Parameter

<code>can_msg[]</code>	Feld mit einem CAN-Nachrichtenobjekt.	ARRAY
		LONG
<code>ret_val</code>	Parameter IDE. Merker für die Länge des Identifiers: 0: 11 Bit Identifier. 1: 29 Bit Identifier.	LONG

Bemerkungen

Mit **P2_CANFD_Get_Header_Parts** erhalten Sie alle Parameter des Nachrichten-Headers auf einmal.

Siehe auch

[P2_CANFD_Read_RMO](#), [P2_CANFD_Get_ID](#), [P2_CANFD_Get_ESI](#), [P2_CANFD_Get_FDF](#), [P2_CANFD_Get_BRS](#), [P2_CANFD_Get_RTR](#), [P2_CANFD_Get_DLC](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.adl` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas`
im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Get_IDE

P2_CANFD_Get_DLC

P2_CANFD_Get_DLC gibt aus einem CAN-Nachrichtenobjekt den Parameter DLC aus dem Nachrichten-Header zurück.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_CANFD_Get_DLC(can_msg[])
```

Parameter

<code>can_msg[]</code>	Feld mit einem CAN-Nachrichtenobjekt.	ARRAY LONG
<code>ret_val</code>	Länge der CAN-Nachricht (data length code): 0...8: 0...8 Bytes 9: 12 Bytes 10: 16 Bytes 11: 20 Bytes 12: 24 Bytes 13: 32 Bytes 14: 48 Bytes 15: 64 Bytes	LONG

Bemerkungen

Mit **P2_CANFD_Get_Header_Parts** erhalten Sie alle Parameter des Nachrichten-Headers auf einmal.

Siehe auch

[P2_CANFD_Read_RMO](#), [P2_CANFD_Get_ID](#), [P2_CANFD_Get_ESI](#), [P2_CANFD_Get_FDF](#), [P2_CANFD_Get_BRS](#), [P2_CANFD_Get_RTR](#), [P2_CANFD_Get_IDE](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.ad` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas`
im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Get_SEQ gibt aus einem Nachrichtenobjekt (EFO) den Parameter SEQ zurück.

Der Befehl greift nicht auf den CAN FD-Controller zu.

Syntax

```
#Include ADwinPro_All.inc  
ret_val = P2_CANFD_Get_SEQ(msg[])
```

Parameter

msg[]	Feld mit einem Nachrichtenobjekt aus dem Überwachungs-Fifo.	ARRAY LONG
ret_val	Parameter SEQ. Kennwert (0...63) der gesendeten Nachricht.	LONG

Bemerkungen

Sie erhalten das Nachrichtenobjekt `msg[]` mit **P2_CANFD_Read_EFO**.

Der Parameter SEQ ist ein Kennwert, der mit **P2_CANFD_Write_TMO** an den Controller übergeben wird und im Überwachungs-Fifo die Nachverfolgung von Nachrichten ermöglicht.

Siehe auch

[P2_CANFD_Read_EFO](#), [P2_CANFD_Write_TMO](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.adl` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas`
im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Get_SEQ

P2_CANFD_Read_EFO

P2_CANFD_Read_EFO prüft, ob im Überwachungs-Fifo ein Nachrichtenobjekt vorhanden ist.

Falls ja, wird das Nachrichtenobjekt EFO in einem Feld gespeichert und der Identifier zurückgegeben.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Read_EFO(canfd_data[], msg[])
```

Parameter

canfd_data[]	Feld mit CAN-Informationen.	ARRAY LONG
msg[]	Feld, das ein Nachrichtenobjekt EFO aufnimmt. Das Feld muss mindestens 3 Elemente haben.	ARRAY LONG
ret_val	≥0: Ein neues Nachrichtenobjekt ist eingegangen, der Wert ist der Identifier des Objekts. -1: Kein Nachrichtenobjekt vorhanden.	LONG

Bemerkungen

Sie können ein Nachrichtenobjekt nur einmal auslesen.

Ein Nachrichtenobjekt **msg[]** enthält 3 Elemente:

- **msg[1..2]**: Header der gesendeten CAN-Nachricht.
- **msg[3]**: Zeitstempel der gesendeten CAN-Nachricht. Die Zeit wird in Einheiten von 25ns gezählt.

Der Nachrichten-Header steht in den Elementen **can_msg[1..2]** und ist zusammengesetzt wie folgt:

Bits	31 / 23 15 / 7	30 / 22 14 / 6	29 / 21 13 / 5	28 / 20 12 / 4	27 / 19 11 / 3	26 / 18 10 / 2	25 / 17 9 / 1	24 / 16 8 / 0
msg [1]								
31:24	–	–	SID11	EID (17:6)				
23:16	EID (12:5)							
15:8	EID (4:0)					SID (10:8)		
7:0	SID (7:0)							
msg [2]								
31:24	–							
23:16	–							
15:8	SEQ (6:0)							ESI
7:0	FDF	BRS	RTR	IDE	DLC (3:0)			

Mit **P2_CANFD_Get_SEQ** erhalten Sie den Parameter SEQ aus **msg[]**. Mit **P2_CANFD_Get_Header_Parts** erhalten Sie die übrigen Bestandteile des Nachrichten-Headers aus **msg[]**.

Siehe auch

[P2_CANFD_Enable_Transmit_Event_Fifo](#), [P2_CANFD_Write_TMO](#), [P2_CANFD_Get_SEQ](#), [P2_CANFD_Init_Controller](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.ad` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas`
im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Write_TMO übergibt eine CAN-Nachricht an den Controller.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Write_TMO(fifo_no, id, seq, fdf,
    brs, rtr, ide, dlc, can_msg[], canfd_data[])
```

Parameter

fifo_no	Nummer (0...31) des Fifo. Nummer 0 steht für den Ausgabepuffer.	LONG
id	Identifizier der CAN-Nachricht (0...2 ¹¹ -1, 0...2 ¹² -1 oder 0...2 ²⁹ -1). Für 12 Bit-Identifizier siehe P2_CANFD_Set_SID11 .	LONG
seq	Kennwert (0...63) zur Nachverfolgung der Nachricht im Überwachungs-Fifo.	LONG
fdf	CAN-Format (nur CAN FD): 0: Format CAN (high speed). 1: Format CAN FD.	LONG
brs	Einstellung der Baudrate (nur CAN FD): 0: Nominale Baudrate. 1: Wechsel zwischen nominaler und Daten-Baudrate.	LONG
rtr	Remote transmission request (nur CAN 2.0). 0: RTR wird nicht verwendet. 1: Nachricht wird von extern angefordert (RTR).	LONG
ide	Merker für die Länge des Identifiziers: 0: 11 Bit Identifizier. 1: 29 Bit Identifizier.	LONG
dlc	Länge der CAN-Nachricht (data length code): 0...8: 0...8 Bytes 9: 12 Bytes 10: 16 Bytes 11: 20 Bytes 12: 24 Bytes 13: 32 Bytes 14: 48 Bytes 15: 64 Bytes	LONG
can_msg[]	Feld mit einem CAN-Nachrichtenobjekt. Das Feld muss mindestens 20 Elemente haben.	ARRAY LONG
canfd_data[]	Feld mit CAN-Informationen.	ARRAY LONG
ret_val	Status der Befehlsausführung: 0: OK, Nachricht erfolgreich an Controller übergeben. -1: Fifo ist voll, Nachricht wurde nicht übergeben.	LONG

Bemerkungen

Die CAN-Nachricht (ohne Header) muss im Feld **can_msg[]** ab Element 3 abgelegt werden. Der Header in den beiden ersten Elementen wird aus den Übergabeparametern erzeugt.

Aus einem Ausgangs-Fifo (**fifo_no** = 1...31) werden CANFD-Nachrichten in der Reihenfolge gesendet, wie sie eingestellt wurden. Aus dem Ausgabepuffer (**fifo_no** = 0) werden CANFD-Nachrichten in der Reihenfolge ihrer Priorität gesendet, d.h. die Nachricht mit der jeweils kleinsten ID wird zuerst gesendet.

Siehe auch

[P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#),
[P2_CANFD_Init_Controller](#), [P2_CANFD_Read_EFO](#), [P2_CANFD_Get_SEQ](#),
[P2_CANFD_Read_RMO](#)

P2_CANFD_Write_TMO

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.adl` und zugehöriges
Programm `Pro2_CANFD_sample_Code.bas`
im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Transmit_Msg gibt das Senden von gespeicherten CAN-Nachrichten aus einem Fifo frei.

Syntax

```
#Include ADwinPro_All.inc

P2_CANFD_Transmit_Msg(module, channel, fifo_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des CAN-Kanals.	LONG
fifo_no	Nummer (0...31) des Fifo. Nummer 0 steht für den Ausgabepuffer.	LONG

Bemerkungen

Um mehrere Fifos freizugeben, verwenden Sie **P2_CANFD_Transmit_Multi_Msg**.

Der Controller sendet Nachrichten aus dem Fifo, sobald er Zugriffsrecht auf den CAN-Bus hat. Die Freigabe zum Senden bleibt bestehen, solange Nachrichten im Fifo vorhanden sind.

Der Controller sendet jeweils die CAN-Nachricht als nächste, die – über alle freigegebenen Ausgangs-Fifos gesehen – den niedrigsten Identifier hat.

Wenn keine Nachrichten mehr im Fifo enthalten sind, wird die Freigabe zum Senden zurückgesetzt. Zum erneuten Senden muss die Freigabe für den Fifo anschließend wieder neu erteilt werden.

Siehe auch

[P2_CANFD_Transmit_Multi_Msg](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Write_TMO](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.ad` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas` im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Transmit_Msg

P2_CANFD_Transmit_Multi_Msg

P2_CANFD_Transmit_Multi_Msg gibt das Senden von gespeicherten CAN-Nachrichten aus mehreren Fifos frei.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_CANFD_Transmit_Multi_Msg(module, channel,  
                             fifo_pattern)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>channel</code>	Nummer (1, 2) des CAN-Kanals.	LONG
<code>fifo_no</code>	Bitmuster, in dem die gewünschten Fifos gesetzt werden. Bit 0 steht für den Ausgabepuffer, Bits 1...31 für die Fifos 1...31. Bit = 1: Fifo wird zum Senden freigegeben. Bit = 0: Fifo-Sendestatus bleibt unverändert.	LONG

Bemerkungen

Um einen einzelnen Fifo freizugeben, verwenden Sie **P2_CANFD_Transmit_Msg**.

Der Controller sendet Nachrichten, sobald er Zugriffsrecht auf den CAN-Bus hat. Die Freigabe zum Senden wird für jeden Fifo separat verwaltet und bleibt bestehen, solange Nachrichten im Fifo vorhanden sind.

Der Controller sendet jeweils die CAN-Nachricht als nächste, die – über alle Ausgangs-Fifos gesehen – den niedrigsten Identifier hat.

Wenn keine Nachrichten mehr in einem Fifo enthalten sind, wird die Freigabe zum Senden zurückgesetzt. Zum erneuten Senden muss die Freigabe für den Fifo anschließend neu erteilt werden.

Siehe auch

[P2_CANFD_Transmit_Msg](#), [P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Enable_Transmit_Queue](#), [P2_CANFD_Write_TMO](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

Sie finden die Registernummern des CAN-Controllers AN82527 im Intel®-Datenblatt (Address map). Beispiele sind:

- Adresse **00h**: Kontroll-Register
- Adresse **01h**: Status-Register
- Adresse **5fh**: Interrupt-Register

Sie finden die Registernummern des CAN-Controllers AN82527 im Intel®-Datenblatt.

P2_CANFD_Get_TREC gibt den Inhalt des Registers TREC auf einem CAN-Controller zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Get_TREC(canfd_data[])
```

Parameter

canfd_data[]	Feld mit CAN-Informationen.	ARRAY
		LONG
ret_val	Inhalt des Registers TREC.	LONG

Bemerkungen

Das Register TREC (transmit / receive error count, siehe Tabelle) enthält Informationen über den Fehlerstatus des Controllers und über aufgetretene Fehler.

Byte	31 / 23 15 / 7	30 / 22 14 / 6	29 / 21 13 / 5	28 / 20 12 / 4	27 / 19 11 / 3	26 / 18 10 / 2	25 / 17 9 / 1	24 / 16 8 / 0
31:24	–	–	–	–	–	–	–	–
23:16	–	–	TXBO	TXBP	RXBP	TX WARN	RX WARN	E WARN
15:8	TEC							
7:0	REC							

Die Informationen sind:

- TXBO (Bit 21): Sender hat den Status „Bus off“ (TEC > 255).
- TXBP (Bit 20): Sender hat den Status „Error passive“ (TEC > 127).
- RXBP (Bit 19): Empfänger hat den Status „Error passive“ (TEC > 127).
- TXWARN (Bit 18): Sender hat den Status „Error warning“ (128 > TEC > 95).
- RXWARN (Bit 17): Empfänger hat den Status „Error warning“ (128 > REC > 95).
- EWARN (Bit 16): Sender oder Empfänger hat den Status „Error warning“.
- TEC (Bits 8...15): Fehleranzahl des Senders.
- REC (Bits 0...7): Fehleranzahl des Empfängers.

Siehe auch

[P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Read_EFO](#)

Gültig für

CAN-FD-2 Rev. E

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.ad` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas` im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Get_TREC

P2_CANFD_Get_BDIAG0

P2_CANFD_Get_BDIAG0 gibt den Inhalt des Registers BDIAG0 auf einem CAN-Controller zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Get_BDIAG0(canfd_data[])
```

Parameter

canfd_data[]	Feld mit CAN-Informationen.	ARRAY LONG
ret_val	Inhalt des CAN-Controller-Registers	LONG

Bemerkungen

Das Register BDIAG0 (bus diagnostic register 0, siehe Tabelle) enthält Informationen über Fehlerzähler des Controllers bei der Baudrate.

Byte	31 / 23 15 / 7	30 / 22 14 / 6	29 / 21 13 / 5	28 / 20 12 / 4	27 / 19 11 / 3	26 / 18 10 / 2	25 / 17 9 / 1	24 / 16 8 / 0
31:24	DTERRCNT							
23:16	DRERRCNT							
15:8	NTERRCNT							
7:0	NRERRCNT							

Die Informationen sind:

- DTERRCNT (Bits 31...24): Fehleranzahl beim Senden mit Daten-Baudrate.
- DRERRCNT (Bits 23...16): Fehleranzahl beim Empfangen mit Daten-Baudrate.
- NTERRCNT (Bits 15...8): Fehleranzahl beim Senden mit nominaler Baudrate.
- NRERRCNT (Bits 7...0): Fehleranzahl beim Empfangen mit nominaler Baudrate.

Siehe auch

[P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Read_EFO](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.adl` und zugehöriges Programm `Pro2_CANFD_sample_Code.bas` im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

P2_CANFD_Get_BDIAG1 gibt den Inhalt des Registers BDIAG0 auf einem CAN-Controller zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_CANFD_Get_BDIAG1(canfd_data[])
```

Parameter

canfd_data[]	Feld mit CAN-Informationen.	ARRAY
		LONG
ret_val	Inhalt des CAN-Controller-Registers	LONG

Bemerkungen

Das Register BDIAG1 (bus diagnostic register 1, siehe Tabelle) enthält Informationen über verschiedene Fehlermeldungen des Controllers:

Byte	31 / 23 15 / 7	30 / 22 14 / 6	29 / 21 13 / 5	28 / 20 12 / 4	27 / 19 11 / 3	26 / 18 10 / 2	25 / 17 9 / 1	24 / 16 8 / 0
31:24	DLC MM	ESI	DCRC ERR	DSTUF ERR	DFORM ERR	–	DBIT1 ERR	DBIT0 ERR
23:16	TXBO ERR	–	NCRC ERR	NSTUF ERR	NFORM ERR	NACK ERR	NBIT1 ERR	NBIT0 ERR
15:8	EFMSGCNT (15:8)							
7:0	EFMSGCNT (7:0)							

Die Informationen sind:

- **DLCMM (Bit 31):** DLC mismatch, die angegebene Länge der CAN-Nachricht ist größer als die reservierte Anzahl der CANFD-Bytes im Speicher (payload size).
- **ESI (Bit 30):** Beim Empfang einer Nachricht war der Parameter ESI gesetzt.
- **DCRCERR (Bit 29):** Die CRC-Prüfsumme einer empfangenen Nachricht war falsch (bei Daten-Baudrate).
- **DSTUFERR (Bit 28):** Es wurden mehr als 5 gleiche Bits in Folge empfangen, wo es nicht erlaubt ist (bei Daten-Baudrate).
- **DFORMERR (Bit 27):** Ein Teil der empfangenen Nachricht hat das falsche Format (bei Daten-Baudrate).
- **DBIT1ERR (Bit 25):** Es sollte eine logische 1 gesendet werden, aber auf dem Bus wurde eine logische 0 beobachtet (bei Daten-Baudrate).
- **DBIT0ERR (Bit 24):** Es sollte eine logische 0 gesendet werden, aber auf dem Bus wurde eine logische 1 beobachtet (bei Daten-Baudrate).
- **TXBOERR (Bit 23):** Controller hatte vorübergehend den Status „Bus off“.
- **NCRCERR (Bit 21):** Die CRC-Prüfsumme einer empfangenen Nachricht war falsch (bei nominaler Baudrate).
- **NSTUFERR (Bit 20):** Es wurden mehr als 5 gleiche Bits in Folge empfangen, wo es nicht erlaubt ist (bei nominaler Baudrate).
- **NFORMERR (Bit 19):** Ein Teil der empfangenen Nachricht hat das falsche Format (bei nominaler Baudrate).
- **NACKERR (Bit 18):** Gesendete Nachricht wurde nicht bestätigt (bei nominaler Baudrate).
- **NBIT1ERR (Bit 17):** Es sollte eine logische 1 gesendet werden, aber auf dem Bus wurde eine logische 0 beobachtet (bei nominaler Baudrate).
- **NBIT0ERR (Bit 16):** Es sollte eine logische 0 gesendet werden, aber auf dem Bus wurde eine logische 1 beobachtet (bei nominaler Baudrate).
- **EFMSGCNT (Bits 0...15):** Anzahl fehlerfreie Nachrichten.

Siehe auch

[P2_CANFD_Enable_Transmit_Fifo](#), [P2_CANFD_Init_Controller](#), [P2_CANFD_Read_EFO](#)

Gültig für

[CAN-FD-2 Rev. E](#)

Beispiel

siehe Beispieloberfläche `ADwinCANFDUserInterface.adl` und zugehöriges
Programm `Pro2_CANFD_sample_Code.bas`
im Ordner `C:\ADwin\ADbasic\samples_ADwin_ProII\CAN FD`.

3.10 Pro II: LIN-Bus-Schnittstelle

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit LIN-Bus-Schnittstellen gelten:

- [P2_LIN_Init](#) (Seite 270)
- [P2_LIN_Init_Write](#) (Seite 272)
- [P2_LIN_Init_Apply](#) (Seite 273)
- [P2_LIN_Reset](#) (Seite 274)
- [P2_LIN_Get_Version](#) (Seite 275)
- [P2_LIN_Read_Dat](#) (Seite 276)
- [P2_LIN_Ch_Read_Cnt](#) (Seite 278)
- [P2_LIN_Msg_Read_Status](#) (Seite 279)
- [P2_LIN_Msg_Write](#) (Seite 280)
- [P2_LIN_Msg_Transmit](#) (Seite 281)
- [P2_LIN_Set_LED](#) (Seite 282)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

P2_LIN_Init

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 <i>ADwin</i> 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.

Siehe auch

[P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#), [P2_LIN_Get_Version](#),
[P2_LIN_Read_Dat](#), [P2_LIN_Ch_Read_Cnt](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Gültig für

LIN-2 Rev. E

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, i 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)
    P2_LIN_Init_Apply(lin_datatable)

    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)
    state = 1

Event:
    SelectCase state
    Case 1 'msg transmit
        For i = 1 To 8
            Data_1[i] = i
        Next i
        Rem set 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

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“.

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#), [P2_LIN_Get_Version](#), [P2_LIN_Read_Dat](#), [P2_LIN_Ch_Read_Cnt](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Gültig für

[LIN-2 Rev. E](#)

Beispiel

siehe [P2_LIN_Init](#)

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_datatable[] Feld, das Einstellungen für die Datenübertragung zwischen *ADwin* CPU und LIN-Modul enthält. **ARRAY** **LONG** |

Bemerkungen

P2_LIN_Init_Apply verändert die Einstellungsdaten der LIN-Schnittstellen nicht.

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Reset](#), [P2_LIN_Get_Version](#), [P2_LIN_Read_Dat](#), [P2_LIN_Ch_Read_Cnt](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Gültig für

[LIN-2 Rev. E](#)

Beispiel

siehe [P2_LIN_Init](#)

P2_LIN_Init_Apply

P2_LIN_Reset

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

<code>lin_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und LIN-Modul enthält.	ARRAY LONG
<code>reset_mode</code>	Einstellungen, die Kanäle zurückgesetzt werden: 1: alle Einstellungen auf Einschaltzustand. 2: nur LIN-Zähler auf 0 rücksetzen	LONG

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.

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Get_Version](#), [P2_LIN_Read_Dat](#), [P2_LIN_Ch_Read_Cnt](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Gültig für

[LIN-2 Rev. E](#)

Beispiel

- / -

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_datatable[]	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und LIN-Modul enthält.	ARRAY 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.

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#), [P2_LIN_Read_Dat](#), [P2_LIN_Ch_Read_Cnt](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Gültig für

[LIN-2 Rev. E](#)

Beispiel

- / -

P2_LIN_Get_Version

P2_LIN_Read_Dat

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.

Alternativ kann die Anzahl übertragener Nachrichten (**rd_dat[4]**) mit dem Befehl **P2_LIN_Ch_Read_Cnt** gelesen werden und der Status (**rd_dat[20]**) mit dem Befehl **P2_LIN_Msg_Read_Status**.

Eine Messagebox / der Schnittstellenstatus besteht aus 20 Elementen. Diese Daten werden in den Feldelementen **rd_dat[1] ... rd_dat[20]** gespeichert. Die folgenden Tabellen zeigen die Bedeutung für eine Messagebox und für den Schnittstellenstatus:

Element	Messagebox, membox =1...64
1	Schnittstellennummer (1...4)
2	Nummer (1...64) der Messagebox
3	Identifizier (0...63) der Messagebox
4...11	Datenbytes 1...8
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).
18	Pausenzeit in µs zwischen 2 Datenbytes. Standard: 0.
19	Anzahl aufgetretener Timeout-Fehler für diese Nachricht.
20	1: neue Nachricht wurde gesendet / empfangen. -1: keine neue Nachricht -2: Nachricht wird gerade empfangen. -3: Nachricht mit Timeout-Fehler -4: Checksummenfehler beim Empfang.

Element	Schnittstellenstatus, <code>membox=65</code>
1	Schnittstellennr. (1...4)
2	Nummer (65) der Messagebox
3	Zuletzt übertragene Kennzahl (0...63)
4	Anzahl der übertragenen Nachrichten seit dem Start des Moduls
5	Ab Rev. E04: Master erkennt einen Wakeup Request: 0: kein Request 1: Request erkannt

Ein erkannter Wake Request (ab Rev. E04) wird auf 0 zurückgesetzt, sobald der Master am Busverkehr teilnimmt.

Siehe auch

[P2_LIN_Init](#), [P2_LIN_Init_Write](#), [P2_LIN_Init_Apply](#), [P2_LIN_Reset](#), [P2_LIN_Get_Version](#), [P2_LIN_Ch_Read_Cnt](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Ch_Read_Cnt](#)

Gültig für

[LIN-2 Rev. E](#)

Beispiel

siehe [P2_LIN_Init](#)

P2_LIN_Ch_Read_Cnt

P2_LIN_Ch_Read_Cnt gibt die Anzahl der übertragenen Nachrichten einer LIN-Schnittstelle zurück.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_LIN_Ch_Read_Cnt(lin_datatable[],  
                             channel)
```

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
ret_val	Anzahl der übertragenen Nachrichten seit dem Start des Moduls.	LONG

Bemerkungen

Der Anzahl der Nachrichten (und andere Informationen) kann auch mit dem Befehl **P2_LIN_Read_Dat** gelesen werden.

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_Read_Status](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Gültig für

[LIN-2 Rev. E](#)

Beispiel

- / -

P2_LIN_Msg_Read_Status gibt den Status einer Messagebox einer LIN-Schnittstelle zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_LIN_Msg_Read_Status (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	Kennzahl zur Auswahl einer Messagebox oder der Schnittstellenstatus: 1...64: Nummer der LIN-Messagebox, deren Daten gelesen werden. 65: Schnittstellenstatus lesen.	LONG
ret_val	Status der LIN-Messagebox. 1: neue Nachricht wurde gesendet / empfangen. -1: keine neue Nachricht -2: Nachricht wird gerade empfangen. -3: Nachricht mit Timeout-Fehler -4: Checksummenfehler beim Empfang.	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.

Der Status (und andere Informationen) kann auch mit dem Befehl **P2_LIN_Read_Dat** gelesen werden.

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_Ch_Read_Cnt](#), [P2_LIN_Msg_Write](#), [P2_LIN_Msg_Transmit](#)

Gültig für

LIN-2 Rev. E

Beispiel

- / -

P2_LIN_Msg_Read_Status

P2_LIN_Msg_Write

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.

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_Ch_Read_Cnt](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Msg_Transmit](#)

Gültig für

LIN-2 Rev. E

Beispiel

siehe [P2_LIN_Init](#)

P2_LIN_Msg_Transmit sendet im Betriebsmodus LIN Master einen Header und den Identifier einer Messagebox auf den LIN-Bus.

Ab Rev. E04 kann im Betriebsmodus LIN Slave ein Wakeup-Signal ausgelöst werden.

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. Ab Rev. E04: Nummer 65 löst ein Wakeup-Signal aus.	LONG

Bemerkungen

P2_LIN_Msg_Transmit wird in der Regel nur im Betriebsmodus LIN-Master verwendet (siehe **P2_LIN_Init_Write**), weil nur der LIN-Master einen Header senden kann.

Durch das Senden eines Headers auf dem LIN-Bus reagieren diejenigen Teilnehmer am LIN-Bus, die eine Messagebox mit dem Identifier **msg_id** verwalten, indem sie ein Datenpaket senden oder ein auf dem Bus anstehendes Datenpaket empfangen. Der Identifier **msg_id** der Messagebox **membox** wird mit **P2_LIN_Msg_Write** festgelegt.

Ab Rev. E04: Das Wakeup-Signal setzt den Bus für 290µs auf den dominanten Pegel, anschließend wird wieder der Ruhepegel gesetzt. Das Wakeup-Signal kann nur im Betriebsmodus LIN Slave ausgelöst werden.

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_Ch_Read_Cnt](#), [P2_LIN_Msg_Read_Status](#), [P2_LIN_Msg_Write](#)

Gültig für

LIN-2 Rev. E

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

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>channel</code>	Nummer (1...2) der LIN-Schnittstelle.	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

[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
```

3.11 Pro II: PWM-Ausgänge

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit PWM-Ausgängen gelten:

- [P2_PWM_Enable](#) (Seite 284)
- [P2_PWM_Get_Status](#) (Seite 285)
- [P2_PWM_Init](#) (Seite 286)
- [P2_PWM_Latch](#) (Seite 288)
- [P2_PWM_Reset](#) (Seite 289)
- [P2_PWM_Standby_Value](#) (Seite 290)
- [P2_PWM_Write_Latch](#) (Seite 291)
- [P2_PWM_Write_Latch_Block](#) (Seite 292)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

P2_PWM_Enable

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

Bitnr.	31:16	15	14	...	1	0
PWM-Ausgang	–	16	15	...	2	1

Bemerkungen

Vor dem Freigeben eines PWM-Ausgangs müssen Sie – insbesondere nach dem Einschalten der Hardware – mit **P2_PWM_Init** die Voreinstellungen sowie mit **P2_PWM_Write_Latch** und **P2_PWM_Latch** Frequenz und Tastverhältnis für die Ausgabe festlegen.

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](#), [P2_PWM_Write_Latch_Block](#)

Gültig für

PWM-16(-I) Rev. E

Beispiel

siehe [P2_PWM_Init](#) (Seite 286)

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

Bitnr.	31:16	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_Get_Status

P2_PWM_Init

P2_PWM_Init setzt die Voreinstellungen für den angegebenen PWM-Ausgang.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_PWM_Init(module, pwm_output, start_delay,  
            start_value, mode, count)
```

Begriffe:

Periode

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pwm_output	Nummer des PWM-Ausgabekanals (1...16).	LONG
start_delay	Einmalige Startverzögerung in Einheiten von 10ns.	LONG
start_value	Startpegel für die PWM-Ausgabe: 0: TTL-Pegel low. 1: TTL-Pegel high.	LONG
mode	Betriebsmodus des PWM-Ausgangs als Bitmuster; nur die Bits 2:0 sind relevant, andere Bits werden ignoriert. Bit 0: Übernahme einer neuen PW-Frequenz: <ul style="list-style-type: none"> Bit =0: Übernahme bei Periodenende Bit=1: Übernahme sofort. Bit 1: Anzahl der Perioden: <ul style="list-style-type: none"> Bit =0: unendlich viele Perioden. Bit=1: Anzahl der Perioden ist count. Bit 2: Anhalten bei Stopp-Befehl: <ul style="list-style-type: none"> Bit =0: Anhalten bei Periodenende Bit=1: Anhalten sofort. 	LONG
count	Anzahl der Perioden (1...32768), die pro Ausgabezyklus ausgeführt werden. Wert ist nur relevant, wenn mode , Bit 1=1.	LONG

Bemerkungen

Die Voreinstellungen werden aktiv, sobald PWM-Ausgänge mit **P2_PWM_Enable** zur Ausgabe freigegeben werden.

Die Änderung der Voreinstellungen bei laufender Ausgabe ist nicht möglich. Stattdessen stoppen Sie die PWM-Ausgänge mit **P2_PWM_Reset** oder sperren sie mit **P2_PWM_Enable**, um die Voreinstellungen zu ändern. Anschließend geben Sie die PWM-Ausgänge wieder zur Ausgabe frei.

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

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,0,0,0,1)
    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

P2_PWM_Latch schreibt Frequenz und Tastverhältnis eines oder mehrerer PWM-Ausgänge in das Register 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

Bitnr.	31:16	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](#).

Das Schreiben ins Ausgangsregister kann synchron mit Aktionen auf anderen Modulen gestartet werden. Verwenden Sie hierzu den Befehl **P2_Sync_All**.

Siehe auch

[P2_PWM_Enable](#), [P2_PWM_Get_Status](#), [P2_PWM_Init](#), [P2_PWM_Reset](#), [P2_PWM_Standby_Value](#), [P2_PWM_Write_Latch](#), [P2_PWM_Write_Latch_Block](#), [P2_Sync_All](#)

Gültig für

[PWM-16\(-I\) Rev. E](#)

Beispiel

siehe [P2_PWM_Init](#) (Seite 286)

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

Bitnr.	31:16	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 286)

P2_PWM_Reset

P2_PWM_Standby_Value

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

Bitnr.	31:16	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_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

module	Eingestellte Moduladresse (1...15).	LONG
pwm_output	Nummer des PWM-Ausgabekanals (1...16).	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 ...50MHz.	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** (oder **P2_Sync_All**) werden die Werte für die PWM-Ausgabe aktiviert.

Die höchste Ausgangsfrequenz, bei der das Tastverhältnis noch in 1%-Schritten einstellbar ist, beträgt etwa 1000kHz.

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 286)

P2_PWM_Write_Latch

P2_PWM_Write_Latch_Block

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 ...50MHz.	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 1000kHz.

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

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)
```




3.12 Pro II: Temperaturmess-Module

Dieser Abschnitt beschreibt Befehle, die für Pro II-Module zur Temperaturmessung gelten:

- [P2_RTD_Channel_Config](#) (Seite 295)
- [P2_RTD_Config](#) (Seite 297)
- [P2_RTD_Convert](#) (Seite 298)
- [P2_RTD_Read](#) (Seite 299)
- [P2_RTD_Read8](#) (Seite 300)
- [P2_RTD_Start](#) (Seite 301)
- [P2_RTD_Status](#) (Seite 303)
- [P2_TC_Latch](#) (Seite 304)
- [P2_TC_Read_Latch](#) (Seite 305)
- [P2_TC_Read_Latch4](#) (Seite 307)
- [P2_TC_Read_Latch8](#) (Seite 309)
- [P2_TC_Set_Rate](#) (Seite 311)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

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...3) des Temperaturfühlers: 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 je 2 Messungen für einen gefilterten Messwert (nicht das Intervall 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. Wir empfehlen die 4-Leiter-Messung, weil sie die genaueste Messmethode ist.

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:

$$\text{sample_period} = \frac{10^6}{\text{Frequenz} \cdot 2^{\text{filter}}}$$

P2_RTD_Channel_Config

periodische Störfrequenz

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 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:

$$T_{\text{Gesamt}} = \sum_{i=1}^{\text{Anz. Kanäle}} (T_{\text{Kanal}} + \text{Einschwingzeit})$$

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_Config

P2_RTD_Convert

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 Temperaturfühlers: 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_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

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1...8) des Messkanals.	LONG
ret_val	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_Read

P2_RTD_Read8

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

<code>module</code>	Eingestellte Moduladresse (1...15).	<code>__LONG</code>
<code>array[]</code>	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY
		<code>LONG</code>
<code>index</code>	Indes des Feldelements, ab dem die Messwerte gespeichert werden.	<code>LONG</code>

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_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:
Bit = 0: Modul ignorieren.
Bit = 1: Temperaturmessung starten.

Bits in pattern	31:15	14	13	...	1	0
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

P2_RTD_Start

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: 4 wire, PT100, 50 Hz filter
    P2_RTD_Channel_Config(module, i, 1, 1, 0, 7, 156)
  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 gibt den Status eines einfachen Temperatur-Messzyklus auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_RTD_Status(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
ret_val	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_RTD_Status

P2_TC_Latch

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).	<code>_LONG</code>
---------------	-------------------------------------	--------------------

Bemerkungen

Die Werte in den Latches werden erst beim Auslesen mit **...Read_Latch** in den gewünschten Rückgabewert (°C, °F, Thermospannung) umgerechnet.

Das Kopieren der Werte in Latches kann synchron mit Aktionen auf anderen Modulen gestartet werden. Verwenden Sie hierzu den Befehl **P2_Sync_All**.

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_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 $-80000\mu\text{V}$... $80000\mu\text{V}$.

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
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

P2_TC_Read_Latch

Typ	Temperatur- bereich [°C]	Temperatur- bereich [°F]	Thermo- spannung [μV]
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 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
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

P2_TC_Read_Latch4

Typ	Temperatur- bereich [°C]	Temperatur- bereich [°F]	Thermo- spannung [μV]
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 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 μV ...80000 μV .

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
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

P2_TC_Read_Latch8

Typ	Temperatur- bereich [°C]	Temperatur- bereich [°F]	Thermo- spannung [μV]
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 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)
```

P2_TC_Set_Rate

3.13 Pro II: Dehnungsmessstreifen-Module

Dieser Abschnitt beschreibt Befehle, die für Pro II-Module zur Messung von Dehnungsmessstreifen gelten:

- [P2_SG_Mode](#) (Seite 313)
- [P2_SG_Start](#) (Seite 315)
- [P2_SG_Wait](#) (Seite 316)
- [P2_SG_Read](#) (Seite 317)
- [P2_SG_Convert](#) (Seite 318)
- [P2_SG_Init](#) (Seite 319)
- [P2_SG_Zero](#) (Seite 321)
- [P2_SG_Set_Gain](#) (Seite 322)

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

P2_SG_Mode stellt den Betriebsmodus für Dehnmessstreifen auf dem angegebenen Modul ein und wählt die zu messenden Kanäle aus.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SG_Mode(module, mode, channels)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
mode	Modus: 0: Abgleich der DMS-Parameter. 1: Modus „single shot“, einfacher Messzyklus. 3: Modus „continuous max“, Messzyklus mit maximaler Geschwindigkeit.	LONG
channels	Bitmuster (nur für Modi 1 und 3), das die zu wandelnden Kanäle (die Messgruppe) bestimmt. Bit = 0: Kanal nicht wandeln. Bit = 1: Kanal wandeln.	LONG

Bitnr.	Kanal
0	Brückenspannung B an DMS1
1	Brückenspannung B an DMS2
2	Brückenspannung B an DMS3
3	Brückenspannung B an DMS4
4	Sense-Leitung SX an DMS1
5	Sense-Leitung SX an DMS2
6	Sense-Leitung SX an DMS3
7	Sense-Leitung SX an DMS4
8	Versorgungsspannung EX an DMS1
9	Versorgungsspannung EX an DMS2
10	Versorgungsspannung EX an DMS3
11	Versorgungsspannung EX an DMS4
12	Analogeingang AIN 13
13	Analogeingang AIN 14
31:10	reserviert

Bemerkungen

Nach dem Einschalten ist der Modus 0 aktiv.

Die Modi 1 und 3 aktivieren die Ablaufsteuerung des Moduls. 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 Abgleich:	Nullpunktabgleich mit P2_SG_Zero oder Abgleich des Verstärkungsfaktors mit P2_SG_Set_Gain .
1 single shot:	Die Ablaufsteuerung wird mit P2_SG_Start gestartet; die Ablaufsteuerung endet, sobald die gewählten Kanäle je einmal gewandelt sind. Das Ende der Ablaufsteuerung wird mit P2_SG_Wait abgefragt und die Messwerte mit P2_SG_Read eingelesen.

P2_SG_Mode

Modus	Art der Messung
3 continuous max:	<p>Die Ablaufsteuerung wandelt ununterbrochen neue Messwerte an den gewählten Kanälen, d.h. Wandlung und Prozesszyklus laufen asynchron.</p> <p>Die Wandlung wird mit P2_SG_Start gestartet. Im Prozesszyklus wird mit P2_SG_Read der jeweils neueste Messwert gelesen.</p>

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.

Bei der Vierleiterschaltung wird die Sense-Leitung SX nicht angeschlossen.

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

Siehe auch

[P2_SG_Start](#), [P2_SG_Wait](#), [P2_SG_Read](#), [P2_SG_Convert](#), [P2_SG_Init](#), [P2_SG_Zero](#), [P2_SG_Set_Gain](#)

Gültig für

[SG-4/18 Rev. E](#)

Beispiel

siehe [P2_SG_Init](#)

P2_SG_Start startet die Ablaufsteuerung auf allen angegebenen Modulen gleichzeitig.

Syntax

```
#Include ADwinPro_All.inc

P2_SG_Start(module_pattern)
```

Parameter

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

Bitmuster	31:15	14	13	...	1	0
Moduladresse	–	15	14	...	2	1

Bemerkungen

- / -

Siehe auch

[P2_SG_Mode](#), [P2_SG_Wait](#), [P2_SG_Read](#), [P2_SG_Convert](#), [P2_SG_Init](#),
[P2_SG_Zero](#), [P2_SG_Set_Gain](#)

Gültig für

SG-4/18 Rev. E

Beispiel

siehe [P2_SG_Init](#)

P2_SG_Start

P2_SG_Wait

P2_SG_Wait wartet, bis die Ablaufsteuerung auf dem angegebenen Modul alle Kanäle der Messgruppe gewandelt und gespeichert hat.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SG_Wait(module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	<code>__LONG</code>
---------------------	-------------------------------------	---------------------

Bemerkungen

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

Siehe auch

[P2_SG_Mode](#), [P2_SG_Start](#), [P2_SG_Read](#), [P2_SG_Convert](#), [P2_SG_Init](#),
[P2_SG_Zero](#), [P2_SG_Set_Gain](#)

Gültig für

[SG-4/18 Rev. E](#)

Beispiel

siehe [P2_SG_Init](#)

P2_SG_Read 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_SG_Read(module, count, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
count	Anzahl (1...10) der zu lesenden Messwerte.	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden.	ARRAY
		LONG
		FLOAT
index	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1...n).	LONG

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **P2_SG_Init** die Ablaufsteuerung des Moduls für eine Messgruppe aktiviert wurde.

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

Es wird ein 24 Bit-Wert zurückgegeben. Der Messwert des 18 Bit-ADC steht in den Bits 23:6 des Rückgabewerts; der Messwert ist also um 6 Bits nach links verschoben und die Bits 5:0 sind Null.

Bitnr.	31:24	23:6	5:0
Inhalt	0	18-Bit Messwert	0

Siehe auch

[P2_SG_Mode](#), [P2_SG_Start](#), [P2_SG_Wait](#), [P2_SG_Convert](#), [P2_SG_Init](#), [P2_SG_Zero](#), [P2_SG_Set_Gain](#)

Gültig für

SG-4/18 Rev. E

Beispiel

siehe [P2_SG_Init](#)

P2_SG_Read

P2_SG_Convert

P2_SG_Convert berechnet aus dem Digitalwert eines DMS die zugehörige Spannung.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SG_Convert(value, gain)
```

Parameter

value	Digitalwert eines DMS (24 Bit).	LONG
gain	Verstärkungsfaktor (0...6): 0: 1 1: 20 2: 40 3: 80 4: 160 5: 320 6: 640	LONG
ret_val	Spannung in Millivolt.	FLOAT

Bemerkungen

Verwenden Sie für die Berechnung der Brückenspannung den gleichen Verstärkungsfaktor, den Sie für den DMS-Kanal mit **P2_SG_Init** eingestellt haben. Für Analogeingänge, Sense-Leitung und Versorgungsspannung ist der Verstärkungsfaktor immer 1 (**gain**=0).

Siehe auch

[P2_SG_Mode](#), [P2_SG_Start](#), [P2_SG_Wait](#), [P2_SG_Read](#), [P2_SG_Init](#), [P2_SG_Zero](#), [P2_SG_Set_Gain](#)

Gültig für

[SG-4/18 Rev. E](#)

Beispiel

siehe [P2_SG_Init](#)

P2_SG_Init stellt Verstärkung und Filterfrequenz für die Brückenspannung sowie die Versorgungsspannung eines DMS ein.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SG_Init(module, channel, gain, filter, excitation)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1...4) des DMS-Messkanals.	LONG
gain	Verstärkungsfaktor (1...6) der Brückenspannung: 1: 20 2: 40 3: 80 4: 160 5: 320 6: 640	LONG
filter	Frequenz des Eingangsfilters: 0: 2kHz 1: 200Hz	LONG
excitation	Versorgungsspannung EX (0...7): 0: 1,25V 1: 2,50V 2: 3,75V 3: 5,00V 4: 6,25V 5: 7,50V 6: 8,75V 7: 9,80V	LONG

Bemerkungen

Der Befehl **P2_SG_Init** soll nur mit niedriger Priorität ausgeführt werden (z. B. im Abschnitt **LowInit**).

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **P2_SG_Init** die Ablaufsteuerung des Moduls (Modus 1 / 3) aktiviert wurde.

Der Verstärkungsfaktor legt Messbereich und Signalaufösung der Brückenspannung fest:

gain	Verstärkung	Messbereich	Auflösung
1	20	±500mV	3,84 µV
2	40	±250mV	1,92 µV
3	80	±125mV	0,96 µV
4	160	±62,5mV	0,48 µV
5	320	±31,25mV	0,24 µV
6	640	±15,625mV	0,12 µV

Siehe auch

[P2_SG_Mode](#), [P2_SG_Start](#), [P2_SG_Wait](#), [P2_SG_Read](#), [P2_SG_Convert](#), [P2_SG_Zero](#), [P2_SG_Set_Gain](#)

Gültig für

SG-4/18 Rev. E

P2_SG_Init

Beispiel

```
#Include ADwinPro_All.inc
#Define module 2
#Define gain_B 2 '1..6: 20,40,80,160,320,640
#Define ex_B 4 'excitation voltage 6.25 V

Dim i As Long
Dim Data_1[10] As Long

LowInit:
  Rem set mode single shot and select all channels
  P2_SG_Mode(module,1,0FFH)
  Rem set gain, filter, and excitation voltage of all bridges
  For i = 1 To 4
    P2_SG_Init(module, i, gain_B, 1, ex_B)
  Next

  Rem start sampling
  P2_SG_Start(Shift_Left(1, module - 1))
  Processdelay = 40000

Event:
  Rem wait for end of sampling
  P2_SG_Wait(module)
  Rem read 8 values
  P2_SG_Read(module, 8, Data_1, 1)
  Rem start next sample
  P2_SG_Start(Shift_Left(1, module - 1))

  Rem get bridge voltage
  For i = 1 To 4
    FPar[i] = P2_SG_Convert(Data_1[i], gain_B) 'bridge [mV]
  Next
  Rem get sense voltage
  For i = 5 To 8
    FPar[i] = P2_SG_Convert(Data_1[i], 0)/1000 'excitation [V]
  Next

  Rem calculate relative strain
  For i = 1 To 4
    FPar[i+8] = FPar[i] / FPar[4+i] 'strain [mV/V]
  Next
```

P2_SG_Zero führt den Nullpunktabgleich für einen DMS-Kanal auf dem angegebenen Modul durch.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SG_Zero(module, channel, save)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1...4) des DMS-Messkanals.	LONG
save	Speicher-Modus: 0: Abgleich ist temporär, die vorherige Einstellung ist nach Neustart des Moduls wieder aktiv. 1: Abgleich ist dauerhaft, die vorherige Einstellung geht verloren.	LONG
ret_val	Ergebnis des Nullpunktabgleichs: 0: Abgleich war erfolgreich. -1: Befehl nur mit niedriger Priorität ausführen. -2: Falscher Modultyp. -3: Betriebsmodus (P2_SG_Mode) ist nicht 0.	LONG

Bemerkungen

Stellen Sie für den Nullpunktabgleich sicher, dass die DMS unbelastet ist. Der Abgleich korrigiert das Nullsignal der DMS-Schaltung um bis zu $\pm 10\text{mV}$.

Der Befehl **P2_SG_Set_Gain** ist nur sinnvoll für den Betriebsmodus 0 (Abgleich, siehe **P2_SG_Mode**).

Der Befehl darf nur mit niedriger Priorität ausgeführt werden (z. B. im Abschnitt **LowInit**).

Nach dem Einschalten des Moduls werden die im EEPROM gespeicherten Werte für den Nullpunktabgleich geladen und für Messungen verwendet. Mit **save=1** speichert **P2_SG_Set_Gain** die Werte für den Abgleich des Verstärkungsfaktors dauerhaft im EEPROM des Moduls.

Mit **P2_SG_Set_Gain** kann nach dem Nullpunkt auch der Verstärkungsfaktor abgeglichen werden.

Siehe auch

[P2_SG_Mode](#), [P2_SG_Set_Gain](#), [P2_SG_Start](#), [P2_SG_Wait](#), [P2_SG_Read](#), [P2_SG_Convert](#), [P2_SG_Init](#)

Gültig für

SG-4/18 Rev. E

Beispiel

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

LowInit:
    Rem set calibration mode
    P2_SG_Mode(module, 0, 0b)
    Rem calibrate channel 1 and save value
    Par_1 = P2_SG_Zero(module, 1, 1)
```

P2_SG_Zero

P2_SG_Set_Gain

P2_SG_Set_Gain führt den Abgleich des Verstärkungsfaktors für einen DMS-Kanal auf dem angegebenen Modul durch.

Der Rückgabewert gibt an, ob der Abgleich erfolgreich war.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SG_Set_Gain(module, channel, mode,
                        value, save)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1...4) des DMS-Messkanals.	LONG
mode	Abgleichmodus: 1: Abgleich für Sechseiter-Schaltung mit Sollwert value , Messung an der Sense-Leitung SX. 2: Abgleich für Vierleiter-Schaltung mit Sollwert value , Messung der Versorgungsspannung EX. 3: (bekannten) Korrekturwert value setzen ohne Abgleich.	LONG
value	Bedeutung und Datentyp je nach Abgleichmodus: mode =1/2: Sollwert in mV/V (Float) für den Abgleich. mode =3: Korrekturwert (Long , 0...231) für den Verstärkungsfaktor.	FLOAT LONG
save	Speicher-Status: 0: Abgleich ist temporär, die vorherige Einstellung ist nach Neustart des Moduls wieder aktiv. 1: Abgleich ist dauerhaft, die vorherige Einstellung geht verloren.	LONG
ret_val	Ergebnis des Abgleichs: >0: Abgleich war erfolgreich, der Rückgabewert ist der Korrekturwert für den Verstärkungsfaktor.	LONG

Bemerkungen

Führen Sie erst mit **P2_SG_Zero** einen Nullpunktabgleich durch. Für den Abgleich des Verstärkungsfaktors muss eine definierte Last auf die DMS aufgebracht werden. Der Abgleich korrigiert den Verstärkungsfaktor der DMS-Schaltung so, dass der Sollwert erreicht wird.

Der Befehl **P2_SG_Set_Gain** ist nur sinnvoll für den Betriebsmodus 0 (Abgleichmodus, siehe **P2_SG_Mode**).

Der Befehl darf nur mit niedriger Priorität ausgeführt werden (z.B. im Abschnitt **LowInit**).

Nach dem Einschalten des Moduls werden die im EEPROM gespeicherten Werte für den Abgleich des Verstärkungsfaktors geladen und für Messungen verwendet. Mit **save**=1 speichert **P2_SG_Set_Gain** die Werte für den Abgleich des Verstärkungsfaktors dauerhaft im EEPROM des Moduls.

Siehe auch

[P2_SG_Mode](#), [P2_SG_Start](#), [P2_SG_Wait](#), [P2_SG_Read](#), [P2_SG_Convert](#), [P2_SG_Init](#), [P2_SG_Zero](#)

Gültig für

SG-4/18 Rev. E

Beispiel

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

LowInit:
    Rem set calibration mode
    P2_SG_Mode(module, 0, 0b)
    Rem calibrate channel 5 with 4-wire-setting and 11.2 mv/V;
    Rem save value afterwards
    Par_1 = P2_SG_Set_Gain(module, 5, 11.2, 2, 1)
```

3.14 Pro II: RSxxx

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit RSxxx-Schnittstellen gelten:

- [P2_Check_Shift_Reg](#) (Seite 325)
- [P2_Get_RS](#) (Seite 326)
- [P2_Read_Fifo](#) (Seite 327)
- [P2_RS_Init](#) (Seite 328)
- [P2_RS_Reset](#) (Seite 330)
- [P2_RS485_Send](#) (Seite 331)
- [P2_RS_Set_LED](#) (Seite 332)
- [P2_Set_RS](#) (Seite 333)
- [P2_Write_Fifo](#) (Seite 334)
- [P2_Write_Fifo_Full](#) (Seite 335)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

P2_Check_Shift_Reg gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO der Schnittstelle (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	Nummer (1, 2 oder 1...4) der Schnittstelle, deren Sende-Status gelesen werden soll.	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](#), [P2_Write_Fifo](#), [P2_Write_Fifo_Full](#)

Gültig für

[RS422-4 Rev. E](#), [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_Check_Shift_Reg

P2_Get_RS

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](#), [P2_Write_Fifo](#), [P2_Write_Fifo_Full](#)

Gültig für

[RS422-4 Rev. E](#), [RSxxx-2 Rev. E](#), [RSxxx-4 Rev. E](#)

Beispiel

- / -

P2_Read_Fifo liest einen Wert aus dem Eingangs-FIFO einer bestimmten Schnittstelle 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 (1, 2 oder 1...4) der auszulesenden Schnittstelle.	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](#), [P2_Write_Fifo](#), [P2_Write_Fifo_Full](#)

Gültig für

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_RS_Reset(1)
Rem Initialisiere Schnittstelle 1 auf Modul 1 mit 9600 Baud,
Rem ohne Parität, 8 Datenbits, 1 Stoppbit und
Rem Hardwarehandshake (nur RS232).
P2_RS_Init(1, 1, 9600, 0, 8, 0, 1)
```

Event:

```
Rem Einen Wert aus dem FIFO holen. Wenn der FIFO leer ist,
Rem wird -1 zurückgeliefert.
Par_1 = P2_Read_Fifo(1, 1)
```

Siehe auch weitere [Beispiele für RS232 und RS485 \(Pro II\)](#) ab Seite 2084.

P2_Read_Fifo

P2_RS_Init

P2_RS_Init initialisiert eine bestimmte Schnittstelle 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	Nummer (1, 2 oder 1...4) der Schnittstelle, die initialisiert werden soll.	LONG
baud_rate	Übertragungsgeschwindigkeit in Baud: RS232: 35 ... 115200 Baud. RS485: 35 ... 2304000 Baud. RS422: 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: RS232, kein Handshake. 1: RS232, Hardware-Handshake (RTS/CTS). 2: RS232, Software-Handshake (Xon/Xoff). 3: RS485, kein Handshake. 4: RS422, kein Handshake. 6: RS422, Software-Handshake (Xon/Xoff).	LONG

Bemerkungen



Diese Anweisung ist vor dem ersten Arbeiten mit der gewählten Schnittstelle notwendig, um die Schnittstellen-Parameter einzustellen. Die Parameter 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.

Übliche Baudraten [Bit/s]		
2304000	57600	2400
1152000	38400	1200
460800	19200	600

Übliche Baudraten [Bit/s]		
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](#), [P2_Write_Fifo](#), [P2_Write_Fifo_Full](#)

Gültig für

[RS422-4 Rev. E](#), [RSxxx-2 Rev. E](#), [RSxxx-4 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_RS_Reset(1)           'RS-Modul zurücksetzen
Rem Initialisierung Schnittstelle 1 auf Modul 1 mit 9600 Baud,
Rem ohne Parität, 8 Datenbits, 1 Stoppbit und
Rem Hardware-Handshake (nur RS232).
P2_RS_Init(1, 1, 9600, 0, 8, 0, 1)
```

Siehe auch weitere [Beispiele für RS232 und RS485 \(Pro II\)](#) ab Seite 2084.

P2_RS_Reset

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](#), [P2_Write_Fifo](#), [P2_Write_Fifo_Full](#)

Gültig für

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
```

Init:

```
P2_RS_Reset(1)           'RS-Modul zurücksetzen  
Rem Initialisierung Schnittstelle 1 auf Modul 1 mit 9600 Baud,  
Rem ohne Parität, 8 Datenbits, 1 Stoppbit und  
Rem Hardware-Handshake (nur RS232).  
P2_RS_Init(1, 1, 9600, 0, 8, 0, 1)
```

Siehe auch weitere [Beispiele für RS232 und RS485 \(Pro II\)](#) ab Seite 2084.

P2_RS485_Send legt die Übertragungsrichtung für eine bestimmte Schnittstelle des angegebenen Moduls fest.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_RS485_Send(module, channel, dir)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Einzustellende Schnittstelle (1, 2 oder 1...4).	LONG
dir	Übertragungsrichtung der Schnittstelle: 0: Schnittstelle als Empfänger einstellen. 1: Schnittstelle als Sender einstellen. 2: Schnittstelle als Sender einstellen, der gleichzeitig die gesendeten Daten empfängt.	LONG

Bemerkungen

Die Einstellung der Übertragungsrichtung bedeutet:

- Empfänger: Die Schnittstelle kann Daten auf dem Bus ausschließlich lesen, auch wenn Daten im Ausgangs-FIFO des Controllers für diese Schnittstelle liegen.
- Sender: Die Schnittstelle kann Daten auf den Bus legen, die von anderen Teilnehmern gelesen werden können.
- Sender/Empfänger: Die Schnittstelle 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](#), [P2_Write_Fifo](#), [P2_Write_Fifo_Full](#)

Gültig für

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

RS485: Daten empfangen und senden Siehe Beispiel „RS485: Daten empfangen und senden“ auf Seite 2101.

P2_RS485_Send

P2_RS_Set_LED

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

Sie schalten die LED oben auf der Frontplatte mit **P2_Set_LED**.

Siehe auch

[P2_Set_LED](#)

Gültig für

RS422-4 Rev. E, 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_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 beschreibenden Registers.	LONG
value	Wert, der in das Register geschrieben werden soll.	LONG

Bemerkungen

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers: TL16C754 von Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

[P2_Check_Shift_Reg](#), [P2_Get_RS](#), [P2_Read_Fifo](#), [P2_RS_Init](#), [P2_RS_Reset](#), [P2_RS485_Send](#), [P2_Write_Fifo](#), [P2_Write_Fifo_Full](#)

Gültig für

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

Beispiel

- / -

P2_Set_RS

P2_Write_Fifo

P2_Write_Fifo schreibt einen Wert in den Sende-FIFO einer bestimmten Schnittstelle 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	Nummer (1, 2 oder 1...4) der Schnittstelle, deren Sende-FIFO beschrieben wird.	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](#), [P2_Write_Fifo_Full](#)

Gültig für

RS422-4 Rev. E, 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 Schnittstelle 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 \(Pro II\)](#) ab Seite 2084.

P2_Write_Fifo_Full gibt zurück, ob noch mindestens ein Speicherplatz im Sende-FIFO einer bestimmten Schnittstelle auf dem angegebenen Modul frei ist.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Write_Fifo_Full(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2 oder 1...4) der Schnittstelle, deren Sende-FIFO beschrieben wird.	LONG
ret_val	Statusmeldung: 0: Im Sende-FIFO ist mindestens ein Element frei. 1: Sende-FIFO ist voll.	LONG

Bemerkungen

Der Rückgabewert ist der gleiche wie bei **P2_Write_Fifo**.

Siehe auch

[P2_Check_Shift_Reg](#), [P2_Get_RS](#), [P2_Read_Fifo](#), [P2_RS_Init](#), [P2_RS_Reset](#), [P2_RS485_Send](#), [P2_Set_RS](#), [P2_Write_Fifo](#)

Gültig für

RS422-4 Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

P2_Write_Fifo_Full

Beispiel

```

Rem sending data to and receiving data from the PC while using
Rem a Fifo in ADwin-Pro II
#include ADwinPro_All.inc

#define outfifo Data_1
#define infifo Data_2
#define rs_adr 5
#define rs_channel 1

Dim outfifo[1000] As Long As Fifo
Dim infifo[1000] As Long As Fifo
Dim value, dummy, check As Long

Rem use LED as signal: red = sending, green = receiving,
Rem orange (red+green) = sending + receiving
Dim red_led, green_led As Long
Dim green_led_time As Long
Dim led_time As Long

Init:
    Rem reset and initialize channel
    P2_RS_Reset(rs_adr)
    P2_RS_Init(rs_adr, 1, 9600, 0, 8, 0, 0)
    Fifo_Clear(1)
    Fifo_Clear(2)
    green_led = 0
    red_led = 0

Event:
    Rem sending
    If (Fifo_Full(1) > 0) Then 'any data present?
        If (P2_Write_Fifo_Full(rs_adr, rs_channel) = 0) Then
            Rem send Fifo empty?
            value = outfifo 'read value from Fifo
            dummy = P2_Write_Fifo(rs_adr, rs_channel, value)
            Rem dummy is not to be checked, since Write_Fifo_Full has
            Rem proved that Fifo has empty elements.

            'do LED settings
            If (red_led = 0) Then
                red_led = 1
                led_time = Read_Timer()
            EndIf

        EndIf
    EndIf

    Rem receiving
    If (Fifo_Empty(2) > 0) Then 'are there empty elements?
        check = P2_Read_Fifo(rs_adr, rs_channel)

        If (check <> -1) Then 'is a value in the receiving buffer?
            inFifo = check 'get value into inFifo

            'do LED settings
            If (green_led = 0) Then
                green_led = 1
                led_time = Read_Timer()
            EndIf
        EndIf
    EndIf

    'output LED settings
    dummy = (red_led And 1) Or Shift_Left(green_led And 1, 1)

```

```
P2_RS_Set_LED(rs_adr, rs_channel, dummy)
If ((red_led > 0) Or (green_led > 0)) Then
  If ((Read_Timer() - led_time) > 20000000) Then
    If (red_led > 0) Then Inc red_led
    If (green_led > 0) Then Inc green_led
    led_time = Read_Timer()
  EndIf
EndIf

If ((red_led = 3) Or (green_led = 3)) Then
  red_led = 0
  green_led = 0
EndIf
```

Siehe auch weitere [Beispiele für RS232 und RS485 \(Pro II\)](#) ab Seite 2084.

3.15 Pro II: Profibus/Profinet-Schnittstelle

Dieser Abschnitt enthält Befehle zum Ansprechen der Profibus- und Profinet-Schnittstellen auf *ADwin-Pro II*.

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit LIN-Bus-Schnittstellen gelten:

- [P2_Init_Profibus](#) (Seite 339)
- [P2_Run_Profibus](#) (Seite 341)
- [P2_Init_Profibus_M40](#) (Seite 342)
- [P2_Run_Profibus_M40](#) (Seite 344)
- [P2_Init_ProfinetIO](#) (Seite 345)
- [P2_Run_ProfinetIO](#) (Seite 347)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

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 Länge der Datenbereiche müssen die gleichen sein wie bei der Projektierung des Profibus. Die Länge der Datenbereiche wird bei der Projektierung auch in Worten angegeben: 1 Wort = 2 Byte.

Die Gesamtanzahl der Datenbytes in einem Bereich wird im Befehl **P2_Run_Profibus** benötigt: sie ist das Produkt aus der Anzahl der Datenbereiche und der Länge der Bereiche in Bytes.

Beispiel: Für **in_mod_cnt** = 7 und **in_mod_type** = 3 ist die Gesamtanzahl 7 x 4 Bytes = 28 Bytes.

P2_Init_Profibus



Siehe auch

[P2_Run_Profibus](#)

Gültig für

Profi-SL Rev. E

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. data bytes input: 38x2=76;
    Rem data bytes output areas: 38x1=76)
    error = P2_Run_Profibus(module,out_arr,76,in_arr,76,conf_arr)
    error = error And 7h
    Par_2 = error

    Rem here the received data in in_arr[] can be processed
```


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 Profibus schreibt.	ARRAY LONG
<code>out_pd_arr_len</code>	Anzahl der Datenbytes (1...76) in den Ausgangsbereichen, die aus dem Feld <code>out_pd_arr[]</code> gelesen werden. Die Anzahl darf nicht größer sein als die Gesamtanzahl der Datenbytes, die bei P2_Init_Profibus durch <code>out_mod_cnt</code> und <code>out_mod_type</code> 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 die Gesamtanzahl der Datenbytes, die bei P2_Init_Profibus durch <code>in_mod_cnt</code> und <code>in_mod_type</code> 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 7:0). 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.

Die Gesamtanzahl der Datenbytes in einem Bereich wird im Befehl **P2_Init_Profibus** festgelegt: sie ist das Produkt aus der Anzahl der Datenbereiche und der Länge der Bereiche in Bytes.

Beispiel: Für `in_mod_cnt` = 7 und `in_mod_type` = 3 ist die Gesamtanzahl $7 \times 4 \text{ Bytes} = 28 \text{ Bytes}$.

Siehe auch

[P2_Init_Profibus](#)

Gültig für

Profi-SL Rev. E

Beispiel

siehe [P2_Init_Profibus](#)

P2_Run_Profibus



P2_Init_Profibus_M40

P2_Init_Profibus_M40 initialisiert den Profibus-Slave.

Syntax

```
#include ADwinPro_All.inc  
  
ret_val = P2_Init_Profibus_M40(module, dev_adr,  
                                in_mod_cnt, out_mod_cnt, work_arr[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
dev_adr	Slave Knotenadresse (1...125) auf dem Profibus.	LONG
in_mod_cnt	Anzahl der Elemente (1, 2, 4, 8, 16, 32, 61) für eingehende Daten in Doppelworten.	LONG
out_mod_cnt	Anzahl der Elemente (1, 2, 4, 8, 16, 32, 61) für ausgehende Daten in Doppelworten.	LONG
work_arr[]	Feld, das für den Betrieb des Profinet-Slave initialisiert wird. Das Feld muss mindestens AB_WORK_ARR_LEN (5000) Elemente haben.	ARRAY LONG
ret_val	Status der Initialisierung: 0: kein Fehler. -1: Fehler, die Größen der Datenbereiche sind falsch.	LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem Profibus-Slave ausgeführt werden.

P2_Init_Profibus_M40 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 und die Anzahl der Datenelemente müssen die gleichen sein wie bei der Projektierung des Profibus. Die Datenbereiche werden bei der Projektierung auch in Worten angegeben: 1 Wort = 2 Byte.

Siehe auch

[P2_Run_Profibus_M40](#)

Gültig für

Profi-SL-40 Rev. E



Beispiel

```
#Include ADwinPro_All.INC
#Define module 5          'module address
#Define node 2            'slave node address
#Define out_arr Data_2
#Define in_arr Data_3

Dim out_arr[61] As Long
Dim in_arr[32] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim error As Long

Init:
    Processdelay = 3000000    'set to 100 Hz
    Rem initialize profibus interface: 38 DWords in, 61 DWords out
    error = P2_Init_Profibus_M40(module, node, 32, 61, conf_arr)
    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 61
        out_arr[i] = (out_arr[i] + i) And 0FFh
    Next i

    Rem send and read data. send 60 DWords, receive 30 DWords
    Par_2 = P2_Run_Profibus_M40(module, in_arr, out_arr, 30, 60, 1,
        conf_arr)

    Rem received data in in_arr[] can be processed
```

P2_Run_Profibus_M40

P2_Run_Profibus_M40 tauscht Daten mit dem Profibus-Slave aus.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Run_Profibus_M40(module, in_pd_arr[],
    out_pd_arr[], in_pd_arr_len, out_pd_arr_len,
    work_arr[], flowrate)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>in_pd_arr[]</code>	Feld, in das der Profibus-Slave Daten schreibt, die vom Profibus gelesen werden.	ARRAY LONG
<code>out_pd_arr[]</code>	Feld, aus dem der Profibus-Slave Daten liest und auf den Profibus schreibt.	ARRAY LONG
<code>in_pd_arr_len</code>	Anzahl der Doppelworte (1, 2, 4, 8, 16, 32, 61), die in <code>in_pd_arr[]</code> übergeben werden.	LONG
<code>out_pd_arr_len</code>	Anzahl der Doppelworte (1, 2, 4, 8, 16, 32, 61), die in <code>out_pd_arr[]</code> übergeben werden.	LONG
<code>work_arr[]</code>	Feld, das Daten für den Betrieb des Profibus-Slave enthält, siehe P2_Init_Profibus_M40 .	ARRAY LONG
<code>flowrate</code>	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>ret_val</code>	Betriebszustand des Profibus-Slave: 0: Initialisierung. 2: Slave wartet auf Busstart durch den Master. 3: Im Leerlauf. 4: In Betrieb. andere: Fehler, Konfiguration prüfen.	LONG

Bemerkungen

P2_Run_Profibus_M40 soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochpriorien Prozess würde die Kommunikation zwischen PC und ADwin-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Siehe auch

[P2_Init_Profibus_M40](#)

Gültig für

Profi-SL-40 Rev. E

Beispiel

siehe [P2_Init_Profibus](#)



P2_Init_ProfinetIO initialisiert ein Feld für den Betrieb mit dem Profinet-Slave.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Init_ProfinetIO(module, in_mod_cnt,
                             out_mod_cnt, work_arr[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
in_mod_cnt	Größe (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320) des Eingangs-Datenbereichs im Profinet-Slave in Doppelworten; 1 Doppelwort = 4 Byte.	LONG
out_mod_cnt	Größe (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320) des Ausgangs-Datenbereichs im Profinet-Slave in Doppelworten; 1 Doppelwort = 4 Byte.	LONG
work_arr[]	Feld, das für den Betrieb des Profinet-Slave initialisiert wird. Das Feld muss mindestens AB_WORK_ARR_LEN (5000) Elemente haben.	ARRAY LONG
ret_val	Status zur Befehlsausführung: 0: Initialisierung war erfolgreich. -1: Fehler, die Größen der Datenbereiche sind falsch.	LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem Profinet-Slave ausgeführt werden.

Kleinere Datenbereiche beschleunigen den Datenverkehr über den Profi-net-Bus.

Die Größe der Datenbereiche muss die gleiche sein wie bei der Projektierung des Profinet. Achten Sie darauf, dass die Größe der Datenbereiche bei der Projektierung auch in anderen Einheiten als Byte angegeben werden kann (z.B. Word, QWord).

P2_Init_Profinet M40 soll in einem Programmabschnitt mit niedriger Priorität ausgeführt werden, weil die Ausführung längere Zeit dauert. Bei einem Aufruf in einem (nicht unterbrechbaren) hochprioritären Prozess würde die Kommunikation zwischen PC und ADwin-System zu lange unterbrochen und daher eine Fehlermeldung (Timeout) erzeugen.

Siehe auch

- / -

Gültig für

Profi-IRT-CU-40 Rev. E, Profi-IRT-FO-40 Rev. E

P2_Init_ProfinetIO



Beispiel

```
#Include ADwinPro_All.inc
#Define out_arr Data_2
#Define in_arr Data_3
#Define module 3

Dim out_arr[1000] As Long
Dim in_arr[1000] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim state As Long

LowInit:
    Processdelay = 3000000 'set to 100 Hz
    Rem init Profinet interface: input data area = 128 DWords
    Rem and output data area = 256 DWords
    Par_10 = P2_Init_ProfinetIO(module, 128, 256, conf_arr)

Event:
    Rem set data in out_arr[] to be transferred
    For i = 1 To 256
        out_arr[i] = (out_arr[i] + i)
    Next i

    Rem send and read data, flowrate slow
    state = P2_Run_ProfinetIO(module, out_arr, in_arr, 128, 256,
        conf_arr, 1)
    Par_2 = state

    Rem now process received data stored in in_arr[1..128];
    Rem in_arr[129..1000] remains unused here.
```

P2_Run_ProfinetIO tauscht Daten mit dem Profinet-Slave aus.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Run_ProfinetIO(module, in_pd_arr[],
    out_pd_arr[], in_pd_arr_len, out_pd_arr_len,
    work_arr[], flowrate)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
in_pd_arr[]	Feld, in das der Profinet-Slave Daten schreibt (Anzahl in_pd_arr_len), die vom Profinet-Bus gelesen werden.	ARRAY LONG
out_pd_arr[]	Feld, aus dem der Profinet-Slave Daten liest (Anzahl out_pd_arr_len) und auf den Profinet-Bus schreibt.	ARRAY LONG
in_pd_arr_len	Anzahl der Doppelworte (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320), die in in_pd_arr[] übergeben werden.	LONG
out_pd_arr_len	Anzahl der Doppelworte (1, 2, 4, 8, 16, 32, 64, 128, 196, 256, 320), die in out_pd_arr[] übergeben werden.	LONG
work_arr[]	Feld, das Daten für den Betrieb des Profinet-Slave enthält, siehe P2_Init_ProfinetIO .	ARRAY LONG
flowrate	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
ret_val	Betriebszustand des Profinet-Slave: 0: Initialisierung. 2: Slave wartet auf Busstart durch den Master. 3: Im Leerlauf. 4: In Betrieb. andere: Fehler, Konfiguration prüfen.	LONG

Bemerkungen

In jedem Feldelement in **in_pd_arr[]** und **out_pd_arr[]** werden 4 Datenbytes = 1 Doppelwort gespeichert. Ein Doppelwort entspricht einem Wert vom Datentyp **Long**.

Siehe auch

[P2_Init_ProfinetIO](#)

Gültig für

[Profi-IRT-CU-40 Rev. E](#), [Profi-IRT-FO-40 Rev. E](#)

Beispiel

siehe [P2_Init_ProfinetIO](#)

P2_Run_Profinet-IO

3.16 Pro II: MIL-STD-1553

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit MIL-STD-1553-Schnittstelle gelten:

- [P2_MIL_Reset](#) (Seite 349)
- [P2_MIL_SMT_Init](#) (Seite 350)
- [P2_MIL_SMT_Message_Read](#) (Seite 351)
- [P2_MIL_SMT_Set_All_Filters](#) (Seite 353)
- [P2_MIL_SMT_Set_Filter](#) (Seite 354)
- [P2_MIL_Set_LED](#) (Seite 355)
- [P2_MIL_Set_Register](#) (Seite 356)
- [P2_MIL_Get_Register](#) (Seite 357)

P2_MIL_Reset initialisiert die MIL-Schnittstelle auf einem bestimmten Modul und setzt alle Register auf die Vorgabewerte zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_MIL_Reset(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Status der Initialisierung: 0: Initialisierung war erfolgreich. <>0: Fehler.	LONG

Bemerkungen

P2_MIL_Reset muss ausgeführt werden, bevor Sie auf die MIL-Schnittstelle zugreifen. Der Befehl sollte im Abschnitt **Init:** ausgeführt werden.

Siehe auch

[P2_MIL_SMT_Init](#), [P2_MIL_SMT_Message_Read](#), [P2_MIL_SMT_Set_All_Filters](#), [P2_MIL_SMT_Set_Filter](#), [P2_MIL_Set_LED](#)

Gültig für

MIL-1553 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define mod_adr 4
#Define cmd_dat Data_1
#Define msg_dat Data_2
```

```
Dim cmd_dat[20] As Long
Dim msg_dat[20] As Long
Dim state As Long
```

Init:

```
Rem Initialize MIL module
Par_1 = P2_MIL_Reset(mod_adr)
If (Par_1 <> 0) Then Exit 'error
Rem initialize SMT 16 bit, time tag counter 64µs
P2_MIL_SMT_Init(mod_adr, 1, 7)
Rem disable all subaddresses for read and write
P2_MIL_SMT_Set_All_Filters(mod_adr, 1, 1)
Rem record RT 8, all subaddresses receive and transmit
Par_2 = P2_MIL_SMT_Set_Filter(mod_adr, 8, 0FFh, 0FFh)
```

Event:

```
Rem check for new message
Par_1 = P2_MIL_SMT_Message_Read(mod_adr, cmd_dat, msg_dat)
If (Par_1 >= 0) Then 'new message found
    Rem process message
    Rem ...
EndIf
```

P2_MIL_Reset

P2_MIL_SMT_Init

P2_MIL_SMT_Init initialisiert den Modus SMT-Monitor 16 Bit (simple monitoring terminal) für beide Busse A und B auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_MIL_SMT_Init(module, timetag_mode, clock_source)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>timetag_mode</code>	Modus für die Zeitmarke: 1: SMT 16 Bit. Andere Modi werden in <i>ADbasic</i> noch nicht unterstützt.	LONG
<code>clock_source</code>	Taktgeber für die Zeitmarke: 0: Zeitmarke ausgeschaltet. 2: Interner Takt 2µs. 3: Interner Takt 4µs. 4: Interner Takt 8µs. 5: Interner Takt 16µs. 6: Interner Takt 32µs. 7: Interner Takt 64µs. 8: Interner Takt 100µs.	LONG

Bemerkungen

Der SMT-Busmonitor überwacht sowohl Bus A als auch Bus B. Der Bus-Controller oder ein Terminal des Moduls können parallel einen oder beide Busse nutzen.

Wenn die Zeitmarke ausgeschaltet ist, wird die Zeitmarke immer auf 0 gesetzt. Der externe Taktgeber wird nicht unterstützt.

Andere Monitor-Optionen (SMT 48 Bit, IRIG Monitor Terminal IMT) können verwendet werden, indem die entsprechenden Register gesetzt werden.

Siehe auch

[P2_MIL_Reset](#), [P2_MIL_SMT_Message_Read](#), [P2_MIL_SMT_Set_All_Filters](#), [P2_MIL_SMT_Set_Filter](#), [P2_MIL_Set_LED](#)

Gültig für

MIL-1553 Rev. E

Beispiel

siehe [P2_MIL_Reset](#)

P2_MIL_SMT_Message_Read liest die Zwischenspeicher für Befehle und Daten der zuletzt gespeicherten MIL-Nachricht auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc
```

```
ret_val = P2_MIL_SMT_Message_Read(module, cmd_dat[],  
    msg_dat[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
cmd_dat[]	Feld, in dem die 4 Worte (16 Bit) aus dem Zwischenspeicher für Befehle gespeichert werden.	ARRAY LONG
	cmd_dat[1] = Blockstatus des SMT.	
	cmd_dat[2] = Zeitmarke der Nachricht.	
	cmd_dat[3] = Zeiger auf den Zwischenspeicher für Daten.	
	cmd_dat[4] = Befehlswort der Nachricht.	
	Das Feld cmd_dat[] muss mindestens 4 Elemente groß sein.	
msg_dat[]	Feld, in dem Datenworte (16 Bit) gespeichert werden.	ARRAY LONG
	Die Anzahl der Datenworte wird in ret_val angegeben.	
	Die Feldgröße sollte mindestens 35 sein.	
ret_val	Lesestatus: -1: keine Nachricht vorhanden. 0...n: Anzahl der Datenworte, die in msg_dat[] gespeichert sind.	LONG

Bemerkungen

Sie können nur die Daten der zuletzt empfangenen Nachricht lesen. Sobald eine neue MIL-Nachricht vollständig empfangen und gespeichert ist, sind die Informationen früherer Nachrichten verloren.

Verwenden Sie **P2_MIL_SMT_Init**, um den Zählmodus für die Zeitmarke einzustellen.

Die Datenworte in **msg_dat[]** werden in aufsteigender Reihenfolge gespeichert wie sie über den MIL-Bus geschickt wurden, angefangen beim Feldelement 1.

Der Zeiger auf den Datenblock (**cmd_dat[3]**) wird nur aus technischen Gründen angegeben, er hat in *ADbasic* keinen Nutzen.

Der Blockstatus (**cmd_dat[1]**) enthält zusätzliche Informationen über Nachrichtenstatus, den verwendeten Bus A/B und eventuell aufgetretene Fehler. Falls vorhanden werden RT-Statusworte in **msg_dat[]** gespeichert.

Die Bits des Blockstatus haben folgende Bedeutung:

Bitnr.	Name	Funktion
15	EOM	Ende der Nachricht: 0: Nachricht ist unvollständig. 1: Nachricht ist vollständig. SOM wird ebenfalls rückgesetzt.
14	SOM	Beginn der Nachricht: 0: Nachricht ist beendet. 1: Nachricht hat begonnen = ein gültiger Befehl wurde vorher beendet.
13	BID	Verwendeter Bus: 0: Bus A. 1: Bus B.

P2_MIL_SMT_Message_Read

Bitnr.	Name	Funktion
12	EO	Fehler-Flag: Eines der Fehler-Bits (10:9,5:0) ist gesetzt oder eine unvollständige Nachricht wurde durch einen gültigen Befehl überschrieben.
11	RR	0: Befehl mit einem Einzel-Befehlswort. 1: RT-RT-Nachricht, d.h. die Nachricht beginnt mit 2 Befehlsworten.
10	IGE	1: Fehler, ungültiger Abstand.
9	TM	1: Fehler, Timeout bei der Antwort.
8	GDB	0: Fehler (in einer vollständigen Nachricht). 1: Erfolgreiche Übertragung des Datenblocks.
7	DSR	1: Rollover im Datenpuffer.
6	SFS	1: Status-Flag gesetzt.
5	LE	1: Fehler, Wortanzahl.
4	SE	1: Fehler, Sync-Typ.
3	WE	1: Fehler, ungültiges Wort.
2	RRGSA	1: Fehler, RT-RT-Nachricht im ersten Befehlswort.
1	RRCW2	1: Fehler, RT-RT-Nachricht im zweiten Befehlswort.
0	CWCE	1: Fehler, Inhalt des Befehlswords.

Sie finden weitere Informationen über Fehler in der separaten Dokumentation "HI-6310 / MIL-STD-1553 / BC/MT/RT Multi-Terminal Device" von Holt Integrated Circuits Inc.

Siehe auch

[P2_MIL_Reset](#), [P2_MIL_SMT_Init](#), [P2_MIL_SMT_Set_All_Filters](#), [P2_MIL_SMT_Set_Filter](#), [P2_MIL_Set_LED](#)

Gültig für

[MIL-1553 Rev. E](#)

Beispiel

siehe [P2_MIL_Reset](#)

P2_MIL_SMT_Set_All_Filters sperrt oder aktiviert die Filter für alle Sende- und Empfangs-Unteradressen aller Terminals auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc

P2_MIL_SMT_Set_All_Filters(module, disable_receive,
                           disable_transmit)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	<code>LONG</code>
<code>disable_receive</code>	Filtereinstellung für alle Unteradressen 0...31 aller 32 Terminal-Adressen beim Empfangen: Filtern aller 32 Terminal-Adressen beim Empfangen mit allen zugehörigen Unteradressen 0...31: 0: aktiviert alle Empfangs-Unteradressen. 1: sperrt alle Empfangs-Unteradressen.	<code>LONG</code>
<code>disable_transmit</code>	Filtereinstellung für alle Unteradressen 0...31 aller 32 Terminal-Adressen beim Senden: 0: aktiviert alle Sende-Unteradressen. 1: sperrt alle Sende-Unteradressen.	<code>LONG</code>

Bemerkungen

Verwenden Sie **P2_MIL_SMT_Set_Filter**, um die Filtereinstellungen für ein einzelnes Terminal zu setzen.

Nach dem Einschalten sind alle Adressen und Unteradressen für Senden und Empfangen aktiviert, d. h. der SMT-Monitor speichert jede Nachricht auf den beiden MIL-Bus-Anschlüssen.

Beim Filtern von MIL-Nachrichten berücksichtigt der SMT Monitor die Geräteadresse und die Unteradresse sowie das Statusbit für Senden / Empfangen im Befehlswort. Sie erhalten das Befehlswort mit **P2_MIL_SMT_Message_Read** im Feldelement `cmd_dat[4]`.

Siehe auch

[P2_MIL_Reset](#), [P2_MIL_SMT_Init](#), [P2_MIL_SMT_Message_Read](#), [P2_MIL_SMT_Set_Filter](#), [P2_MIL_Set_LED](#)

Gültig für

MIL-1553 Rev. E

Beispiel

siehe [P2_MIL_Reset](#)

P2_MIL_SMT_Set_All_Filters

P2_MIL_SMT_Set_Filter

P2_MIL_SMT_Set_Filter sperrt oder aktiviert die Filter für alle Sende- und Empfangs-Unteradressen eines Terminals auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_MIL_SMT_Set_Filter(module, rt_addr,
                                rx_subaddr, tx_subaddr)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
rt_addr	Terminal-Adresse (0...31), die gefiltert wird.	LONG
rx_subaddr	Bitmuster als Filtereinstellung für die Unteradressen 0...31 beim Empfangen (siehe Tabelle): Bit = 0: aktiviert die Empfangs-Unteradresse. Bit = 1: sperrt die Empfangs-Unteradresse.	LONG
tx_subaddr	Bitmuster als Filtereinstellung für die Unteradressen 0...31 beim Senden (siehe Tabelle): Bit = 0: aktiviert die Sende-Unteradresse. Bit = 1: sperrt die Sende-Unteradresse.	LONG
ret_val	Ausführungsstatus: 0: OK 1: Fehler, ungültige Terminal-Adresse	LONG

Bitnr.	31	30	...	1	0
Unteradresse	31	30	...	1	0

Bemerkungen

Verwenden Sie **P2_MIL_SMT_Set_All_Filters**, um die Filtereinstellungen für alle Terminals auf einmal zu sperren oder zu aktivieren.

Nach dem Einschalten sind alle Adressen und Unteradressen für Senden und Empfangen aktiviert, d.h. der SMT-Monitor speichert jede Nachricht auf den beiden MIL-Bus-Anschlüssen.

Beim Filtern von MIL-Nachrichten berücksichtigt der SMT Monitor die Geräteadresse und die Unteradresse sowie das Statusbit für Senden / Empfangen im Befehlsword. Sie erhalten das Befehlsword mit **P2_MIL_SMT_Message_Read** im Feldelement `cmd_dat[4]`.

Siehe auch

[P2_MIL_Reset](#), [P2_MIL_SMT_Init](#), [P2_MIL_SMT_Message_Read](#), [P2_MIL_SMT_Set_All_Filters](#), [P2_MIL_Set_LED](#)

Gültig für

MIL-1553 Rev. E

Beispiel

siehe [P2_MIL_Reset](#)

P2_MIL_Set_LED schaltet die Zusatz-LEDs der MIL-Schnittstelle auf dem Modul ein oder aus.

Syntax

```
#Include ADwinPro_All.Inc

P2_MIL_Set_LED (module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster zum Setzen der Zusatz-LEDs. Bit = 0: LED ausschalten. Bit = 1: LED einschalten.	LONG

Bitnr.	31:4	3	2	1	0
LED	–	Bus A, LED 2	Bus A, LED 1	Bus B, LED 2	Bus B, LED 1

Bemerkungen

Sie setzen die LED oben auf der Frontplatte mit **P2_Set_LED**.

Siehe auch

[P2_MIL_Reset](#), [P2_MIL_SMT_Init](#), [P2_MIL_SMT_Message_Read](#), [P2_MIL_SMT_Set_All_Filters](#), [P2_MIL_SMT_Set_Filter](#), [P2_Set_LED](#)

Gültig für

MIL-1553 Rev. E

Beispiel

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

Init:
    Rem initialize MIL controller
    Par_1 = P2_MIL_Reset (mod_adr)
    If (Par_1 <> 0) Then Exit 'error
    P2_MIL_Set_LED (mod_adr, 1100b) 'set both bus A LEDs
```

P2_MIL_Set_LED

P2_MIL_Set_Register

P2_MIL_Set_Register setzt den Wert eines Registers auf der MIL-Schnittstelle auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.Inc  
  
P2_MIL_Set_Register(module, reg_addr, value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
reg_addr	Adresse (0...65535), deren Wert gesetzt wird.	LONG
value	Wert (0...65535), der gesetzt wird.	LONG

Bemerkungen

Sie finden Informationen über die Register der MIL-Schnittstelle in der separaten Dokumentation "HI-6310 / MIL-STD-1553 / BC/MT/RT Multi-Terminal Device" von Holt Integrated Circuits Inc.

Siehe auch

[P2_MIL_Get_Register](#), [P2_MIL_Reset](#), [P2_MIL_SMT_Init](#), [P2_MIL_SMT_Message_Read](#), [P2_MIL_SMT_Set_All_Filters](#), [P2_MIL_SMT_Set_Filter](#), [P2_MIL_Set_LED](#)

Gültig für

[MIL-1553 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.Inc  
#Define mod_adr 4  
  
Init:  
    Rem initialize MIL controller  
    Par_1 = P2_MIL_Reset(mod_adr)  
    If (Par_1 <> 0) Then Exit 'error  
    Rem set register 1000h  
    P2_MIL_Set_Register(mod_adr, 1000h, 3)
```


P2_MIL_Get_Register gibt den Wert eines Registers von der MIL-Schnittstelle auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_MIL_Get_Register(module, reg_addr)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
reg_addr	Adresse (0...65535), deren Wert gelesen wird.	LONG
ret_val	Wert (0...65535), der zurückgegeben wird.	LONG

Bemerkungen

Sie finden Informationen über die Register der MIL-Schnittstelle in der separaten Dokumentation "HI-6310 / MIL-STD-1553 / BC/MT/RT Multi-Terminal Device" von Holt Integrated Circuits Inc.

Siehe auch

[P2_MIL_Set_Register](#), [P2_MIL_Reset](#), [P2_MIL_SMT_Init](#), [P2_MIL_SMT_Message_Read](#), [P2_MIL_SMT_Set_All_Filters](#), [P2_MIL_SMT_Set_Filter](#), [P2_MIL_Set_LED](#)

Gültig für

MIL-1553 Rev. E

Beispiel

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

Init:

```
Rem initialize MIL controller
Par_1 = P2_MIL_Reset(mod_adr)
If (Par_1 <> 0) Then Exit 'error
Rem read register 1000h
Par_10 = P2_MIL_Get_Register(mod_adr, 1000h)
```

P2_MIL_Get_Register

3.17 Pro II: ARINC-429

Dieser Abschnitt beschreibt Befehle, die für Pro II Module mit ARINC-Schnittstelle gelten:

- [P2_ARINC_Reset](#) (Seite 359)
- [P2_ARINC_Config_Transmit](#) (Seite 360)
- [P2_ARINC_Config_Receive](#) (Seite 361)
- [P2_ARINC_Transmit_Fifo_Full](#) (Seite 363)
- [P2_ARINC_Transmit_Fifo_Empty](#) (Seite 364)
- [P2_ARINC_Write_Transmit_Fifo](#) (Seite 365)
- [ARINC_Create_Value32](#) (Seite 366)
- [P2_ARINC_Transmit_Enable](#) (Seite 367)
- [P2_ARINC_Receive_Fifo_Empty](#) (Seite 368)
- [P2_ARINC_Read_Receive_Fifo](#) (Seite 369)
- [ARINC_Split_Value32](#) (Seite 370)
- [P2_ARINC_Set_Labels](#) (Seite 371)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

P2_ARINC_Reset startet einen Master-Reset auf der ARINC-Schnittstelle des Moduls.

Syntax

```
#Include ADwinPro_All.Inc

P2_ARINC_Reset (module)
```

Parameter

module Eingestellte Moduladresse (1...15). `_LONG` |

Bemerkungen

P2_ARINC_Reset muss ausgeführt werden, bevor Sie auf die ARINC-Schnittstelle zugreifen können. Der Befehl sollte im Abschnitt **Init:** verwendet werden. Anschließend konfigurieren Sie die Einstellungen für den Transmitter und/oder die Receiver.

Bei einem Master-Reset werden das Senden und Empfangen von Daten sofort beendet, die Sende- und Empfangs-Fifos sowie die Fifo-Flags gelöscht.

Siehe auch

[P2_ARINC_Config_Transmit](#), [P2_ARINC_Config_Receive](#), [P2_ARINC_Write_Transmit_Fifo](#), [ARINC_Create_Value32](#), [P2_ARINC_Transmit_Enable](#), [P2_ARINC_Transmit_Fifo_Full](#), [P2_ARINC_Read_Receive_Fifo](#), [P2_ARINC_Set_Labels](#)

Gültig für

ARINC-429 Rev. E

Beispiel

siehe [P2_ARINC_Config_Transmit](#) oder [P2_ARINC_Config_Receive](#)

P2_ARINC_Reset

P2_ARINC_Config_Transmit

P2_ARINC_Config_Transmit konfiguriert die Sende-Einstellungen auf der ARINC-Schnittstelle des Moduls.

Syntax

```
#Include ADwinPro_All.Inc
```

```
P2_ARINC_Config_Transmit(module, clk_speed,  
    parity_enable, parity)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
clk_speed	Übertragungs-Bitrate: 0: 100kHz (high speed), Flankenanstieg 1,5µs 1: 12,5kHz (low speed), Flankenanstieg 10µs	LONG
parity_enable	Paritätsprüfung sperren oder aktivieren. 0: Keine Paritätsprüfung, das Parity-Bit ist Teil der übertragenen Daten. 1: Parität prüfen und Parity-Bit setzen.	LONG
parity	Typ der Paritätsprüfung einstellen; nur sinnvoll, wenn parity_enable auf 1 gesetzt ist. 0: Ungerade Parität. 1: Gerade Parität.	LONG

Bemerkungen

Der Befehl sollte im Abschnitt **Init:** verwendet werden.

Beachten Sie, dass die Paritätsprüfung bei den Receivern des Moduls immer aktiv ist; sie kann nicht gesperrt werden.

Siehe auch

[P2_ARINC_Reset](#), [P2_ARINC_Config_Receive](#), [P2_ARINC_Write_Transmit_Fifo](#), [ARINC_Create_Value32](#), [P2_ARINC_Transmit_Enable](#)

Gültig für

ARINC-429 Rev. E

Beispiel

```
#Include ADwinPro_All.Inc
#Define mod_adr 4
#Define number Par_10
#Define value Par_11
Dim arinc_label As Long
```

Init:

```
Rem Initialize ARINC module
P2_ARINC_Reset(mod_adr)
Rem configure transmitter to 100kHz and even parity
P2_ARINC_Config_Transmit(mod_adr, 0, 1, 1)
Rem enable transmitter
P2_ARINC_Transmit_Enable(mod_adr, 1)
number = 0
arinc_label = 10010001b 'set label
```

Event:

```
Rem check for space in transmitter fifo
If (P2_ARINC_Transmit_Fifo_Full(mod_adr) = 0) Then
    Inc number 'increase number to be sent
    If (number > 07FFFFh) Then number = 1
    Rem create value to be sent (with SSM=11b and SDI=01b)
    value = ARINC_Create_Value32(arinc_label, 11b, 01b, number)
    Rem Write value to transmitter fifo
    P2_ARINC_Write_Transmit_Fifo(mod_adr, value)
EndIf
```

P2_ARINC_Config_Receive konfiguriert die Empfangs-Einstellungen auf der ARINC-Schnittstelle des Moduls.

Syntax

```
#Include ADwinPro_All.Inc

P2_ARINC_Config_Receive(module, channel,
    clk_speed, enable_label, enable_decoder,
    decoder_value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des Empfangskanals auf dem Modul.	LONG
clk_speed	Übertragungs-Bitrate: 0: 100kHz (high speed), Flankenanstieg 1,5µs 1: 12,5kHz (low speed), Flankenanstieg 10µs	LONG
enable_label	Labelprüfung sperren oder aktivieren: 0: Labelprüfung sperren; alle eingehenden Werte kommen in den Empfangs-Fifo. 1: Labelprüfung aktivieren; nur Werte mit dem passenden Label kommen in den Empfangs-Fifo.	LONG
enable_decoder	SDI-Prüfung beim Empfangen sperren oder aktivieren: 0: SDI-Prüfung sperren. 1: SDI-Prüfung aktivieren.	LONG
decoder_value	Bitmuster (00b...11b) für die SDI-Entschlüsselung.	LONG

Bemerkungen

Der Befehl sollte im Abschnitt **Init:** verwendet werden.

Labels werden mit **P2_ARINC_Set_Labels** festgelegt.

Wenn die SDI-Prüfung aktiviert ist, kopiert der Receiver nur ARINC-Nachrichten in den Empfangs-Fifo, wenn die SDI-Bits mit dem Bitmuster **decoder_value** übereinstimmen. Der Receiver übergeht alle anderen ARINC-Nachrichten.

Wenn sowohl Labelprüfung als auch SDI-Prüfung aktiviert sind, muss eine eingehende Nachricht beide Bedingungen erfüllen. Ist das nicht der Fall, übergeht der Receiver die Nachricht.

Beachten Sie, dass die Paritätsprüfung bei den Receivern des Moduls immer aktiv ist; sie kann nicht gesperrt werden.

Siehe auch

[P2_ARINC_Reset](#), [P2_ARINC_Config_Transmit](#), [P2_ARINC_Read_Receive_Fifo](#), [P2_ARINC_Set_Labels](#)

Gültig für

ARINC-429 Rev. E

P2_ARINC_Config_Receive

Beispiel

```
#Include ADwinPro_All.Inc
#Define mod_adr 4
#Define labels_1 Data_1
#Define labels_2 Data_2
Dim i As Long
Dim labels_1[16] As Long 'labels array for channel 1
Dim labels_2[16] As Long 'labels array for channel 2

Init:
  Rem Initialize ARINC module
  P2_ARINC_Reset(mod_adr)

  Rem Initialize labels arrays
  For i = 1 To 16
    labels_1[i] = 0
    labels_2[i] = 3
  Next i
  Rem add some more labels
  labels_1[2] = 15
  labels_1[3] = 143 'channel 1 uses labels 0, 15, 143
  P2_ARINC_Set_Labels(mod_adr, 1, labels_1, 1)
  labels_2[2] = 18 'channel 2 uses labels 3, 18
  P2_ARINC_Set_Labels(mod_adr, 1, labels_2, 1)

  Rem configure receiver 1 to 12.5kHz, enable label matching,
  Rem enable decoding with pattern 01b
  P2_ARINC_Config_Receive(mod_adr, 1, 1, 1, 1, 01b)
  Rem configure receiver 2 to 100kHz, enable label matching,
  Rem disable decoding (provide 00b as dummy value)
  P2_ARINC_Config_Receive(mod_adr, 2, 0, 1, 0, 00b)

Event:
  Rem check for values in receiver fifo 1
  If (P2_ARINC_Receive_Fifo_Empty(mod_adr, 1) = 0) Then
    Rem Read value from receiver fifo 1 into Par_10
    Par_10 = P2_ARINC_Read_Receive_Fifo(mod_adr, 1)
    Rem split value into label, ssm, sdi, data, and parity
    ARINC_Split_Value32(Par_10, Par_11, Par_12, Par_13,
      Par_14, Par_15)
  EndIf
  Rem same procedure for receiver fifo 2
  If (P2_ARINC_Receive_Fifo_Empty(mod_adr, 2) = 0) Then
    Rem Read value from receiver fifo 1 into Par_20
    Par_20 = P2_ARINC_Read_Receive_Fifo(mod_adr, 2)
    Rem split value into label, ssm, sdi, data, and parity
    ARINC_Split_Value32(Par_20, Par_21, Par_22, Par_23,
      Par_24, Par_25)
  EndIf
```

P2_ARINC_Transmit_Fifo_Full gibt zurück, ob der Sende-Fifo auf der ARINC-Schnittstelle des Moduls voll ist.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_ARINC_Transmit_Fifo_Full(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Angabe, ob der Sende-Fifo voll ist: 0: Sende-Fifo ist nicht voll. 1: Sende-Fifo ist voll.	LONG

Bemerkungen

Der Sende-Fifo kann bis zu 32 Nachrichten aufnehmen.

Verwenden Sie **P2_ARINC_Transmit_Fifo_Full**, um auf freie Plätze im Sende-Fifo zu prüfen, bevor Sie neue Nachrichten mit **P2_ARINC_Write_Transmit_Fifo** schreiben.

Siehe auch

[P2_ARINC_Config_Transmit](#), [P2_ARINC_Transmit_Fifo_Empty](#), [P2_ARINC_Write_Transmit_Fifo](#), [ARINC_Create_Value32](#), [P2_ARINC_Transmit_Enable](#)

Gültig für

[ARINC-429 Rev. E](#)

Beispiel

siehe [P2_ARINC_Config_Transmit](#)

P2_ARINC_Transmit_Fifo_Full

P2_ARINC_Transmit_Fifo_Empty

P2_ARINC_Transmit_Fifo_Empty gibt zurück, ob der Sende-Fifo auf der ARINC-Schnittstelle des Moduls leer ist.

Syntax

```
#Include ADwinPro_All.Inc
```

```
ret_val = P2_ARINC_Transmit_Fifo_Empty(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
ret_val	Angabe, ob der Sende-Fifo leer ist: 0: Sende-Fifo enthält Daten. 1: Sende-Fifo ist leer.	LONG

Bemerkungen

Der Sende-Fifo kann bis zu 32 Nachrichten aufnehmen.

Mit **P2_ARINC_Transmit_Fifo_Empty** prüfen Sie, ob alle Daten aus dem Sende-Fifo versendet wurden.

Siehe auch

[P2_ARINC_Config_Transmit](#), [P2_ARINC_Transmit_Fifo_Full](#), [P2_ARINC_Write_Transmit_Fifo](#), [ARINC_Create_Value32](#), [P2_ARINC_Transmit_Enable](#)

Gültig für

[ARINC-429 Rev. E](#)

Beispiel

siehe [P2_ARINC_Config_Transmit](#)

P2_ARINC_Write_Transmit_Fifo schreibt einen 32 Bit-Wert in den Sende-Fifo auf der ARINC-Schnittstelle des Moduls.

Syntax

```
#Include ADwinPro_All.inc

P2_ARINC_Write_Transmit_Fifo(module, value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
value	32 Bit-Wert, der versendet werden soll, Format siehe unten.	LONG

Bemerkungen

Verwenden Sie **P2_ARINC_Transmit_Fifo_Full**, um auf freie Plätze im Sende-Fifo zu prüfen, bevor Sie eine neue Nachricht schreiben.

Der Sende-Fifo kann bis zu 32 Nachrichten aufnehmen. Sobald der Fifo voll ist, ignoriert er Versuche weitere Daten hineinzuschreiben.

Verwenden Sie **ARINC_Create_Value32**, um einen 32 Bit-Wert korrekt zu erstellen. Wenn Sie die Bits eines Werts selbst setzen wollen, setzen Sie die Bits in folgender Reihenfolge:

	MSB									LSB			
Bit	31	30	...	14	13	12	11	10	9	8	7:0		
	MSB			Daten		LS B		SDI		SSM		P	Label

Falls Paritätsprüfung mit **P2_ARINC_Config_Transmit** aktiviert ist, wird das Parity-Bit (P) automatisch berechnet und gesetzt, sobald der Wert in den Sende-Fifo geschrieben wird. Wenn die Paritätsprüfung ausgeschaltet ist, bleibt der 32 Bit-Wert unverändert.

Sobald der Transmitter freigegeben ist, werden alle Werte im Sende-Fifo so schnell wie möglich über den ARINC-Bus gesendet.

Siehe auch

[P2_ARINC_Config_Transmit](#), [P2_ARINC_Transmit_Fifo_Full](#), [P2_ARINC_Transmit_Fifo_Empty](#), [P2_ARINC_Transmit_Enable](#), [ARINC_Create_Value32](#), [P2_ARINC_Read_Receive_Fifo](#)

Gültig für

ARINC-429 Rev. E

Beispiel

siehe [P2_ARINC_Config_Transmit](#)

P2_ARINC_Write_Transmit_Fifo

ARINC_Create_Value32

ARINC_Create_Value32 erzeugt einen 32 Bit-Wert aus den Komponenten SSM, SDI, Daten und Label.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = ARINC_Create_Value32(arinc_label, ssm, sdi,
                                data)
```

Parameter

arinc_label	ARINC-Label, 8 Bits.	LONG
ssm	Sign/status matrix (SSM), 2 Bits.	LONG
sdi	Source/destination identifier (SDI), 2 Bits.	LONG
data	Daten, 19 Bits.	LONG
ret_val	32 Bit-Wert, Format siehe unten.	LONG

Bemerkungen

Stellen Sie sicher, dass ungenutzte Bits in den Übergabeparametern auf Null gesetzt sind, sonst ist der erzeugte Wert ungültig.

Das Format der erzeugten Werts ist in folgender Tabelle dargestellt:

	MSB										LSB	
Bit	31	30	...	14	13	12	11	10	9	8	7:0	
	MSB		Daten		LS B	SDI		SSM		P	Label	

Das Parity-Bit (P) wird auf Null (0) gesetzt. Wenn Sie den Wert mit **P2_ARINC_Write_Transmit_Fifo** in den Sende-Fifo schreiben, wird das Parity-Bit automatisch berechnet und gesetzt (nur wenn die Paritätsprüfung aktiviert ist, siehe **P2_ARINC_Config_Transmit**).

Um einen Wert in seine Komponenten aufzuteilen, verwenden Sie **ARINC_Split_Value32**.

Siehe auch

[P2_ARINC_Reset](#), [P2_ARINC_Config_Transmit](#), [P2_ARINC_Transmit_Fifo_Full](#), [P2_ARINC_Transmit_Fifo_Empty](#), [P2_ARINC_Write_Transmit_Fifo](#), [ARINC_Split_Value32](#)

Gültig für

[ARINC-429 Rev. E](#)

Beispiel

siehe [P2_ARINC_Config_Transmit](#)

P2_ARINC_Transmit_Enable sperrt oder aktiviert das Senden aus dem Sende-Fifo auf der ARINC-Schnittstelle des Moduls.

Syntax

```
#Include ADwinPro_All.Inc

P2_ARINC_Transmit_Enable(module, enable)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
enable	Angabe (0,1), ob Senden gesperrt oder aktiviert ist: 0: Senden ist gesperrt. 1: Senden ist aktiviert.	LONG

Bemerkungen

Wenn Senden aktiviert ist, werden die Daten aus dem Sende-Fifo so schnell wie möglich über den ARINC-Bus verschickt.

Siehe auch

[P2_ARINC_Reset](#), [P2_ARINC_Config_Transmit](#), [P2_ARINC_Transmit_Fifo_Full](#), [P2_ARINC_Transmit_Fifo_Empty](#), [P2_ARINC_Write_Transmit_Fifo](#)

Gültig für

[ARINC-429 Rev. E](#)

Beispiel

siehe [P2_ARINC_Config_Transmit](#)

P2_ARINC_Transmit_Enable

P2_ARINC_Receive_Fifo_Empty

P2_ARINC_Receive_Fifo_Empty gibt zurück, ob der Empfangs-Fifo auf der ARINC-Schnittstelle des Moduls leer ist.

Syntax

```
#Include ADwinPro_All.Inc  
  
ret_val = P2_ARINC_Receive_Fifo_Empty(module,  
                                         channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>channel</code>	Nummer (1, 2) des Empfangskanals.	LONG
<code>ret_val</code>	Angabe, ob das Empfangs-Fifo leer ist: 0: Empfangs-Fifo enthält Daten. 1: Empfangs-Fifo ist leer.	LONG

Bemerkungen

Verwenden Sie **P2_ARINC_Receive_Fifo_Empty**, um auf Daten im Empfangs-Fifo zu prüfen, bevor Sie neue Nachrichten lesen.

Siehe auch

[P2_ARINC_Config_Receive](#), [P2_ARINC_Transmit_Fifo_Empty](#), [P2_ARINC_Read_Receive_Fifo](#), [P2_ARINC_Set_Labels](#)

Gültig für

[ARINC-429 Rev. E](#)

Beispiel

siehe [P2_ARINC_Config_Receive](#)

P2_ARINC_Read_Receive_Fifo gibt einen 32 Bit-Wert aus dem Empfangs-Fifo auf der ARINC-Schnittstelle des Moduls zurück.

Syntax

```
#Include ADwinPro_All.Inc

ret_val = P2_ARINC_Read_Receive_Fifo(module,
                                     channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des Empfangskanals.	LONG
ret_val	Gelesener 32 Bit-Wert.	LONG

Bemerkungen

Verwenden Sie **P2_ARINC_Receive_Fifo_Empty**, um auf Daten im Empfangs-Fifo zu prüfen, bevor Sie neue Nachrichten lesen.

Der Empfangs-Fifo kann bis zu 32 Nachrichten aufnehmen. Wenn der Fifo leer ist, gibt **P2_ARINC_Read_Receive_Fifo** den zuletzt empfangenen 32 Bit-Wert zurück.

Verwenden Sie **ARINC_Split_Value32**, um den 32 Bit-Wert in seine Komponenten aufzuteilen. Wenn Sie die Bits von **ret_val** selbst auswerten wollen, beachten Sie die folgende Bitreihenfolge:

	MSB										LSB	
Bit	31	30	...	14	13	12	11	10	9	8	7...0	
	MSB			Daten		LS B		SDI		SSM		P Label

Siehe auch

[P2_ARINC_Config_Receive](#), [P2_ARINC_Write_Transmit_Fifo](#), [P2_ARINC_Receive_Fifo_Empty](#), [ARINC_Split_Value32](#), [P2_ARINC_Set_Labels](#)

Gültig für

[ARINC-429 Rev. E](#)

Beispiel

siehe [P2_ARINC_Config_Receive](#)

P2_ARINC_Read_Receive_Fifo

ARINC_Split_Value32

ARINC_Split_Value32 teilt einen 32 Bit-Wert in seine Komponenten Label, SSM, SDI, Daten und Parity-Bit auf.

Syntax

```
#Include ADwinPro_All.Inc

ARINC_Split_Value32(value, arinc_label, ssm, sdi,
                    data, parity)
```

Parameter

value	32 Bit-Wert, Format siehe unten.	LONG
arinc_label	ARINC-Label, 8 Bits.	LONG
ssm	Sign/status matrix (SSM), 2 Bits.	LONG
sdi	Source/destination identifier (SDI), 2 Bits.	LONG
data	Daten, 19 Bits.	LONG
parity	Parity-Bit. 0: value hat ungerade Parität. 1: value hat gerade Parität.	LONG

Bemerkungen

Die Auswertung bezieht sich auf das folgende Format:

	MSB							LSB			
Bit	31	30	...	14	13	12	11	10	9	8	7...0
	MSB		Daten		LS B	SDI		SSM		P	Label

Die Modul-Receiver Beachten Sie, dass die Paritätsprüfung bei den Receivern des Moduls immer aktiv ist und damit das Parity-Bit (P) beim Empfang entsprechend geändert wird; die Paritätsprüfung kann nicht gesperrt werden.

Um einen neuen 32 Bit-Wert zu erzeugen, verwenden Sie **ARINC_Create_Value32**.

Siehe auch

[P2_ARINC_Config_Receive](#), [P2_ARINC_Read_Receive_Fifo](#), [P2_ARINC_Receive_Fifo_Empty](#), [P2_ARINC_Set_Labels](#), [ARINC_Create_Value32](#)

Gültig für

[ARINC-429 Rev. E](#)

Beispiel

siehe [P2_ARINC_Config_Receive](#)

P2_ARINC_Set_Labels setzt alle 16 Label eines Receivers auf der ARINC-Schnittstelle des Moduls.

Syntax

```
#Include ADwinPro_All.Inc

P2_ARINC_Set_Labels(module, channel, labels[],
                    array_index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des Empfangskanals.	LONG
labels[]	Quellfeld mit den Label-Werten (je 8 Bit Länge), die gesetzt werden. Die Feldgröße muss mindestens 16 sein.	LONG
array_index	Erstes Feldelement in labels[] , das gelesen wird.	LONG

Bemerkungen

Labels werden nur angewendet, wenn Sie die Label-Prüfung mit **P2_ARINC_Config_Receive, enable_label=1** aktiviert haben.

Wenn die Label-Prüfung aktiviert ist, kopiert der Receiver nur ARINC-Nachrichten in den Empfangs-Fifo, wenn sie mit einem der gesetzten Label übereinstimmen. Der Receiver übergeht alle anderen ARINC-Nachrichten.

Jedes Label hat eine Länge von 8 Bit. In einem Feldelement von **labels[]** wird das Label in den Bits 0...7 abgelegt.

Mit **P2_ARINC_Set_Labels** setzen Sie alle 16 Label auf einmal. Um weniger als 16 Label zu verwenden, füllen Sie nicht verwendete Feldelemente mit einem der bereits genutzten Label.

Beispiel: Wenn Sie 2 Labels benutzen möchten, setzen Sie die Werte der Labels 1 und 2 in **labels[]**, und wiederholen das Label 2 (oder auch Label 1) im Feld 14mal, um auf 16 Werte aufzufüllen.

Siehe auch

[P2_ARINC_Reset](#), [P2_ARINC_Config_Receive](#), [P2_ARINC_Receive_Fifo_Empty](#), [P2_ARINC_Read_Receive_Fifo](#)

Gültig für

ARINC-429 Rev. E

Beispiel

siehe [P2_ARINC_Config_Receive](#)

P2_ARINC_Set_Labels

3.18 Pro II: EtherCAT-Schnittstelle

Dieser Abschnitt enthält Befehle zum Ansprechen der EtherCAT-Schnittstellen auf *ADwin-Pro II*.

- [P2_ECAT_Get_Version](#) (Seite 373)
- [P2_ECAT_Get_State](#) (Seite 374)
- [P2_ECAT_Init](#) (Seite 375)
- [P2_ECAT_Set_Mode](#) (Seite 377)
- [P2_ECAT_Read_Data_16L](#) (Seite 378)
- [P2_ECAT_Write_Data_16L](#) (Seite 379)
- [P2_ECAT_Read_Data_16F](#) (Seite 380)
- [P2_ECAT_Write_Data_16F](#) (Seite 381)
- [P2_Init_EtherCAT](#) (Seite 382)
- [P2_Run_EtherCAT](#) (Seite 384)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

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

ecat_datatable[]	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und EtherCAT-Modul enthält.	ARRAY LONG
ret_val	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.

Siehe auch

[P2_ECAT_Init](#)

Gültig für

[EtherCAT-SL Rev. E](#)

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

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>ecat_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und EtherCAT-Modul enthält.	ARRAY LONG
<code>ret_val</code>	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.

Siehe auch

[P2_ECAT_Init](#), [P2_ECAT_Read_Data_16L](#), [P2_ECAT_Write_Data_16L](#)

Gültig für

[EtherCAT-SL Rev. E](#)

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 ADwin 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.

Die Initialisierung muss mit niedriger Priorität ablaufen, da sie einige Sekunden lang dauert; bei hoher Priorität würde ADwin-Software auf einem angeschlossenen PC nach einer bestimmten Zeit (time-out) die Kommunikation unterbrechen.

Wir empfehlen außerdem, mit **P2_ECAT_Set_Mode** den Datenübertragungs-Modus des EtherCAT-Slave einzustellen (ab Firmware-Version 2.0).

Siehe auch

[P2_ECAT_Get_Version](#), [P2_ECAT_Get_State](#), [P2_ECAT_Read_Data_16L](#), [P2_ECAT_Write_Data_16L](#)

Gültig für

[EtherCAT-SL Rev. E](#)

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 ADwin CPU <-> TiCo
    Par_1 = P2_ECAT_Init(ecat_module_address, ecat_comtable)
    Par_2 = P2_ECAT_Get_Version(ecat_comtable) '10000h
    Rem set data transfer mode to LONG values
    P2_ECAT_Set_Mode(ecat_comtable, 0) 'avail. since firmware v2.0

    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_Set_Mode stellt den Datenübertragungs-Modus des EtherCAT-Slave ein.

Syntax

```
#Include ADwinPro_All.inc

P2_ECAT_Set_Mode(ecat_datatable[], mode)
```

Parameter

<code>ecat_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und EtherCAT-Modul enthält.	ARRAY LONG
<code>mode</code>	Übertragungsmodus des EtherCAT-Slave: 0: Datentyp Long 1: Datentyp Float 2: Datentypen Long und Float	LONG

Bemerkungen

Die Funktion ist verfügbar seit der Firmware-Version 2.0.

Bei `mode=2` arbeitet das Modul mit der halben Datenübertragungsgeschwindigkeit, aber es werden dabei sowohl Long- als auch Float-Daten übertragen.

Siehe auch

[P2_ECAT_Read_Data_16L](#), [P2_ECAT_Write_Data_16L](#), [P2_ECAT_Read_Data_16F](#), [P2_ECAT_Write_Data_16F](#), [P2_ECAT_Get_Version](#)

Gültig für

[EtherCAT-SL Rev. E](#)

Beispiel

siehe [P2_ECAT_Init](#)

P2_ECAT_Set_Mode

P2_ECAT_Read_Data_16L

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 Long-Daten schreibt.	ARRAY LONG
<code>ret_val</code>	Lese-Status: 0: Lesen war erfolgreich. ≠0: Fehler beim Lesen der Daten.	LONG

Bemerkungen

- / -

Siehe auch

[P2_ECAT_Get_State](#), [P2_ECAT_Init](#), [P2_ECAT_Write_Data_16L](#)

Gültig für

[EtherCAT-SL Rev. E](#)

Beispiel

siehe [P2_ECAT_Init](#)

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 ADwin CPU und EtherCAT-Modul enthält.	ARRAY LONG
ecat_outputs[]	Feld, aus dem der EtherCAT-Slave Long-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

- / -

Siehe auch

[P2_ECAT_Get_State](#), [P2_ECAT_Init](#), [P2_ECAT_Read_Data_16L](#)

Gültig für

[EtherCAT-SL Rev. E](#)

Beispiel

siehe [P2_ECAT_Init](#)

P2_ECAT_Write_Data_16L

P2_ECAT_Read_Data_16F

P2_ECAT_Read_Data_16F liest 16 Float-Werte vom EtherCAT-Slave und gibt sie in einem Feld zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_ECAT_Read_Data_16F(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 Float-Daten schreibt.	ARRAY LONG
<code>ret_val</code>	Lese-Status: 0: Lesen war erfolgreich. ≠0: Fehler beim Lesen der Daten.	LONG

Bemerkungen

Die Funktion ist verfügbar seit der Firmware-Version 2.0.

Siehe auch

[P2_ECAT_Get_State](#), [P2_ECAT_Init](#), [P2_ECAT_Write_Data_16F](#)

Gültig für

[EtherCAT-SL Rev. E](#)

Beispiel

siehe [P2_ECAT_Init](#)

P2_ECAT_Write_Data_16F schreibt 16 Float-Werte aus einem Feld zum EtherCAT-Slave.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_ECAT_Write_Data_16F(ecat_datatable[],
    ecat_outputs[])
```

Parameter

ecat_datatable[]	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und EtherCAT-Modul enthält.	ARRAY LONG
ecat_outputs[]	Feld, aus dem der EtherCAT-Slave Float-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

Die Funktion ist verfügbar seit der Firmware-Version 2.0.

Siehe auch

[P2_ECAT_Get_State](#), [P2_ECAT_Init](#), [P2_ECAT_Read_Data_16F](#)

Gültig für

[EtherCAT-SL Rev. E](#)

Beispiel

siehe [P2_ECAT_Init](#)

P2_ECAT_Write_Data_16F

P2_Init_EtherCAT

P2_Init_EtherCAT initialisiert ein Feld für den Betrieb mit dem EtherCAT-Slave.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_Init_EtherCAT(module, in_mod_cnt,  
                           out_mod_cnt, data_type, work_arr[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
in_mod_cnt	Größe (0...360) des Eingangs-Datenbereichs im EtherCAT-Slave in Doppelworten.	LONG
out_mod_cnt	Größe (0...360) des Ausgangs-Datenbereichs im EtherCAT-Slave in Doppelworten.	LONG
data_type	Datentyp für Eingangs- und Ausgangsbereich. Ein Wert belegt ein Doppelwort (=4 Byte). Verfügbar sind: AB_DATA_TYPE_SINT32: Signed integer, 32 Bit. AB_DATA_TYPE_FLOAT: Floating point, 32 Bit.	LONG
work_arr[]	Feld, das für den Betrieb des EtherCAT-Slave initialisiert wird. Das Feld muss mindestens AB_WORK_ARR_LEN (5000) Elemente haben.	ARRAY LONG
ret_val	Status der Initialisierung: 0: kein Fehler. -1: Fehler, die Größen der Datenbereiche sind falsch.	LONG

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem EtherCAT-Slave ausgeführt werden.

Die Initialisierung muss mit niedriger Priorität ablaufen, da sie einige Sekunden lang dauert; bei hoher Priorität würde ein angeschlossener PC nach einer bestimmten Zeit (time-out) die Kommunikation unterbrechen.

Die Größe der Datenbereiche muss die gleiche sein wie bei der Projektierung des EtherCAT. Achten Sie darauf, dass die Größe der Datenbereiche bei der Projektierung auch in anderen Einheiten als Doppelworten angegeben werden kann.

Siehe auch

[P2_Run_EtherCAT](#)

Gültig für

[EtherCAT-SL-40 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.INC

#Define module 5
#Define out_arr Data_2
#Define in_arr Data_3

'Data type according to Init_EtherCAT:
' AB_DATA_TYPE_SINT32 -> Long
' AB_DATA_TYPE_FLOAT -> Float32
Dim out_arr[1000] As Long
Dim in_arr[1000] As Long
Dim conf_arr[AB_WORK_ARR_LEN] As Long
Dim i As Long
Dim ret As Long

LowInit:
    Processdelay = 3000000 'set to 100 Hz
    Rem init EtherCAT interface: input data area = 76 values
    Rem and output data area = 92 values
    Par_10 = P2_Init_EtherCAT(module, 76, 92, AB_DATA_TYPE_SINT32,
        conf_arr)

Event:
    Rem set data in out_arr[] to be transferred
    For i = 1 To 92
        out_arr[i] = (out_arr[i] + i)
    Next i

    Rem send and read data
    state = P2_Run_EtherCAT(module, out_arr, in_arr, 92, conf_arr)
    Par_2 = state

    Rem now process received data stored in in_arr[1..76];
    Rem in_arr[77..92] has been filled with unusable data,
    Rem in_arr[93..1000] remains unused here.
```

P2_Run_EtherCAT

P2_Run_EtherCAT tauscht Daten mit dem EtherCAT-Slave aus.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Run_EtherCAT(module, in_pd_arr[],
    out_pd_arr[], in_pd_arr_len, out_pd_arr_len,
    work_arr[], flowrate)

ret_val = P2_Run_EtherCAT(module, in_pd_arr[],
    out_pd_arr[], in_pd_arr_len, out_pd_arr_len,
    work_arr[], flowrate)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>in_pd_arr[]</code>	Feld, in das der EtherCAT-Slave Daten schreibt (Anzahl <code>in_pd_arr_len</code>), die vom EtherCAT-Bus gelesen werden.	ARRAY LONG FLOAT
<code>out_pd_arr[]</code>	Feld, aus dem der EtherCAT-Slave Daten liest (Anzahl <code>out_pd_arr_len</code>) und auf den EtherCAT-Bus schreibt.	ARRAY LONG FLOAT
<code>in_pd_arr_len</code>	Anzahl der Werte (1...360), die in <code>in_pd_arr[]</code> übergeben werden.	LONG
<code>out_pd_arr_len</code>	Anzahl der Werte (1...360), die in <code>out_pd_arr[]</code> übergeben werden.	LONG
<code>work_arr[]</code>	Feld, das Daten für den Betrieb des EtherCAT-Slave enthält, siehe P2_Init_EtherCAT .	ARRAY LONG
<code>flowrate</code>	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>ret_val</code>	Betriebszustand des EtherCAT-Slave: 0: Initialisierung. 2: Slave wartet auf Busstart durch den Master. 3: Im Leerlauf. 4: In Betrieb. andere: Fehler, Konfiguration prüfen.	LONG

Bemerkungen

Deklarieren Sie die Felder `in_pd_arr[]` und `out_pd_arr[]` mit dem Datentyp, der zu der Einstellung mit **P2_Init_EtherCAT**, Parameter `data_type` passt:

- Long für `AB_DATA_TYPE_SINT32`
- Float32 für `AB_DATA_TYPE_FLOAT`

In jedem Feldelement in `in_pd_arr[]` und `out_pd_arr[]` werden 4 Datenbytes = 1 Doppelwort gespeichert.

Siehe auch

[P2_Init_EtherCAT](#)

Gültig für

[EtherCAT-SL-40 Rev. E](#)

Beispiel

siehe [P2_Init_EtherCAT](#)

3.19 Pro II: Flexray

Dieser Abschnitt beschreibt Befehle, die für Pro II-Module mit Flexray-Schnittstellen gelten:

- [P2_FlexRay_Get_Version](#) (Seite 386)
- [P2_FlexRay_Init](#) (Seite 387)
- [P2_FlexRay_Read_Word](#) (Seite 388)
- [P2_FlexRay_Reset](#) (Seite 389)
- [P2_FlexRay_Set_LED](#) (Seite 390)
- [P2_FlexRay_Write_Word](#) (Seite 391)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp 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_datatable[]	Feld mit Einstellungen für den Zugriff der ADwin CPU auf das FlexRay-Modul.	ARRAY 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 hexadezimaler 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 ADwin 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

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_Init

P2_FlexRay_Read_Word

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

<code>fr_datatable[]</code>	Feld mit Einstellungen für den Zugriff der ADwin CPU auf das FlexRay-Modul.	ARRAY LONG
<code>controller</code>	Nummer (1, 2) des FlexRay-Controllers.	LONG
<code>address</code>	Adresse (0...1FFh) auf dem FlexRay-Controller, deren Wert gelesen wird. Geben Sie die Adresse im 2-Byte-Alignment an.	LONG
<code>status</code>	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
<code>ret_val</code>	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_Reset setzt einen FlexRay-Controller auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_FlexRay_Reset(fr_datatable[], controller, status)
```

Parameter

fr_datatable[]	Feld mit Einstellungen für den Zugriff der ADwin CPU auf das FlexRay-Modul.	ARRAY LONG
controller	Nummer (1, 2) des FlexRay-Controllers.	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_Write_Word](#)

Gültig für

[FlexRay-2 Rev. E](#)

Beispiel

siehe [P2_FlexRay_Init](#)

P2_FlexRay_Reset

P2_FlexRay_Set_LED

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_datatable[]	Feld mit Einstellungen für den Zugriff der ADwin CPU auf das FlexRay-Modul.	ARRAY 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_Write_Word schreibt einen 16 Bit-Wert an eine Adresse in einem Flex-Ray-Controller des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.inc

P2_FlexRay_Write_Word(fr_datatable[],
                      controller, address, value, status)
```

Parameter

fr_datatable[]	Feld mit Einstellungen für den Zugriff der ADwin CPU auf das FlexRay-Modul.	ARRAY LONG
controller	Nummer (1, 2) des FlexRay-Controllers.	LONG
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](#)

P2_FlexRay_Write_Word

SENT-Eingänge

SENT-Ausgänge

3.20 Pro II: SENT-Schnittstelle

Dieser Abschnitt enthält Befehle für SENT-Schnittstellen auf *ADwin-Pro II*.

- [P2_SENT_Init](#) (Seite 393)
- [P2_SENT_Get_Msg_Counter](#) (Seite 395)
- [P2_SENT_Command_Ready](#) (Seite 396)
- [P2_SENT_Get_Version](#) (Seite 394)
- [P2_SENT_Get_ChannelState](#) (Seite 397)
- [P2_SENT_Get_ClockTick](#) (Seite 398)
- [P2_SENT_Get_PulseCount](#) (Seite 399)
- [P2_SENT_Get_Fast_Channel_CRC_OK](#) (Seite 400)
- [P2_SENT_Get_Fast_Channel1](#) (Seite 401)
- [P2_SENT_Get_Fast_Channel2](#) (Seite 403)
- [P2_SENT_Get_Serial_Message_CRC_OK](#) (Seite 404)
- [P2_SENT_Get_Serial_Message_Id](#) (Seite 405)
- [P2_SENT_Get_Serial_Message_Data](#) (Seite 406)
- [P2_SENT_Get_Serial_Message_Array](#) (Seite 407)
- [P2_SENT_Clear_Serial_Message_Array](#) (Seite 409)
- [P2_SENT_Set_CRC_Implementation](#) (Seite 410)
- [P2_SENT_Set_Detection](#) (Seite 411)
- [P2_SENT_Set_ClockTick](#) (Seite 412)
- [P2_SENT_Set_PulseCount](#) (Seite 413)
- [P2_SENT_Set_Sensor_Type](#) (Seite 414)
- [P2_SENT_Request_Latch](#) (Seite 415)
- [P2_SENT_Check_Latch](#) (Seite 416)
- [P2_SENT_Get_Latch_Data](#) (Seite 417)
- [P2_SENT_Set_Output_Mode](#) (Seite 420)
- [P2_SENT_Config_Output](#) (Seite 421)
- [P2_SENT_Config_Serial_Messages](#) (Seite 422)
- [P2_SENT_Enable_Channel](#) (Seite 423)
- [P2_SENT_Invert_Channel](#) (Seite 424)
- [P2_SENT_Set_Reserved_Bits](#) (Seite 425)
- [P2_SENT_Set_Fast_Channel1](#) (Seite 426)
- [P2_SENT_Set_Fast_Channel2](#) (Seite 427)
- [P2_SENT_Set_Serial_Message_Pattern](#) (Seite 428)
- [P2_SENT_Set_Serial_Message_Data](#) (Seite 429)
- [P2_SENT_Fifo_Empty](#) (Seite 430)
- [P2_SENT_Fifo_Clear](#) (Seite 431)
- [P2_SENT_Set_Fifo](#) (Seite 432)

Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

P2_SENT_Init initialisiert die Datenübertragung zwischen ADwin CPU und der SENT-Schnittstelle auf einem bestimmten Modul.

Syntax

```
#Include ADwinPro_All.Inc

REM define SENT settings array
Dim sent_datatable[150] As Long

P2_SENT_Init(module, sent_datatable[])
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>sent_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und SENT-Modul aufnimmt.	ARRAY LONG

Bemerkungen

P2_SENT_Init muss vor der Datenübertragung zwischen ADwin CPU und SENT-Modul ausgeführt werden. Der Befehl sollte im Abschnitt **Init:** stehen. Bei der Initialisierung muss für jedes SENT-Modul ein Feld `sent_datatable[]` mit 150 Elementen angelegt werden.

Siehe auch

[P2_SENT_Get_Serial_Message_Array](#), [P2_SENT_Set_Serial_Message_Pattern](#), [P2_SENT_Get_Latch_Data](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-4-Out Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Latch_Data](#)

P2_SENT_Init

P2_SENT_Get_Version

P2_SENT_Get_Version gibt die Version der SENT-Schnittstelle auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SENT_Get_Version(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
ret_val	Versionsnummer der SENT-Schnittstelle.	LONG

Bemerkungen

Die Versionsnummer wird nur benötigt, wenn Sie Fragen zur Programmierung der SENT-Kanäle an unseren Support haben.

Siehe auch

[P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#), [P2_SENT_Get_ChannelState](#), [P2_SENT_Get_ClockTick](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-4-Out Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc  
  
#Define module 2  
  
Init:  
    REM get software version  
    Par_1 = P2_SENT_Get_Version(module)
```

P2_SENT_Get_Msg_Counter gibt die Anzahl der empfangenen/gesendeten Nachrichten auf dem angegebenen SENT-Kanal zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SENT_Get_Msg_Counter(module,
    sent_channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
ret_val	Anzahl empfangener / gesendeter SENT-Nachrichten.	LONG

Bemerkungen

Für SENT-Eingangsmodule: Solange auf einem SENT-Kanal neue Nachrichten empfangen werden, ist der SENT-Sensor aktiv. Die kontinuierliche Änderung des Rückgabewerts dient daher als Timeout-Wächter für den SENT-Sensor.

Siehe auch

[P2_SENT_Get_ChannelState](#), [P2_SENT_Get_ClockTick](#), [P2_SENT_Get_PulseCount](#), [P2_SENT_Set_Output_Mode](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-4-Out Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_Msg_Counter

P2_SENT_Command_Ready

P2_SENT_Command_Ready gibt zurück, ob die SENT-Schnittstelle auf dem angegebenen Modul bereit ist zum Verarbeiten eines Befehls.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SENT_Command_Ready(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Status der Schnittstelle: 0: Schnittstelle ist bereit für neue Befehle. ≠0: Schnittstelle arbeitet noch.	LONG

Bemerkungen

Verwenden Sie **P2_SENT_Command_Ready**, bevor einer der folgenden Befehle eingesetzt wird:

- **P2_SENT_Set_CRC_Implementation**
- **P2_SENT_Set_Detection**
- **P2_SENT_Set_ClockTick**
- **P2_SENT_Set_PulseCount**
- **P2_SENT_Set_Sensor_Type**
- **P2_SENT_Request_Latch**
- **P2_SENT_Config_Output**
- **P2_SENT_Config_Serial_Messages**
- **P2_SENT_Clear_Serial_Message_Array**
- **P2_SENT_Set_Serial_Message_Pattern**
- **P2_SENT_Set_Serial_Message_Data**
- **P2_SENT_Fifo_Clear**
- **P2_SENT_Set_Fifo**

Wenn Sie einen Befehl verwenden, obwohl die Schnittstelle noch arbeitet, wird der Befehl ignoriert.

Siehe auch

[P2_SENT_Set_CRC_Implementation](#), [P2_SENT_Set_Detection](#), [P2_SENT_Set_ClockTick](#), [P2_SENT_Set_PulseCount](#), [P2_SENT_Set_Sensor_Type](#), [P2_SENT_Request_Latch](#), [P2_SENT_Config_Output](#), [P2_SENT_Config_Serial_Messages](#), [P2_SENT_Clear_Serial_Message_Array](#), [P2_SENT_Set_Serial_Message_Pattern](#), [P2_SENT_Set_Serial_Message_Data](#), [P2_SENT_Fifo_Clear](#), [P2_SENT_Set_Fifo](#)

Gültig für

SENT-4 Rev. E, SENT-4-Out Rev. E, SENT-6 Rev. E

Beispiel

```
#Include ADwinPro_All.inc  
#Define module 2  
#Define channel 1  
  
Init:  
    Rem set CRC mode to 'recommended'  
    Do : Until (P2_SENT_Command_Ready(module) = 0)  
    Par_1 = P2_SENT_Set_CRC_Implementation(module, channel, 1)
```


P2_SENT_Get_ChannelState gibt den Empfangsmodus der SENT-Kanäle auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SENT_Get_ChannelState(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
ret_val	Bitmuster, das den Betriebszustand der SENT-Kanäle angibt. Bit = 0: Erkennungsmodus. Bit = 1: Lesemodus. Die Zuordnung der Bits zu den Kanälen ist in der Tabelle angegeben.	LONG

Bitnr.	31:6	5	4	3	2	1	0
SENT-Kanal	–	6	5	4	3	2	1

Bemerkungen

Nach dem Einschalten sind alle Eingangskanäle im Erkennungsmodus, in dem eingehende SENT-Nachrichten analysiert werden. Sobald das Modul den Basistakt und die Pulsanzahl einer Nachricht erkannt hat, schaltet es den Eingangskanal in den Lesemodus.

Siehe auch

[P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#), [P2_SENT_Get_PulseCount](#), [P2_SENT_Get_ClockTick](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_ChannelState

P2_SENT_Get_ClockTick

P2_SENT_Get_ClockTick gibt den Basistakt eines SENT-Eingangskanals auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SENT_Get_ClockTick(module,  
    sent_channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	_LONG
<code>sent_channel</code>	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
<code>ret_val</code>	Basistakt (1500...90000) in ns.	_LONG

Bemerkungen

Der Basistakt kann für jeden SENT-Eingangskanal unterschiedlich sein.

Nach dem Einschalten des Moduls wird der Basistakt einer eingehenden SENT-Nachricht automatisch erkannt (Erkennungsmodus). Alternativ können Sie den Basistakt mit **P2_SENT_Set_ClockTick** manuell festlegen.

Siehe auch

[P2_SENT_Set_ClockTick](#), [P2_SENT_Command_Ready](#), [P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#), [P2_SENT_Get_ChannelState](#), [P2_SENT_Get_Pulse-Count](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_PulseCount gibt zurück, wie viele Pulse eine SENT-Nachricht an einem Eingangskanal auf dem angegebenen Modul enthält.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SENT_Get_PulseCount(module,
    sent_channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
ret_val	Anzahl von Pulsen in der Nachricht: 9: SENT-Signal ohne Pausenpuls. 10: SENT-Signal mit Pausenpuls.	LONG

Bemerkungen

Eine SENT-Nachricht besteht aus mehreren Pulsen. Der Rückgabewert von **P2_SENT_Get_PulseCount** ist die Anzahl der Pulse in der zuletzt empfangenen SENT-Nachricht.

Bei SENT-Nachrichten mit zwei 12 Bit-Werten ergibt sich mit Pausenpuls eine Nachrichtenlänge von 10 Pulsen:

- Kalibrierpuls zur Synchronisierung
- 1 Nibble-Puls (=4 Bit): Status und Kommunikation
- 3 Nibble-Pulse: erster 12 Bit-Wert (fast channel 1)
- 3 Nibble-Pulse: zweiter 12 Bit-Wert (fast channel 2)
- 1 Nibble-Puls: Prüfsumme
- Pausenpuls (optional)

Siehe auch

[P2_SENT_Set_PulseCount](#), [P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#), [P2_SENT_Get_ChannelState](#), [P2_SENT_Get_ClockTick](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_PulseCount

P2_SENT_Get_Fast_Channel_CRC_OK

P2_SENT_Get_Fast_Channel_CRC_OK gibt das Ergebnis der CRC-Prüfung für die Signale einer SENT-Nachricht auf einem Kanal auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SENT_Get_Fast_Channel_CRC_OK(  
    module, sent_channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
ret_val	Ergebnis der CRC-Prüfung für die 12 Bit-Werte: 0: Übertragung war erfolgreich. 1: Prüfsummenfehler, Fehler bei der Datenübertragung.	LONG

Bemerkungen

Die Prüfsumme bezieht sich auf eine vollständige SENT-Nachricht mit zwei 12 Bit-Werten (fast channels).

Das Modul berechnet aus den zwei 12 Bit-Werten (fast channels) die CRC-Prüfsumme und vergleicht den Wert mit der CRC-Prüfsumme in der SENT-Nachricht. Nur wenn beide Prüfsummen gleich sind, ist der Rückgabewert gleich 0.

Siehe auch

[P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Serial_Message_CRC_OK](#), [P2_SENT_Get_Serial_Message_Id](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_Fast_Channel1 liest den ersten 12 Bit-Wert vom angegebenen SENT-Kanal auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SENT_Get_Fast_Channel1(module,
    sent_channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
ret_val	12 Bit-Wert (0...4095).	LONG

Bemerkungen

In einer SENT-Nachricht sind zwei 12 Bit-Werte enthalten; sie werden auch als „fast channel signals“ bezeichnet. Der Befehl gibt den ersten der beiden Werte zurück.

Siehe auch

[P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#),
[P2_SENT_Get_Serial_Message_CRC_OK](#), [P2_SENT_Get_Serial_Message_Id](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc

#Define module 2
#Define channel 1
#Define ready_state Par_20
#Define returncode Par_21
#Define status Par_22
#Define value1 Par_23
#Define value2 Par_24
#Define msg_cnt Par_25
#Define prev_msg_cnt Par_25

Init:
    Processdelay = 300000    '1 kHz

    REM Wait until detection mode of SENT channel has finished and
    REM read mode is set
    Par_1 = 2^(channel - 1)
    Do
    Until (P2_SENT_Get_ChannelState(module) And Par_1 = Par_1)

    prev_msg_cnt = 0          'message counter (timeout check)

    Rem CRC mode 'recommended'
    Do : Until (P2_SENT_Command_Ready(module) = 0)
    Par_1 = P2_SENT_Set_CRC_Implementation(module, channel, 1)

    Rem read pulse count: 9 = without pause pulse,
    Rem 10 = with pause pulse
    Par_11 = P2_SENT_Get_PulseCount(module, channel)
    Par_12 = P2_SENT_Get_ClockTick(module, channel)

Event:
    Inc Par_2
    If (Par_2 = 1000) Then    'timeout check every second
        Par_2 = 0
```

P2_SENT_Get_Fast_Channel1

```
msg_cnt = P2_SENT_Get_Msg_Counter(module, channel)
If (msg_cnt = prev_msg_cnt) Then
    End
Else
    prev_msg_cnt = msg_cnt 'store counter for next check
EndIf
EndIf

Rem read CRC status of fast channels
If (P2_SENT_Get_Fast_Channel_CRC_OK(module, channel) = 0) Then
    Rem read fast channel values 1+2
    Par_13 = P2_SENT_Get_Fast_Channel1(module, channel)
    Par_14 = P2_SENT_Get_Fast_Channel2(module, channel)
EndIf

Rem read CRC status of serial message
If (P2_SENT_Get_Serial_Message_CRC_OK(module, channel)=0) Then
    Rem read serial message ID and data
    Par_16 = P2_SENT_Get_Serial_Message_Id(module, channel)
    Par_17 = P2_SENT_Get_Serial_Message_Data(module, channel)
EndIf

Rem -- write commands --
ready_state = P2_SENT_Command_Ready(module)
If ((ready_state = 0) And (status <> 0)) Then
    If (status = 1) Then 'reset channel
        value1 = 1
        returncode = P2_SENT_Set_Detection(module, value1)
    EndIf

    If (status = 2) Then 'set clock period
        value1 = 1
        returncode = P2_SENT_Set_ClockTick(module, value1, value2)
    EndIf

    status = 0
EndIf
```

P2_SENT_Get_Fast_Channel2 liest den zweiten 12 Bit-Wert vom angegebenen SENT-Kanal auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SENT_Get_Fast_Channel2(module,
    sent_channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
ret_val	12 Bit-Wert (0...4095).	LONG

Bemerkungen

In einer SENT-Nachricht sind zwei 12 Bit-Werte enthalten; sie werden auch als „fast channel signals“ bezeichnet. Der Befehl gibt den zweiten der beiden Werte zurück.

Siehe auch

[P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#),
[P2_SENT_Get_Serial_Message_CRC_OK](#), [P2_SENT_Get_Serial_Message_Id](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_Fast_Channel2

P2_SENT_Get_Serial_Message_CRC_OK

P2_SENT_Get_Serial_Message_CRC_OK gibt das Ergebnis der CRC-Prüfung einer seriellen Nachricht auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SENT_Get_Serial_Message_CRC_OK(module,  
sent_channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	<code>LONG</code>
<code>sent_channel</code>	Nummer (1...4, 1...6) des SENT-Kanals.	<code>LONG</code>
<code>ret_val</code>	Ergebnis der CRC-Prüfung: 0: Übertragung war erfolgreich. 1: Prüfsummenfehler, Fehler bei der Datenübertragung.	<code>LONG</code>

Bemerkungen

Die Prüfsumme bezieht sich auf die zuletzt vollständig empfangene serielle Nachricht. Eine serielle Nachricht wird über mehrere SENT-Nachrichten verteilt gesendet.

Folgende Sendeformate von seriellen Nachrichten werden erkannt:

- Short Serial Message Format, 12 Bit Länge:
Kennung 4 Bit und Datenwert 8 Bit.
- Enhanced Serial Message Format, 20 Bit Länge:
Kennung und Datenwert 4 Bit/16 Bit oder 8 Bit/12 Bit.

Siehe auch

[P2_SENT_Get_Serial_Message_Id](#), [P2_SENT_Get_Serial_Message_Data](#),
[P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_Serial_Message_Id gibt die Kennung einer seriellen Nachricht auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SENT_Get_Serial_Message_Id(module,
    sent_channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
ret_val	Kennung (0...255) der seriellen Nachricht.	LONG

Bemerkungen

Die Kennung bezieht sich auf die zuletzt vollständig empfangene serielle Nachricht. Eine serielle Nachricht wird über mehrere SENT-Nachrichten verteilt gesendet.

Je nach Sendeformat der seriellen Nachricht hat die Kennung eine Länge von 4 Bit oder 8 Bit. Folgende Formate werden erkannt:

- Short Serial Message Format, 12 Bit Länge:
Kennung 4 Bit und Datenwert 8 Bit.
- Enhanced Serial Message Format, 20 Bit Länge:
Kennung und Datenwert 4 Bit/16 Bit oder 8 Bit/12 Bit.

Siehe auch

[P2_SENT_Get_Serial_Message_CRC_OK](#), [P2_SENT_Get_Serial_Message_Data](#), [P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_Serial_Message_Id

P2_SENT_Get_Serial_Message_Data

P2_SENT_Get_Serial_Message_Data gibt den Datenwert einer seriellen Nachricht auf dem angegebenen Modul zurück.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SENT_Get_Serial_Message_Data(module,  
sent_channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	<code>LONG</code>
<code>sent_channel</code>	Nummer (1...4, 1...6) des SENT-Kanals.	<code>LONG</code>
<code>ret_val</code>	Datenwert (0...65535) der seriellen Nachricht.	<code>LONG</code>

Bemerkungen

Der Datenwert bezieht sich auf die zuletzt vollständig empfangene serielle Nachricht. Eine serielle Nachricht wird über mehrere SENT-Nachrichten verteilt gesendet.

Je nach Sendeformat der seriellen Nachricht hat der Datenwert eine Länge von 8 Bit, 12 Bit oder 16 Bit. Folgende Formate werden erkannt:

- Short Serial Message Format, 12 Bit Länge:
Kennung 4 Bit und Datenwert 8 Bit.
- Enhanced Serial Message Format, 20 Bit Länge:
Kennung und Datenwert 4 Bit/16 Bit oder 8 Bit/12 Bit.

Einen vollständigen Nachrichtensatz von seriellen Nachrichten erhalten Sie mit **P2_SENT_Get_Serial_Message_Array**.

Siehe auch

[P2_SENT_Get_Serial_Message_CRC_OK](#), [P2_SENT_Get_Serial_Message_Id](#), [P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Get_Serial_Message_Array gibt einen vollständigen Nachrichtensatz von seriellen Nachrichten auf einem SENT-Kanal zurück.

Syntax

```
#Include ADwinPro_All.inc

REM define SENT settings array
P2_SENT_Init(module, sent_datatable[])

P2_SENT_Get_Serial_Message_Array(sent_datatable[],
    sent_channel, serial_array[], serial_array_index)
```

Parameter

<code>sent_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und SENT-Modul enthält.	ARRAY LONG
<code>sent_channel</code>	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
<code>serial_array[]</code>	Zielfeld, in dem der Nachrichtensatz gespeichert wird. Das Feld muss mindestens 512 Elemente haben.	ARRAY LONG
<code>serial_array_index</code>	Erstes Feldelement in <code>serial_array[]</code> , in das geschrieben wird.	LONG

Bemerkungen

Führen Sie erst **P2_SENT_Init** aus, bevor Sie den Befehl **P2_SENT_Get_Serial_Message_Array** verwenden.

Eine einzelne serielle Nachricht erhalten Sie mit **P2_SENT_Get_Serial_Message**.

Ab dem Einschalten sammelt und speichert das Modul eintreffende serielle Nachrichten des SENT-Sensors als Nachrichtensatz. Ein Nachrichtensatz wird von mehreren (bis zu 32) seriellen Nachrichten gebildet. Nur für Kennungen, die der Sensor nutzt, werden Daten gespeichert.

Mit **P2_SENT_Clear_Serial_Message_Array** können Sie die zwischengespeicherten Daten auf Null zurücksetzen.

Für jede übertragene Kennung werden im Zielfeld der zuletzt empfangene Datenwert und die Anzahl der vollständig empfangenen seriellen Nachrichten abgelegt (siehe unten); für alle anderen Kennungen bleiben die Feldelemente unverändert. Wenn die Kennung 4 Bit lang ist, können im Feld `serial_array[]` nur die Elemente [1] ... [16] und [257] ... [272] belegt sein.

Im Feld `serial_array[]` werden sowohl die Datenwerte als auch die Anzahl empfangener Nachrichten in folgender Weise übergeben:

	Datenwert	Anzahl Nachrichten
Kennung 0	<code>serial_array[1]</code>	<code>serial_array[257]</code>
Kennung 1	<code>serial_array[2]</code>	<code>serial_array[258]</code>
...
Kennung 15	<code>serial_array[16]</code>	<code>serial_array[272]</code>
...
Kennung 254	<code>serial_array[255]</code>	<code>serial_array[511]</code>
Kennung 255	<code>serial_array[256]</code>	<code>serial_array[512]</code>

Je nach Sendeformat der seriellen Nachricht haben Datenwert und Kennung verschiedene Längen. Folgende Formate werden erkannt:

- Short Serial Message Format, 12 Bit Länge:
Kennung 4 Bit und Datenwert 8 Bit.
- Enhanced Serial Message Format, 20 Bit Länge:
Kennung und Datenwert 4 Bit/16 Bit oder 8 Bit/12 Bit.

Siehe auch

P2_SENT_Get_Serial_Message_Array

P2_SENT_Init, P2_SENT_Get_PulseCount, P2_SENT_Get_Fast_Channel1,
P2_SENT_Get_Fast_Channel2, P2_SENT_Get_Fast_Channel_CRC_OK,
P2_SENT_Get_ChannelState, P2_SENT_Get_ClockTick, P2_SENT_Clear_
Serial_Message_Array

Gültig für

SENT-4 Rev. E, SENT-6 Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

```
#Define module 2
#Define channel 1
#Define id_array Data_1
Dim senttable[150] As Long At DM_Local
Dim id_array[32] As Long At DM_Local
Dim array[512] As Long At DM_Local
Dim i, j As Long
```

Init:

```
Processdelay = 300000      '1 kHz
Rem initialize module, request latch for channel 1 (once)
P2_SENT_Init(module, senttable)

Rem Wait until detection mode of SENT channel has finished and
Rem read mode is set
Par_1 = 2^(channel - 1)
Do : Until (P2_SENT_Get_ChannelState(module) And Par_1 = Par_1)

Rem clear serial message array
Do : Until (P2_SENT_Command_Ready(module) = 0)
Par_2 = P2_SENT_Clear_Serial_Message_Array(module, channel)
```

Event:

```
Rem get set of serial messages
P2_SENT_Get_Serial_Message_Array(senttable, channel, array, 1)

Rem get list of used IDs
For i = 257 To 512
Rem if count > 0, the ID is used
If (array[i] > 0) Then
Inc j
Rem store ID
id_array[j] = i
EndIf
Next i
```

P2_SENT_Clear_Serial_Message_Array setzt den zwischengespeicherten Nachrichtensatz von seriellen Nachrichten eines SENT-Kanals zurück.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Clear_Serial_Message_Array(module,
    sent_channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Clear_Serial_Message_Array** verwenden.

Ab dem Einschalten sammelt das Modul alle eintreffenden seriellen Nachrichten des SENT-Sensors. Mehrere (bis zu 32) serielle Nachrichten bilden zusammen einen Nachrichtensatz, der im Zielfeld abgebildet wird.

Die Abfrage einzelner serieller Nachrichten, z.B. mit **P2_SENT_Get_Serial_Message_Data**, wird durch das Zurücksetzen nicht beeinflusst.

Siehe auch

[P2_SENT_Command_Ready](#), [P2_SENT_Get_PulseCount](#), [P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#), [P2_SENT_Get_ChannelState](#), [P2_SENT_Get_ClockTick](#), [P2_SENT_Get_Serial_Message_Array](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Serial_Message_Array](#)

P2_SENT_Clear_Serial_Message_Array

P2_SENT_Set_CRC_Implementation

P2_SENT_Set_CRC_Implementation setzt den Berechnungsalgorithmus für die CRC-Prüfsumme für einen SENT-Kanal auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Set_CRC_Implementation(module,
    sent_channel, crc_implement)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
crc_implement	Kennziffer für den Berechnungsalgorithmus der CRC-Prüfsumme: 0: Legacy (Voreinstellung). 1: Recommended.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_CRC_Implementation** verwenden.

Nach dem Einschalten ist der Berechnungsalgorithmus „Legacy“ eingestellt. Der Berechnungsalgorithmus gilt sowohl für Signale (fast channels) als auch für serielle Nachrichten.

Siehe auch

[P2_SENT_Set_ClockTick](#), [P2_SENT_Get_ChannelState](#) [P2_SENT_Command_Ready](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Set_Detection setzt einen SENT-Kanal auf dem angegebenen Modul in den Erkennungsmodus.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Set_Detection(module,
    sent_channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>sent_channel</code>	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
<code>ret_val</code>	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_Detection** verwenden.

Im Erkennungsmodus werden eingehende SENT-Nachrichten analysiert. Sobald das Modul den Basistakt und die Pulsanzahl der Nachricht erkannt hat, schaltet es den Eingangskanal in den Lesemodus.

Alternativ können Sie mit **P2_SENT_Set_ClockTick** und **P2_SENT_Set_PulseCount** Basistakt und Pulsanzahl auch manuell festlegen.

Siehe auch

[P2_SENT_Set_ClockTick](#), [P2_SENT_Get_ChannelState](#) [P2_SENT_Command_Ready](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Set_Detection

P2_SENT_Set_ClockTick

P2_SENT_Set_ClockTick schaltet einen SENT-Kanal auf dem angegebenen Modul in den Lesemodus mit einem definierten Basistakt.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Set_ClockTick(module,
    sent_channel, clocktick)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
clocktick	SENT-Basistakt (1500...90000) in Nanosekunden. Ein zu kleiner Basistakt wird automatisch korrigiert.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_ClockTick** verwenden.

Der Empfangskanal wird ohne weitere Prüfung in den Lesemodus geschaltet und wertet SENT-Nachrichten anhand des eingestellten Basistakts aus.

Alternativ kann das Modul den Basistakt der SENT-Nachricht im Erkennungsmodus automatisch selbst erkennen, siehe **P2_SENT_Set_Detection**.

Siehe auch

[P2_SENT_Get_ClockTick](#), [P2_SENT_Set_Detection](#), [P2_SENT_Get_ChannelState](#), [P2_SENT_Command_Ready](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Set_PulseCount stellt ein, ob ein Pausenpuls in SENT-Nachrichten eines Eingangskanals auf dem angegebenen Modul erwartet wird.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Set_PulseCount(module,
    sent_channel, pulse_count)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
pulse_count	Kennziffer für den Pausenpuls: 9: SENT-Nachricht ohne Pausenpuls. 10: SENT-Nachricht mit Pausenpuls.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_PulseCount** verwenden.

Eine SENT-Nachricht besteht aus mehreren Pulsen. Die Kennziffer **pulse_count** legt fest, ob in der SENT-Nachricht ein Pausenpuls erwartet wird oder nicht.

Bei SENT-Nachrichten mit zwei 12 Bit-Werten und Pausenpuls ergibt sich eine Nachrichtenlänge von 10 Pulsen:

- Kalibrierpuls zur Synchronisierung
- 1 Nibble-Puls (=4 Bit): Status und Kommunikation
- 3 Nibble-Pulse: erster 12 Bit-Wert (fast channel 1)
- 3 Nibble-Pulse: zweiter 12 Bit-Wert (fast channel 2)
- 1 Nibble-Puls: Prüfsumme
- Pausenpuls (optional)

Alternativ kann das Modul im Erkennungsmodus automatisch erkennen, ob die SENT-Nachricht einen Pausenpuls enthält, siehe **P2_SENT_Set_Detection**.

Siehe auch

[P2_SENT_Command_Ready](#), [P2_SENT_Get_PulseCount](#), [P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#), [P2_SENT_Get_ChannelState](#), [P2_SENT_Get_ClockTick](#)

Gültig für

SENT-4 Rev. E, SENT-6 Rev. E

Beispiel

siehe [P2_SENT_Get_Fast_Channel1](#)

P2_SENT_Set_PulseCount

P2_SENT_Set_Sensor_Type

P2_SENT_Set_Sensor_Type stellt den erwarteten Sensortyp für die SENT-Nachrichten auf einem Eingangskanal auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Set_Sensor_Type(module,
    sent_channel, sensor_type)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
sensor_type	Kennziffer für den Sensortyp: 0: kein Sensortyp gewählt (Default). 1: throttle position sensor. 2: single secure sensor.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_Sensor_Type** verwenden.

Manche Sensortypen senden Nachrichten mit einer festgelegten Kombination und Funktion der Nibbles. Wenn mit **P2_SENT_Set_Sensor_Type** ein Sensortyp gewählt ist, prüft das Modul, ob die empfangenen SENT-Daten auch den Vorgaben des eingestellten Sensortyps entsprechen.

Wenn die empfangenen SENT-Daten nicht zum erwarteten Sensortyp passen, werden Fehlercodes gesetzt. Die Fehlercodes erhalten Sie mit **P2_SENT_Get_Latch_Data**.

Siehe auch

[P2_SENT_Command_Ready](#), [P2_SENT_Get_Latch_Data](#), [P2_SENT_Get_PulseCount](#), [P2_SENT_Get_Fast_Channel1](#), [P2_SENT_Get_Fast_Channel2](#), [P2_SENT_Get_Fast_Channel_CRC_OK](#), [P2_SENT_Get_ChannelState](#), [P2_SENT_Get_ClockTick](#)

Gültig für

SENT-4 Rev. E, SENT-6 Rev. E

Beispiel

- / -

P2_SENT_Request_Latch fordert an, bestimmte SENT-Kanäle auf dem angegebenen Modul einmalig über den Latch-Zwischenspeicher zu puffern.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Request_Latch(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
pattern	Bitmuster, das die SENT-Kanäle angibt. Bit = 0:Erkennungsmodus. Bit = 1:Lesemodus. Die Zuordnung der Bits zu den Kanälen ist in der Tabelle angegeben.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bitnr.	31:6	5	4	3	2	1	0
SENT-Kanal	–	6	5	4	3	2	1

Bemerkungen

Im Latch-Zwischenspeicher werden Informationen von SENT-Nachrichten gesammelt, bis alle Nibbles einer Nachricht vollständig empfangen sind. Erst dann kann die Nachricht mit **P2_SENT_Get_Latch_Data** aus dem Latch-Zwischenspeicher gelesen werden.

Sie stellen das Zwischenspeichern von SENT-Daten nur für die jeweils nächste SENT-Nachricht ein, nicht dauerhaft.

P2_SENT_Request_Latch setzt die Einstellungen aller SENT-Kanäle gleichzeitig. Wenn Sie nur einen SENT-Kanal setzen wollen, lesen Sie den aktuellen Status mit **P2_SENT_Check_Latch**, ändern nur das Bit zum gewünschten Kanal und fordern die Daten neu an (siehe Beispiel).

Das Puffern über den Latch-Zwischenspeicher hat keinen Einfluss auf das Lesen von Einzelinformationen wie mit **P2_SENT_Get_Fast_Channel1** oder **P2_SENT_Get_Serial_Message_Data**.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Check_Latch](#), [P2_SENT_Get_Latch_Data](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Latch_Data](#)

P2_SENT_Request_Latch

P2_SENT_Check_Latch

P2_SENT_Check_Latch gibt zurück, ob der Latch-Zwischenspeicher Daten der angeforderten SENT-Kanäle enthält.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SENT_Check_Latch(module)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
ret_val	Bitmuster (siehe Tabelle), das angibt, ob Daten im Latch-Zwischenspeicher zum Lesen bereit stehen: Bit = 0: Es stehen Daten zum Lesen bereit. Bit = 1: Der Datensatz ist noch unvollständig.	LONG

Bitnr.	31:6	5	4	3	2	1	0
SENT-Kanal	–	6	5	4	3	2	1

Bemerkungen

P2_SENT_Check_Latch ist nur für die SENT-Kanäle sinnvoll einsetzbar, bei denen Sie das Puffern im Zwischenspeicher mit **P2_Request_Latch** angefordert haben.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Request_Latch](#), [P2_SENT_Get_Latch_Data](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

siehe [P2_SENT_Get_Latch_Data](#)

P2_SENT_Get_Latch_Data liest die Daten einer SENT-Nachricht für einen SENT-Kanal aus dem Latch-Zwischenspeicher.

Syntax

```
#Include ADwinPro_All.inc

REM define SENT settings array
P2_SENT_Init(module, sent_datatable[])

P2_SENT_Get_Latch_Data(sent_datatable[],
    sent_channel, data_array[], data_array_index)
```

Parameter

<code>sent_datatable[]</code>	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und SENT-Modul enthält.	ARRAY LONG
<code>sent_channel</code>	Nummer (1...4, 1...6) des SENT-Kanals.	LONG
<code>data_array[]</code>	Zielfeld, in dem die SENT-Daten gespeichert werden. Das Feld muss mindestens 32 Elementen haben. Die Bedeutung der Feldelemente ist unten beschrieben.	ARRAY LONG
<code>data_array_index</code>	Erstes Feldelement in <code>data_array[]</code> , in das geschrieben wird.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Check_Latch**, ob Daten zum Abholen bereit steht, bevor Sie die Daten abholen. Wenn Sie den Zwischenspeicher auslesen, obwohl der Datensatz noch unvollständig ist, erhalten Sie die Daten der vorherigen Nachricht.

Ein Datensatz besteht aus 32 Feldelementen, die ab dem Index `data_array_index` abgelegt sind. Ein Datensatz enthält alle 8 Nibbles einer SENT-Nachricht sowie einige bereits ausgewertete Informationen.

Die Informationen der seriellen Nachricht (Elemente 7...11) sind im Datensatz nur enthalten, wenn die SENT-Nachricht die serielle Nachricht abschließt. Wenn keine serielle Nachricht im Datensatz enthalten ist, hat das Element 7 den Wert Null.

Der Inhalt der Feldelemente ist unten beschrieben. Zur Vereinfachung der Darstellung ist angenommen, dass `data_array_index` auf 1 gesetzt ist.

Index	Bedeutung
[1]	Anzahl der empfangenen Nachrichten auf dem SENT-Kanal.
[2] [3]	Empfangszeit der SENT-Nachricht als 64 Bit-Zählerwert, aufgeteilt in das untere [2] und obere Wort [3]. Der Zähler arbeitet mit einer Taktfrequenz von 100MHz, d.h. der Zählerwert ist in Einheiten von 10ns angegeben.
[4]	Erster 12 Bit-Wert der Nachricht, fast channel 1.
[5]	Zweiter 12 Bit-Wert der Nachricht, fast channel 2.
[6]	Ergebnis der CRC-Prüfung (fast channels): 0: Übertragung war erfolgreich. 1: Prüfsummenfehler, Fehler bei der Datenübertragung.
[7]	Kennziffer für das Sendeformat der seriellen Nachricht: 0: keine Daten für serielle Nachricht vorhanden. 1: Short Serial Message Format, 12 Bit Länge: Kennung 4 Bit, Datenwert 8 Bit. 2: Enhanced Serial Message Format, 20 Bit Länge: Kennung 4 Bit, Datenwert 16 Bit 3: Enhanced Serial Message Format, 20 Bit Länge: Kennung 8 Bit, Datenwert 12 Bit

P2_SENT_Get_Latch_Data

Index	Bedeutung
[8]	Kennung (ID) der seriellen Nachricht.
[9]	Datenwert der seriellen Nachricht.
[10]	CRC-Prüfsumme der seriellen Nachricht.
[11]	Ergebnis der CRC-Prüfung (serielle Nachricht): 0: Übertragung war erfolgreich. 1: Prüfsummenfehler, Fehler bei der Datenübertragung.
[12]	Bitmuster mit Fehlerbits. Wenn kein Fehler aufgetreten ist, ist der Wert des Bitmusters gleich 0. Bedeutung der gesetzten Bits siehe unten.
[13] ... [19]	reserviert
[20]	Dauer des Kalibrierpulses in Einheiten von 10ns.
[21] ... [28]	Alle 8 Nibbles der SENT-Nachricht in der Reihenfolge: Status-Nibble, Daten-Nibbles 1...6, CRC-Nibble.
[29] ... [32]	reserviert

Gesetzte Fehlerbits im Feldelement 12 haben folgende Bedeutung:

Bitnr.	Bedeutung
0	Stat Reserved 0
1	Stat Reserved 1
2	ungültiger Nibble-Wert: <0 oder >15
3	Kalibrier-Pulslänge ungültig: <56 Basistakte - 1.5625% oder >56 Basistakte + 1.5625%
4	Abs Sync Size Fail
5	2 oder mehr aufeinander folgende Sync-Pulse. Kann geschehen, wenn Pausenpuls und Sync-Puls die gleiche Länge haben.
6	Falsche Pulsanzahl zwischen zwei Sync-Pulsen.
7	Rolling count fail
8	Inverted nibble fail
9	serial message fail
31:10	reserviert

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Check_Latch](#), [P2_SENT_Check_Latch](#)

Gültig für

[SENT-4 Rev. E](#), [SENT-6 Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
#Define module 4
Rem set SENT channel and appropriate bit pattern
#Define channel 1
#Define ch_pattern Shift_Left(1, channel-1)
#Define msg_count Par_2 'number of received messages
#Define error_count Par_7 'number of CRC errors
#Define lost_msg Par_9 'number of lost messages
Dim senttable[150] As Long At DM_Local
Dim sent_data[100] As Long At DM_Local
Dim msg_no, msg_no_old As Long
Dim i As Long

Init:
Rem Initialize receive buffer for SENT data
For i = 1 To 100
    sent_data[i] = 0
Next i
Processdelay = 60000 '60000=5kHz / 30000=10kHz
Rem initialize module, request latch for channel 1 (once)
P2_SENT_Init(module, senttable)
Do : Until (P2_SENT_Command_Ready(module) = 0)
Par_3 = P2_SENT_Request_Latch(module, ch_pattern)
Do : Until (P2_SENT_Command_Ready(module) = 0)
Par_4 = P2_SENT_Set_CRC_Implementation(module, channel, 1)
msg_count = 0 'number of received messages
msg_no_old = -1 'init msg number
lost_msg = -1 'number of lost messages, skip first check

Event:
Rem read latch status of all channels
Par_1 = P2_SENT_Check_Latch(module)
Rem Any data for SENT channel 1?
If ((Par_1 And ch_pattern) = 0) Then
    Rem read latch data into sent_data[]
    P2_SENT_Get_Latch_Data(senttable, channel, sent_data, 1)
    Rem request latch for channel 1 again
    Do : Until (P2_SENT_Command_Ready(module) = 0)
    Par_3 = P2_SENT_Request_Latch(module, Par_1 XOr ch_pattern)
    Inc msg_count 'number of received messages

    Rem check for serial message
    If (sent_data[7] <> 0) Then
        Rem serial message data are given in sent_data[7]..[11]
    EndIf

    Rem check for lost messages
    msg_no = sent_data[1]
    If ((msg_no - msg_no_old) <> 1) Then Inc lost_msg
    msg_no_old = msg_no 'store for next check
    Rem check for CRC errors (index 6)
    If (sent_data[6] <> 0) Then Inc error_count
EndIf
```

P2_SENT_Set_Output_Mode

P2_SENT_Set_Output_Mode stellt den Ausgabemodus für einen SENT-Kanal auf dem angegebenen Modul ein.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SENT_Set_Output_Mode(module,  
    sent_channel, mode)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>sent_channel</code>	Nummer (1...4) des SENT-Kanals.	LONG
<code>mode</code>	Ausgabemodus des SENT-Kanals: 0: Kontinuierliche Ausgabe (Default). 1: FIFO-Ausgabe.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_Output_Mode** verwenden.

Deaktivieren Sie den SENT-Kanal mit **P2_SENT_Enable_Channel**, bevor sie den Ausgabemodus ändern. Die Ausgabe beginnt, sobald Sie den SENT-Kanal wieder aktivieren.

Der Ausgabemodus legt fest, woher das Modul die SENT-Nachricht bezieht:

- kontinuierliche Ausgabe: Die gerade definierte SENT-Nachricht wird immer wieder gesendet.
Sie setzen den Inhalt der SENT-Nachricht mit den Befehlen
P2_SENT_Set_Fast_Channel1 / _Channel2
P2_SENT_Set_Serial_Message_Pattern
P2_SENT_Set_Reserved_Bits.
- FIFO-Ausgabe: Sie füllen einen FIFO kontinuierlich mit Ausgabewerten, siehe **P2_SENT_Set_Fifo**. Das Modul liest die Werte aus dem FIFO und gibt sie nacheinander aus.
Wenn der FIFO leer läuft, stoppt die Ausgabe und der SENT-Kanal wird automatisch deaktiviert.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Enable_Channel](#), [P2_SENT_Set_Fast_Channel1](#),
[P2_SENT_Set_Fast_Channel2](#), [P2_SENT_Set_Serial_Message_Pattern](#), [P2_SENT_Set_Reserved_Bits](#), [P2_SENT_Set_Fifo](#)

Gültig für

[SENT-4-Out Rev. E](#)

Beispiel

- / -

P2_SENT_Config_Output setzt die Grundeinstellungen für einen SENT-Ausgabekanal auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = ret_val = P2_SENT_Config_Output(module,
    sent_channel, baselock, length, lowticks, crc)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4) des SENT-Kanals.	LONG
baselock	Basistakt (1500...90000) für SENT-Nachrichten, angegeben in ns.	LONG
length	Feste Gesamtlänge (0; 166...922) der SENT-Nachricht, angegeben in baselock -Einheiten. 0: variable Länge ohne Pausenpuls. >0: Gesamtlänge der Nachricht.	LONG
lowticks	Dauer (üblicherweise 4) des Low-Pegels am Anfang von Kalibrierpuls oder Nibble-Puls, angegeben in baselock -Einheiten.	LONG
crc	Berechnungsalgorithmus für die CRC-Prüfsumme: 0: Legacy. 1: Recommended.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Config_Output** verwenden.

Eine SENT-Nachricht ohne Pausenpuls hat eine variable Länge, nämlich 154...270 Basistakte. Mit **length** > 0 können Sie die Gesamtlänge einer Nachricht auf einen festen Wert einstellen. Dies wird durch einen Pausenpuls mit der jeweils nötigen Länge erreicht.

Achten Sie darauf, dass **length** immer um 12 Einheiten größer ist als die tatsächliche Länge der SENT-Nachricht.

Der Berechnungsalgorithmus für die CRC-Prüfsumme gilt sowohl für Signale (fast channels) als auch für serielle Nachrichten.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Command_Ready](#), [P2_SENT_Enable_Channel](#), [P2_SENT_Set_Fast_Channel1](#), [P2_SENT_Set_Fast_Channel2](#), [P2_SENT_Set_Fifo](#)

Gültig für

SENT-4-Out Rev. E

Beispiel

- / -

P2_SENT_Config_Output

P2_SENT_Config_Serial_Messages

P2_SENT_Config_Serial_Messages konfiguriert das Sendeformat für serielle Nachrichten auf einem SENT-Kanal.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Config_Serial_Messages (
    module, sent_channel, format)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4) des SENT-Kanals.	LONG
format	Sendeformat der seriellen Nachricht: 0: keine serielle Nachricht. 1: Format „Standard“, 4 Bit ID, 8 Bit Daten. 2: Format „Enhanced“, 4 Bit ID, 16 Bit Daten. 3: Format „Enhanced“, 8 Bit ID, 12 Bit Daten.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im kontinuierlichen Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Config_Serial_Messages** verwenden.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Command_Ready](#), [P2_SENT_Set_Serial_Message_Pattern](#), [P2_SENT_Set_Serial_Message_Data](#)

Gültig für

SENT-4-Out Rev. E

Beispiel

- / -

P2_SENT_Enable_Channel sperrt mehrere SENT-Kanäle oder gibt sie frei.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SENT_Enable_Channel(module, enable)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
enable	Bitmuster zum Sperren oder Freigeben der SENT-Kanäle: Bit = 0: SENT-Kanal sperren. Bit = 1: SENT-Kanal freigeben.	LONG

Bit-Nr.	31:4	3	2	1	0
SENT-Kanal	–	4	3	2	1

Bemerkungen

Der SENT-Kanal muss erst mit **P2_SENT_Config_Output** konfiguriert werden, bevor Sie ihn zum Senden freigeben.

Wenn Sie einen SENT-Kanal sperren, wird die aktuelle SENT-Nachricht noch vollständig gesendet, danach wird die Ausgabe beendet.

Nach der Freigabe eines Kanals beginnt die Ausgabe sofort. Im kontinuierlichen Modus wird die definierte SENT-Nachricht gesendet, im FIFO-Modus wird die nächste im Fifo vorhandene SENT-Nachricht gesendet.

Beim Sperren eines Kanals wird das Senden einer seriellen Nachricht unterbrochen. Bei der Freigabe des Kanals wird das Senden der seriellen Nachricht beim nächsten Bit fortgeführt.

Im Fifo-Modus wird der SENT-Kanal automatisch gesperrt (und die Ausgabe gestoppt), wenn der FIFO leer läuft. Achten Sie daher darauf, nach dem Freigeben des SENT-Kanals schnell genug neue Daten in den FIFO zu schreiben.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Config_Output](#), [P2_SENT_Invert_Channel](#), [P2_SENT_Set_Serial_Message_Pattern](#)

Gültig für

[SENT-4-Out Rev. E](#)

Beispiel

- / -

P2_SENT_Enable_Channel

P2_SENT_Invert_Channel

P2_SENT_Invert_Channel invertiert alle Signalpegel auf einem SENT-Kanal des angegebenen Moduls.

Syntax

```
#Include ADwinPro_All.inc

P2_SENT_Invert_Channel(module, invert)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
invert	Bitmuster zum Invertieren der Ausgabepegel für alle SENT-Kanäle. Bit = 0: Normaler Ausgabepegel. Bit = 1: Invertierter Ausgabepegel. Die Zuordnung der Bits zu den Kanälen ist in der Tabelle angegeben.	LONG

Bitnr.	31:4	3	2	1	0
SENT-Kanal	—	4	3	2	1

Bemerkungen

Wenn ein SENT-Kanal gesperrt ist, wird im Normalzustand Pegel Low ausgegeben, invertiert aber Pegel High.

Umschalten mit **P2_SENT_Invert_Channel** erlaubt auch bei gesperrtem SENT-Kanal das Ausgeben von wechselnden Pegeln.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Config_Output](#), [P2_SENT_Enable_Channel](#)

Gültig für

[SENT-4-Out Rev. E](#)

Beispiel

- / -

P2_SENT_Set_Reserved_Bits setzt die reservierten Bits im Status-Nibble eines SENT-Kanals auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

P2_SENT_Set_Reserved_Bits(module, sent_channel,
                           value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4) des SENT-Kanals.	LONG
value	Bitmuster (0...11b), das in die reservierten Bits des Status-Nibbles übertragen wird.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im kontinuierlichen Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

Im Status-Nibble sind die Bits 0 und 1 für spezielle Anwendungen reserviert.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Enable_Channel](#), [P2_SENT_Config_Output](#), [P2_SENT_Config_Serial_Messages](#), [P2_SENT_Set_Serial_Message_Pattern](#), [P2_SENT_Set_Serial_Message_Data](#), [P2_SENT_Set_Fast_Channel1](#), [P2_SENT_Set_Fast_Channel2](#)

Gültig für

[SENT-4-Out Rev. E](#)

Beispiel

- / -

P2_SENT_Set_Reserved_Bits

P2_SENT_Set_Fast_Channel1

P2_SENT_Set_Fast_Channel1 setzt den ersten 12 Bit-Wert in der SENT-Nachricht für einen SENT-Kanal auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc  
  
P2_SENT_Set_Fast_Channel1(module, sent_channel,  
                           value)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	<code>LONG</code>
<code>sent_channel</code>	Nummer (1...4) des SENT-Kanals.	<code>LONG</code>
<code>value</code>	Erster 12 Bit-Wert (0...4095) in der SENT-Nachricht.	<code>LONG</code>

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im kontinuierlichen Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

In einer SENT-Nachricht sind zwei 12 Bit-Werte enthalten; sie werden auch als „fast channel signals“ bezeichnet. Der Befehl setzt den ersten der beiden Werte.

Die CRC-Prüfsumme wird automatisch berechnet (Berechnungsalgorithmus siehe **P2_SENT_Config_Output**) und in der SENT-Nachricht gesetzt.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Enable_Channel](#), [P2_SENT_Config_Output](#), [P2_SENT_Config_Serial_Messages](#), [P2_SENT_Set_Serial_Message_Pattern](#), [P2_SENT_Set_Serial_Message_Data](#), [P2_SENT_Set_Fast_Channel2](#), [P2_SENT_Set_Reserved_Bits](#)

Gültig für

[SENT-4-Out Rev. E](#)

Beispiel

- / -

P2_SENT_Set_Fast_Channel2 setzt den zweiten 12 Bit-Wert in der SENT-Nachricht für einen SENT-Kanal auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

P2_SENT_Set_Fast_Channel2(module, sent_channel,
                           value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4) des SENT-Kanals.	LONG
ret_val	Zweiter 12 Bit-Wert (0...4095) in der SENT-Nachricht.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im kontinuierlichen Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

In einer SENT-Nachricht sind zwei 12 Bit-Werte enthalten; sie werden auch als „fast channel signals“ bezeichnet. Der Befehl setzt den zweiten der beiden Werte.

Die CRC-Prüfsumme wird automatisch berechnet (Berechnungsalgorithmus siehe **P2_SENT_Config_Output**) und in der SENT-Nachricht gesetzt.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Enable_Channel](#), [P2_SENT_Config_Output](#), [P2_SENT_Config_Serial_Messages](#), [P2_SENT_Set_Serial_Message_Pattern](#), [P2_SENT_Set_Serial_Message_Data](#), [P2_SENT_Set_Fast_Channel1](#), [P2_SENT_Set_Reserved_Bits](#)

Gültig für

[SENT-4-Out Rev. E](#)

Beispiel

- / -

P2_SENT_Set_Fast_Channel2

P2_SENT_Set_Serial_Message_Pattern

P2_SENT_Set_Serial_Message_Pattern definiert einen Nachrichtensatz für serielle Nachrichten auf einem SENT-Kanal.

Syntax

```
#Include ADwinPro_All.inc

REM define SENT settings array
P2_SENT_Init(module, sent_datatable[])

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Set_Serial_Message_Pattern(module,
    sent_datatable[], sent_channel, sm_id_array[],
    sm_value_array[], sm_len)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_datatable[]	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und SENT-Modul aufnimmt.	ARRAY LONG
sent_channel	Nummer (1...4) des SENT-Kanals.	LONG
sm_id_array[]	Feld mit bis zu 32 Kennungen von seriellen Nachrichten.	ARRAY LONG
sm_value_array[]	Feld mit bis zu 32 Datenwerten für serielle Nachrichten zu den Kennungen in sm_id_array[] .	ARRAY LONG
sm_len	Anzahl (1...32) der verwendeten seriellen Nachrichten.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im kontinuierlichen Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_Serial_Message_Pattern** verwenden.

Sperren Sie den Ausgabekanal mit **P2_SENT_Enable_Channel**, bevor Sie den Nachrichtensatz festlegen.

Der Wertebereich der Kennungen und Datenwerte hängt vom eingestellten Sendeformat der seriellen Nachricht ab. Sie stellen das Sendeformat (standard, enhanced) mit **P2_SENT_Config_Serial_Messages** ein.

Der Nachrichtensatz in **sm_id_array** und in **sm_value_array** wird der Reihe nach in den SENT-Nachrichten übertragen. Wenn das Ende der Liste (**sm_len**) erreicht ist, beginnt die Übertragung wieder von vorn.

Sie können einzelne Datenwerte nachträglich mit **P2_SENT_Set_Serial_Message_Data** ändern. Um Kennungen zu ändern, müssen Sie den ganzen Nachrichtensatz neu definieren.

Die zugehörige CRC-Prüfsumme wird automatisch berechnet und übertragen.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Command_Ready](#), [P2_SENT_Enable_Channel](#), [P2_SENT_Config_Output](#), [P2_SENT_Config_Serial_Messages](#), [P2_SENT_Set_Serial_Message_Data](#), [P2_SENT_Set_Fast_Channel1](#), [P2_SENT_Set_Fast_Channel2](#), [P2_SENT_Set_Reserved_Bits](#)

Gültig für

SENT-4-Out Rev. E

Beispiel

- / -

P2_SENT_Set_Serial_Message_Data ändert einen Datenwert im Nachrichtensatz für serielle Nachrichten.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Set_Serial_Message_Data(module,
    sent_channel, sm_id_idx, sm_value)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_channel	Nummer (1...4) des SENT-Kanals.	LONG
sm_id_idx	Index für das Feld mit dem Nachrichtensatz.	LONG
sm_value	Datenwert für eine serielle Nachricht.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im kontinuierlichen Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_Serial_Message_Data** verwenden.

Der Befehl ändert einen Datenwert im Nachrichtensatz, den Sie mit **P2_SENT_Set_Serial_Message_Pattern** eingestellt haben, und zwar an der Position **sm_id_idx**. Sie können nur Datenwerte ändern, nicht aber die eingestellten Kennungen.

Die zugehörige CRC-Prüfsumme wird automatisch berechnet und übertragen.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Command_Ready](#), [P2_SENT_Config_Output](#), [P2_SENT_Config_Serial_Messages](#), [P2_SENT_Set_Serial_Message_Pattern](#), [P2_SENT_Set_Fast_Channel1](#), [P2_SENT_Set_Fast_Channel2](#), [P2_SENT_Set_Reserved_Bits](#)

Gültig für

SENT-4-Out Rev. E

Beispiel

- / -

P2_SENT_Set_Serial_Message_Data

P2_SENT_Fifo_Empty

P2_SENT_Fifo_Empty ermittelt die Anzahl der freien Elemente im Ausgabe-FIFO eines SENT-Kanals.

Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_SENT_Fifo_Empty(module, sent_channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
sent_channel	Nummer (1...4) des SENT-Kanals.	LONG
ret_val	Anzahl (0...400) der belegten Elemente im Ausgabe-FIFO des SENT-Kanals.	__LONG

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im FIFO-Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Set_Output_Mode](#), [P2_SENT_Fifo_Clear](#), [P2_SENT_Set_Fifo](#)

Gültig für

[SENT-4-Out Rev. E](#)

Beispiel

- / -

P2_SENT_Fifo_Clear initialisiert den Schreib- und Lese-Zeiger des Ausgabe-FIFOs eines SENT-Kanals.

Syntax

```
#Include ADwinPro_All.inc

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Fifo_Clear(module, sent_channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>sent_channel</code>	Nummer (1...4) des SENT-Kanals.	LONG
<code>ret_val</code>	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im FIFO-Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Fifo_Clear** verwenden.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Command_Ready](#), [P2_SENT_Set_Output_Mode](#), [P2_SENT_Fifo_Empty](#), [P2_SENT_Set_Fifo](#)

Gültig für

[SENT-4-Out Rev. E](#)

Beispiel

- / -

P2_SENT_Fifo_Clear

P2_SENT_Set_Fifo

P2_SENT_Set_Fifo schreibt neue Daten in den Ausgabe-Fifo eines SENT-Kanals auf dem angegebenen Modul.

Syntax

```
#Include ADwinPro_All.inc

REM define SENT settings array
Dim sent_datatable[150] As Long

Do : Until (P2_SENT_Command_Ready(module) = 0)

ret_val = P2_SENT_Set_Fifo(module, sent_datatable[],
    sent_channel, array[], array_count)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
sent_datatable[]	Feld, das Einstellungen für die Datenübertragung zwischen ADwin CPU und SENT-Modul enthält.	ARRAY LONG
sent_channel	Nummer (1...4) des SENT-Kanals.	LONG
array[]	Feld, aus dem die Ausgabewerte gelesen werden. Das Feld darf höchstens 100 Elemente groß sein.	ARRAY LONG
array_count	Anzahl der übertragenen Ausgabewerte.	LONG
ret_val	Status der Befehlsverarbeitung: 0: Befehl wurde erfolgreich ausgeführt.	LONG

Bemerkungen

Der Befehl ist nur sinnvoll einsetzbar, wenn das Modul im FIFO-Modus arbeitet, siehe **P2_SENT_Set_Output_Mode**.

Führen Sie einmalig **P2_SENT_Init** aus, bevor Sie den Befehl **P2_SENT_Get_Serial_Message_Array** verwenden.

Prüfen Sie erst mit **P2_SENT_Fifo_Empty**, ob genügend viel Platz im Fifo ist, bevor Sie neue Daten schreiben.

Prüfen Sie erst mit **P2_SENT_Command_Ready**, ob die SENT-Schnittstelle bereit ist zum Verarbeiten des nächsten Befehls, bevor Sie **P2_SENT_Set_Fifo** verwenden.

Jedes Feldelement in **array[]** enthält eine SENT-Nachricht. Sie müssen die 32 Bit der SENT-Nachrichten jeweils selbst zusammensetzen. Die Bits werden in folgender Reihenfolge erwartet:

Bitnr.	31:28	27:16	15:4	3:0
Inhalt	Prüfsumme	Signal 2 Bits 11:0	Signal 1, Bits 11:0	Nibble Status + Kommunikation

Wenn der FIFO leer läuft, wird die Ausgabe gestoppt und der SENT-Kanal gesperrt. Achten Sie daher darauf, nach dem Freigeben des SENT-Kanals regelmäßig und schnell genug neue Daten in den FIFO zu schreiben.

Siehe auch

[P2_SENT_Init](#), [P2_SENT_Command_Ready](#), [P2_SENT_Set_Output_Mode](#), [P2_SENT_Fifo_Empty](#), [P2_SENT_Fifo_Clear](#)

Gültig für

SENT-4-Out Rev. E

Beispiel

- / -

3.21 Pro II: SPI-Schnittstelle

Dieser Abschnitt enthält Befehle zum Ansprechen von SPI-Schnittstellen auf *ADwin-Pro II*.

- [P2_SPI_Mode](#) (Seite 434)
- [P2_SPI_Config](#) (Seite 435)
- [P2_SPI_Master_Config](#) (Seite 437)
- [P2_SPI_Master_Set_Value32](#) (Seite 441)
- [P2_SPI_Master_Set_Value64](#) (Seite 442)
- [P2_SPI_Master_Start](#) (Seite 443)
- [P2_SPI_Master_Status](#) (Seite 444)
- [P2_SPI_Master_Get_Value32](#) (Seite 445)
- [P2_SPI_Master_Get_Value64](#) (Seite 446)
- [P2_SPI_Master_Get_Static_Input](#) (Seite 447)
- [P2_SPI_Master_Set_Clk_Wait](#) (Seite 448)
- [P2_SPI_Slave_Config](#) (Seite 450)
- [P2_SPI_Slave_OutFifo_Write](#) (Seite 451)
- [P2_SPI_Slave_OutFifo_Empty](#) (Seite 453)
- [P2_SPI_Slave_InFifo_Full](#) (Seite 454)
- [P2_SPI_Slave_InFifo_Read](#) (Seite 455)
- [P2_SPI_Slave_Clear_Fifo](#) (Seite 457)

Für das Modul SPI-2-T Rev. E sind weitere Befehle verfügbar. Die Befehlsübersicht nach Modulen (Anhang A.2) zeigt, welche Befehle für einen Modultyp anwendbar sind.

P2_SPI_Mode

P2_SPI_Mode legt den Betriebsmodus (SPI-Master / SPI-Slave / Digitalmodul) des angegebenen Moduls fest.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SPI_Mode(module, mode)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
mode	Kennziffer für den Betriebsmodus des Moduls: 1: Digitalmodul; entspricht dem Modul Pro II-DIO-32-TiCo, jedoch ohne DRAM. 2: SPI-Modul mit 2 Slave-Schnittstellen. 3: SPI-Modul; Kanal 1 ist Master-Schnittstelle, Kanal 2 ist Slave-Schnittstelle. 4: SPI-Modul mit 2 Master-Schnittstellen. 5: SPI-Modul mit 2 Slave-Schnittstellen, gemeinsamer CLK-Eingang <i>SCLK1</i> .	LONG

Bemerkungen

Die Pinbelegung ist für jeden Betriebsmodus unterschiedlich. In den SPI-Modi (**mode** = 2...5) sind einige Pins, die nicht für SPI verwendet werden, als einfache Digitalkanäle verwendbar. Pinbelegung siehe Hardware-Handbuch.

Es ist möglich, im laufenden Betrieb den Betriebsmodus umzuschalten.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Master_Set_Clk_Wait](#), [P2_SPI_Slave_Config](#), [P2_DigProg](#), [P2_DigProg_Bits](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
```

```
Rem Betriebsmodus auf 2 SPI-Slave-Schnittstellen einstellen
```

```
P2_SPI_Mode(1, 2)
```

P2_SPI_Config legt Eigenschaften einer SPI-Schnittstelle des angegebenen Moduls fest, für Master wie für Slave.

Syntax

```
#Include ADwinPro_All.inc

P2_SPI_Config(module, channel, mode, bitlength,
              data_order, ss_select)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Kanals.	LONG
mode	Modus des Taktsignals des SPI-Masters mit CPOL (clock polarity) und CPHA (clock phase): 0: CPOL = 0, CPHA = 0 1: CPOL = 0, CPHA = 1 2: CPOL = 1, CPHA = 0 3: CPOL = 1, CPHA = 1	LONG
bitlength	Anzahl (1...64) der übertragenen Bits in einer SPI-Nachricht.	LONG
data_order	Bitreihenfolge bei der Übertragung einer SPI-Nachricht: 0: höchstwertiges Bit (MSB) zuerst 1: niedrigstwertiges Bit (LSB) zuerst	LONG
ss_select	Pegel, mit dem die Slave-Select-Leitungen als aktiv gelten: 0: Pegel low aktiv 1: Pegel high aktiv	LONG

Bemerkungen

Beim Taktsignal bestimmt CPOL das Ruhesignal (0: low; 1: high). CPHA legt fest, bei welcher Flanke die Daten übernommen werden (0: erste; 1: zweite).

Wenn das Slave-Select-Signal automatisch gesetzt wird (siehe **P2_SPI_Master_Config**), stellen Sie sicher, dass die Slave-Select-Leitung des SPI-Masters inaktiv ist (Abfrage mit **P2_SPI_Master_Status**), bevor Sie den SPI-Master neu konfigurieren. Beim Konfigurieren des SPI-Masters entstehen sonst Spikes, die von angeschlossenen Slaves falsch interpretiert werden und damit die Datenübertragung stören können.

Die Taktsignale liegen auf den Pins SCLK1 / SCLK2, Pinbelegung siehe Hardware-Handbuch. Wenn bei **SPI_Mode** der Parameter **mode=5** gesetzt ist, verwenden die SPI-Slaves SCLK1 als einen gemeinsamen Takteingang; SCLK2 hat in diesem Fall keine Funktion.

Die Bitlänge bezieht sich gleichermaßen auf die Leitungen SCLK, DATAIN und DATAOUT. Beachten Sie, dass **ADbasic**-Variablen eine Bitlänge von 32 Bit haben, daher müssen SPI-Nachrichten mit 33...64 Bit Länge auf 2 Variablen aufgeteilt gespeichert werden.

Siehe auch

[P2_SPI_Mode](#), [P2_SPI_Master_Config](#), [P2_SPI_Master_Set_Clk_Wait](#), [P2_SPI_Slave_Config](#), [P2_SPI_Master_Start](#), [P2_SPI_Master_Status](#)

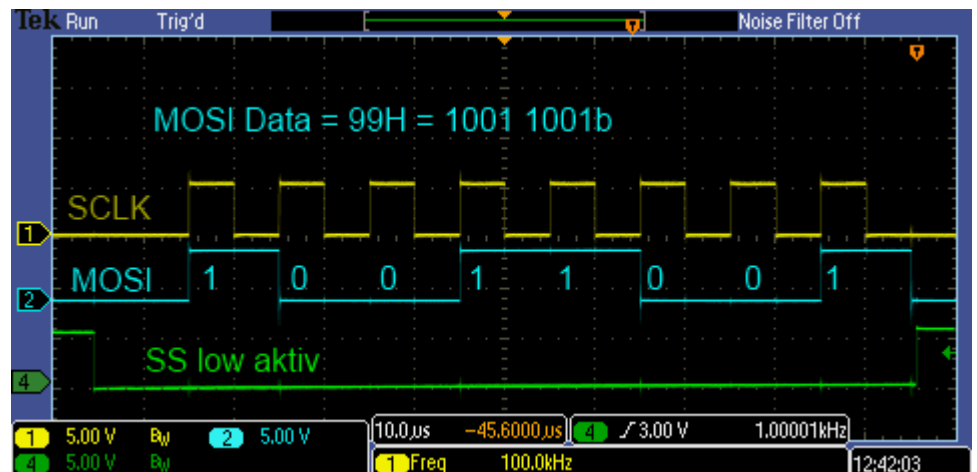
Gültig für

SPI-2-D Rev. E, SPI-2-T Rev. E

P2_SPI_Config

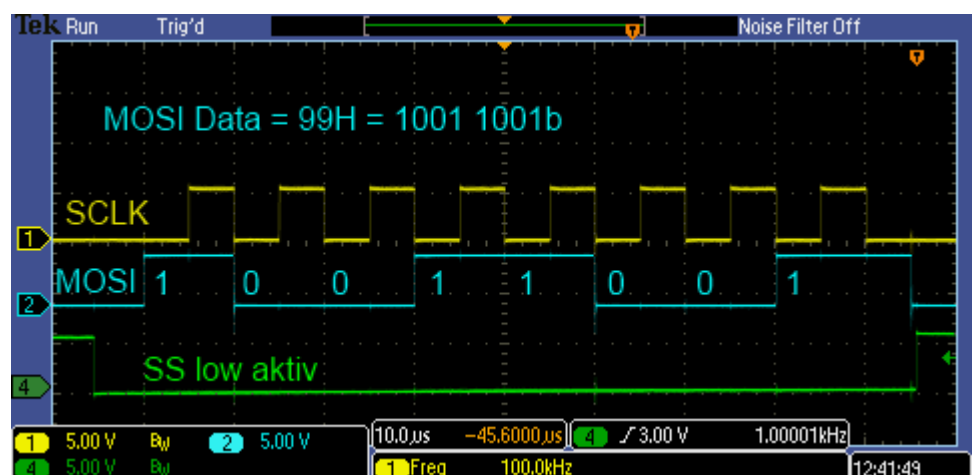
Beispiel

```
#Include ADwinPro_All.inc
#Define module 1
Init:
    P2_SPI_Mode(module, 4)          '2 Master-Schnittstellen
    Rem Master 1 konfigurieren:
    Rem CPOL = 0, CPHA = 1; Nachrichtenlänge 8 Bit
    Rem MSB zuerst; Slave-Select aktiv bei High
    P2_SPI_Config(module, 1, 1, 8, 0, 0)
    Rem Taktfrequenz 1 MHz etc. einstellen
    P2_SPI_Master_Config(module, 1, 250, 75, 1, 0)
```



MOSI-Ausgabe bei steigender Flanke von SCLK mit 8 Bit-Wert 99h

```
#Include ADwinPro_All.inc
#Define module 1
Init:
    P2_SPI_Mode(module, 4)          '2 Master-Schnittstellen
    Rem Master 1 konfigurieren:
    Rem CPOL = 1, CPHA = 0; Nachrichtenlänge 8 Bit
    Rem MSB zuerst; Slave-Select aktiv bei High
    P2_SPI_Config(module, 1, 2, 8, 0, 0)
    Rem Taktfrequenz 1 MHz etc. einstellen
    P2_SPI_Master_Config(module, 1, 250, 75, 1, 0)
```



MOSI-Ausgabe bei fallender Flanke von SCLK

P2_SPI_Master_Config legt (zusätzliche) Eigenschaften einer SPI-Master-Schnittstelle des Moduls fest.

- **clk_factor** setzt die Frequenz des SPI-Taktsignals f_{SPI} .
- **miso_delay** verzögert das Einlesen von SPI-Nachrichten (relativ zum MOSI-Signal).
- **ss_manual** legt fest, ob Slave-Select automatisch oder manuell gesetzt wird.
- **ss_time** verlängert das (automatisch aktivierte) Slave-Select-Signal bezogen auf das Taktsignal.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SPI_Master_Config(module, channel, clk_factor,  
    miso_delay, ss_manual, ss_time)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) der Schnittstelle, die als SPI-Master konfiguriert ist.	LONG
clk_factor	Faktor (2...2500) zur Berechnung der Taktfrequenz f_{SPI} aus der Grundfrequenz des Timers: $f_{\text{SPI}} = 25 \text{ MHz} / \text{clk_factor}$	LONG
miso_delay	Verzögerungszeit beim Einlesen von SPI-Nachrichten in Einheiten (0...255) von 20 ns. Damit sind Zeiten im Bereich von 0...5.1µs einstellbar.	LONG
ss_manual	Einstellung des Slave-Select-Betriebsmodus des SPI-Masters: 0: Slave-Select-Leitung automatisch aktivieren. 1: Slave-Select-Leitung manuell aktivieren.	LONG
ss_time	Nur wenn ss_manual =0, also bei automatischer Aktivierung: Zusätzlicher Zeitabstand für das Slave-Select-Signal in Einheiten (0...255) von 20 ns. Voreinstellung ist 25 (= 0,5 µs). Es sind Zusatzzeiten von 0...5.1µs einstellbar. Der Zeitabstand T zwischen Slave-Select-Signal und Taktsignal ergibt sich zu: $T = 0,5 / f_{\text{SPI}} + \text{ss_time} \times 20 \text{ ns}$	LONG

Bemerkungen

Mit dem Faktor **clk_factor** legen Sie die Frequenz f_{SPI} des Signals, mit der die MISO- und MOSI-Signale ausgegeben und eingelesen werden, an den Pins SCLK1 oder SCLK2 fest. Pinbelegung siehe Hardware-Handbuch.

Einstellbar sind Taktfrequenzen im Bereich von 100Hz 12,5MHz.

Eine neu eingestellte Frequenz wird erst verwendet, nachdem der Befehl **SPI_Master_Start** aufgerufen wurde.

Eingehende MISO-Daten werden um die eingestellte Verzögerungszeit **miso_delay** versetzt zum MOSI-Signal eingelesen. Somit lassen sich Signallaufzeiten kompensieren, die z.B. durch Signalkonditionierung entstehen.

Die eingestellte Verzögerungszeit bleibt solange bestehen, bis eine neue gesetzt wird. Mit dem Wert 0 wird die Verzögerungszeit deaktiviert.

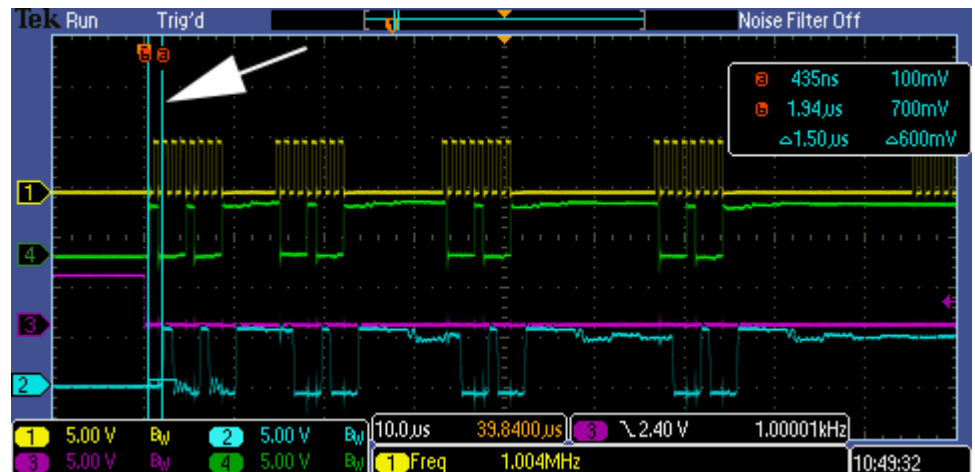
P2_SPI_Master_Config

Taktfrequenz

Verzögertes Einlesen

Slave-Select-Modus

Verlängertes Slave-Select-Signal



MISO-Daten (Linie 2) um 1,5 μ s (`miso_delay = 75`)
zeitverzögert zu MOSI-Daten (Linie 4) einlesen

Bei `ss_manual=0` wird die jeweilige Slave-Select-Leitung SS out des Masters beim Start der Datenübertragung mit **P2_SPI_Master_Start** automatisch aktiviert und anschließend deaktiviert. Das ist z.B. sinnvoll, wenn am SPI-Bus nur ein Slave-Teilnehmer vorhanden ist.

Bei manueller Aktivierung (`ss_manual=1`) aktivieren Sie die Slave-Select-Leitungen einzeln mit einem Befehl **P2_Digout_...**, bevor die Datenübertragung mit **P2_SPI_Master_Start** gestartet wird.

Alle verfügbaren Digitalausgänge können als manuell betriebene Slave-Select-Leitungen verwendet werden, darunter auch SS1 out oder SS2 out (Pinbelegung siehe Hardware-Handbuch).

Beachten Sie: Digitalkanäle sind nach dem Einschalten zunächst als Eingänge konfiguriert und werden mit **P2_DigProg** als Ausgänge konfiguriert.

Welcher Pegel die Slave-Select-Leitungen aktiv schaltet, legen Sie mit **P2_SPI_Config** fest.

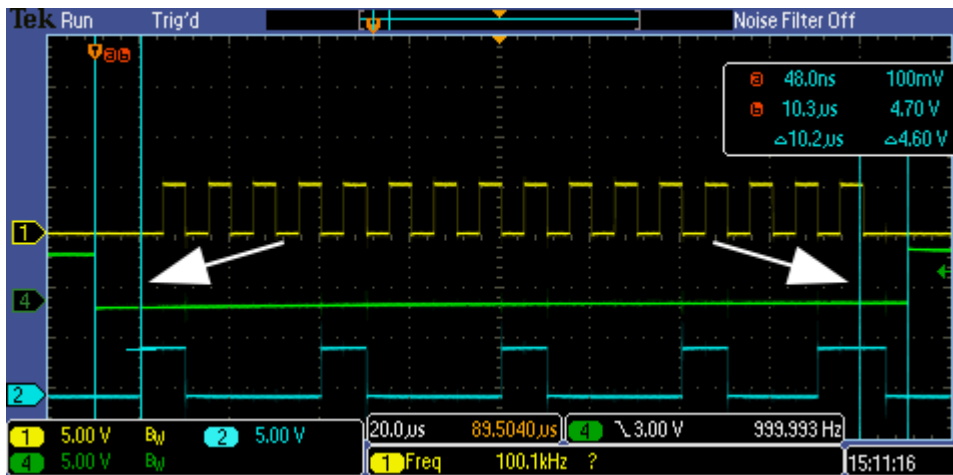
Diese Option ist nur einsetzbar, wenn mit `ss_manual=0` die automatische Aktivierung der Slave-Select-Leitung festgelegt ist.

Grundsätzlich beginnt das Slave-Select-Signal um einen definierten Zeitabstand vor dem Start des Taktsignals und endet um den gleichen Zeitabstand nach dem Ende des Taktsignals. Der Zeitabstand ist eine halbe Taktperiode plus die Zusatzzeit, die sich aus der Einstellung von `ss_time` ergibt.

Diesen Zeitabstand verändern Sie mit dem Parameter `ss_time`. Die Verlängerungszeit wirkt gleichermaßen vor dem Taktsignal – was einem zeitverzögerten Start des Taktsignals entspricht – wie auch danach.

Die eingestellte Verlängerung bleibt solange bestehen, bis eine neue konfiguriert wird.

Beispiel: `f_SPI = 100 kHz`, `ss_time = 250`
 $T = 0,5 / 100 \text{ kHz} + 250 \times 20\text{ns} = 5\mu\text{s} + 5\mu\text{s} = 10\mu\text{s}$



Signal-Select-Signal (Linie 4) beginnt 10 μ s früher und endet 10 μ s später als das Taktsignal (Linie 1)

Siehe auch

[P2_SPI_Mode](#), [P2_SPI_Config](#), [P2_SPI_Master_Set_Clk_Wait](#), [P2_SPI_Slave_Config](#), [P2_SPI_Master_Set_Value32](#), [P2_SPI_Master_Get_Value32](#), [P2_SPI_Master_Start](#), [P2_SPI_Master_Status](#), [P2_SPI_Master_Get_Static_Input](#)

[P2_Digout_Long](#), [P2_Digout](#), [P2_Digout_Bits](#), [P2_Digout_Reset](#), [P2_Digout_Set](#)

Gültig für

SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

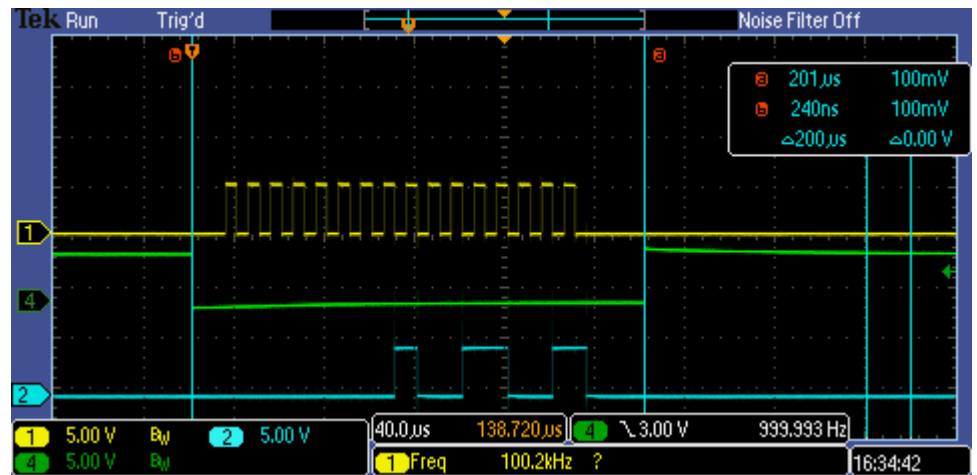
```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 1
```

Init:

```
P2_SPI_Mode(mod_no, 4)      '2 Master-Schnittstellen
Rem Master konfigurieren:
Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 16 Bit
Rem MSB zuerst; Slave-Select aktiv bei Low
P2_SPI_Config(mod_no, master_no, 0, 16, 0, 0)
Rem clk factor 250: Taktfrequenz 100 kHz
Rem miso_delay 75: MISO-Signal um 1.5 μs verzögert einlesen
Rem ss_time 250: Slave-Select-Signal auf 10 μs verlängern
Rem Slave-Select-Leitung manuell aktivieren
P2_SPI_Master_Config(mod_no, master_no, 250, 75, 1, 1)
Rem SPI-Nachricht zur Ausgabe bereitstellen
P2_SPI_Master_Set_Value32(mod_no, master_no, 99h)
```

Event:

```
Rem Slave-Select via DIO25 aktivieren, Übertragung starten
P2_Digout(mod_no, 25, 0)
P2_SPI_Master_Start(mod_no, master_no)
Rem 200 μs warten;
Rem Signaldauer T=16Bits*10 μs=160 μs + Wartezeit
P2_Sleep(20000)
Rem Slave-Select-Leitung DIO25 des Masters deaktivieren
P2_Digout(mod_no, 25, 1)
```



Slave-Select (Linie 4) ist für 200µs aktiv

P2_SPI_Master_Set_Value32 stellt eine SPI-Nachricht bis 32 Bit Länge am Master-Ausgang zur Ausgabe bereit.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SPI_Master_Set_Value32 (module, channel, mosi_data)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Masters.	LONG
mosi_data	Zu sendende SPI-Nachricht mit bis zu 32 Bit.	LONG

Bemerkungen

Die Pinbelegung der Pins DataOut zur Ausgabe der MOSI-Datensignale finden Sie im Hardware-Handbuch.

Die SPI-Nachricht wird nur zur Ausgabe bereit gestellt. Sie starten die Datenübertragung mit dem Befehl **P2_SPI_Master_Start**.

Mit **P2_SPI_Master_Set_Value32** kann in jedem Fall nur das untere Wort (Bits 31:0) einer SPI-Nachricht bereit gestellt werden, auch wenn die SPI-Bitlänge größer ist als 32 Bit. Um auch das obere Wort bereit zu stellen, verwenden Sie **P2_SPI_Master_Set_Value64**.

Sie stellen mit **P2_SPI_Config** mehrere Parameter für die Datenübertragung ein wie die Bitreihenfolge und die Nachrichtenlänge.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Master_Set_Value64](#), [P2_SPI_Master_Start](#), [P2_SPI_Master_Status](#), [P2_SPI_Master_Get_Value32](#)

Gültig für

SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
```

```
#Define mod_no 4
```

```
#Define master_no 2
```

Init:

```
P2_SPI_Mode(mod_no, 4)          '2 Master-Schnittstellen
Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 16 Bit
P2_SPI_Config(mod_no, master_no, 0, 16, 0, 0)
Rem Taktfrequenz 1 MHz etc. einstellen
P2_SPI_Master_Config(mod_no, master_no, 250, 75, 1, 0)
```

Event:

```
Rem SPI-Nachricht zur Ausgabe bereitstellen und
Rem Datenübertragung starten
P2_SPI_Master_Set_Value32(mod_no, master_no, 12345678h)
P2_SPI_Master_Start(mod_no, master_no)
```

P2_SPI_Master_Set_Value32

P2_SPI_Master_Set_Value64

P2_SPI_Master_Set_Value64 stellt eine SPI-Nachricht bis 64 Bit Länge am Master-Ausgang zur Ausgabe bereit.

Syntax

```
#Include ADwinPro_All.inc

P2_SPI_Master_Set_Value64(module, channel,
                           mosi_high, mosi_low)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Masters.	LONG
mosi_data	Obere 32 Bit der zu sendenden SPI-Nachricht.	LONG
mosi_low	Untere 32 Bit der zu sendenden SPI-Nachricht.	LONG

Bemerkungen

Die Pins DataOut zur Ausgabe der MOSI-Datensignale finden Sie im Hardware-Handbuch.

Die SPI-Nachricht wird nur zur Ausgabe bereit gestellt. Sie starten die Datenübertragung mit dem Befehl **P2_SPI_Master_Start**.

Wenn die SPI-Bitlänge nicht größer ist als 32 Bit, ist **P2_SPI_Master_Set_Value32** der schnellere Befehl.

Sie stellen mit **P2_SPI_Config** mehrere Parameter für die Datenübertragung ein wie die Bitreihenfolge und die Nachrichtenlänge.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Master_Set_Value32](#), [P2_SPI_Master_Start](#), [P2_SPI_Master_Status](#), [P2_SPI_Master_Get_Value32](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 2

Init:
    P2_SPI_Mode(mod_no, 4)           '2 Master-Schnittstellen
    Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 64 Bit
    P2_SPI_Config(mod_no, master_no, 0, 64, 0, 0)
    Rem Taktfrequenz 1 MHz etc. einstellen
    P2_SPI_Master_Config(mod_no, master_no, 250, 75, 1, 0)

Event:
    Rem SPI-Nachricht 64 Bit zur Ausgabe bereitstellen
    P2_SPI_Master_Set_Value64(mod_no, master_no, 0F678h, 5678h)
    Rem Datenübertragung starten
    P2_SPI_Master_Start(mod_no, master_no)
```

P2_SPI_Master_Start startet die Datenübertragung über den SPI-Bus. Falls konfiguriert, wird die Slave-Select-Leitung des SPI-Masters automatisch aktiviert.

Syntax

```
#Include ADwinPro_All.inc

P2_SPI_Master_Start(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Masters.	LONG

Bemerkungen

Die Datenübertragung arbeitet grundsätzlich in beide Richtungen, d.h. der Master sendet die SPI-Nachricht, die zuletzt bereit gestellt wurde (MOSI). Der Slave antwortet in der Regel noch während der gleichen Datenübertragung (MISO).

Das Ende der Datenübertragung können Sie mit **P2_SPI_Master_Status** ermitteln.

Mit **P2_SPI_Master_Config** legen Sie fest, ob die Slave-Select-Leitungen eines SPI-Masters manuell oder automatisch gesetzt werden. Im manuellen Modus müssen Sie die Slave-Select-Leitungen vor der Datenübertragung aktivieren und danach wieder deaktivieren. Im automatischen Modus wird die Leitung SS out vom SPI-Master gesetzt.

Mit **P2_SPI_Config** legen Sie fest, welcher Pegel die Slave-Select-Leitungen aktiv schaltet.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Master_Set_Value32](#), [P2_SPI_Master_Set_Value64](#), [P2_SPI_Master_Status](#), [P2_SPI_Master_Get_Value32](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 2

Init:
    P2_SPI_Mode(mod_no, 4)          '2 Master-Schnittstellen
    Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 64 Bit
    P2_SPI_Config(mod_no, master_no, 0, 64, 0, 0)
    Rem Taktfrequenz 100 kHz
    Rem MISO-Signal um 1.5µs verzögert einlesen
    Rem Slave-Select-Signal um 5µs auf 10µs verlängern
    Rem Slave-Select-Leitung automatisch aktivieren
    P2_SPI_Master_Config(mod_no, master_no, 250, 75, 1, 0)

Event:
    Rem SPI-Nachricht zur Ausgabe bereitstellen
    P2_SPI_Master_Set_Value64(mod_no, master_no, 0F678h, 5678h)
    Rem Slave-Select-Ausgang SS out automatisch aktivieren und
    Rem Datenübertragung starten
    P2_SPI_Master_Start(mod_no, master_no)
```

P2_SPI_Master_Start

P2_SPI_Master_Status

P2_SPI_Master_Status gibt zurück, ob die Datenübertragung des SPI-Masters aktiv oder inaktiv ist.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SPI_Master_Status(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Masters.	LONG
ret_val	Status des SPI-Masters: 0: Datenübertragung ist beendet; im Automatik-Modus auch: Slave-Select-Leitung ist inaktiv. 1: Datenübertragung dauert noch an; im Automatik-Modus auch: Slave-Select-Leitung ist aktiv.	LONG

Bemerkungen

Solange die Datenübertragung andauert (Rückgabewert = 1), darf kein weiterer SPI-Befehl ausgeführt werden.

Wenn mit **P2_SPI_Master_Config** das automatische Setzen der Slave-Select-Leitung eingestellt ist, gibt **P2_SPI_Master_Status** auch an, ob die Slave-Select-Leitung SS out aktiv oder inaktiv ist. Mit **P2_SPI_Config** legen Sie fest, welcher Pegel die Slave-Select-Leitungen aktiv schaltet.

Siehe auch

[P2_SPI_Master_Set_Value32](#), [P2_SPI_Master_Set_Value64](#), [P2_SPI_Master_Start](#), [P2_SPI_Master_Get_Value32](#), [P2_SPI_Master_Get_Value64](#)

Gültig für

SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 2

Init:
    P2_SPI_Mode(mod_no, 4)          '2 Master-Schnittstellen
    Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 64 Bit
    P2_SPI_Config(mod_no, master_no, 0, 64, 0, 0)
    Rem Taktfrequenz 100 kHz
    Rem MISO-Signal um 1.5µs verzögert einlesen
    Rem Slave-Select-Signal um 5µs auf 10µs verlängern
    Rem Slave-Select-Leitung automatisch aktivieren
    P2_SPI_Master_Config(mod_no, master_no, 250, 75, 1, 0)

    Rem SPI-Nachricht 64 Bit zur Ausgabe bereitstellen
    P2_SPI_Master_Set_Value64(mod_no, master_no, 0F678h, 5678h)

Event:
    Rem Slave-Select-Ausgang SS out automatisch aktivieren und
    Rem Datenübertragung starten
    P2_SPI_Master_Start(mod_no, master_no)
    Rem Status des Masters abfragen, bis er nicht mehr aktiv ist
    Do
        Par_80 = P2_SPI_Master_Status(mod_no, master_no)
    Until (Par_80 <> 1)
```


P2_SPI_Master_Get_Value32 liest eine (bereits empfangene) SPI-Nachricht mit bis zu 32 Bit aus dem Eingangsregister der SPI-Schnittstelle.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SPI_Master_Get_Value32(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Masters.	LONG
ret_val	Empfangene SPI-Nachricht mit bis zu 32 Bit.	LONG

Bemerkungen

Die Pins **DataIn** für den Eingang der MISO-Datensignale finden Sie im Hardware-Handbuch.

Die Anzahl der übertragenen Bits in der SPI-Nachricht stellen Sie mit **P2_SPI_Config** ein. Sind weniger als 32 Bit eingestellt, werden ungenutzte Bits in **ret_val** auf Null gesetzt.

Sie können die SPI-Nachricht erst lesen, wenn die Datenübertragung beendet ist. Den Status fragen Sie mit **SPI_Master_Status** ab.

Mit **P2_SPI_Master_Get_Value32** kann in jedem Fall nur das untere Wort (Bits 31:0) einer SPI-Nachricht gelesen werden, auch wenn die SPI-Bitlänge größer ist als 32 Bit. Um auch das obere Wort zu lesen, verwenden Sie **P2_SPI_Master_Get_Value64**.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Master_Set_Value32](#), [P2_SPI_Master_Status](#), [P2_SPI_Master_Get_Value64](#)

Gültig für

SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 1
```

Init:

```
P2_SPI_Mode(mod_no, 4)          '2 Master-Schnittstellen
Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 18 Bit
P2_SPI_Config(mod_no, master_no, 0, 18, 0, 0)
Rem Taktfrequenz 1 MHz etc. einstellen
P2_SPI_Master_Config(mod_no, master_no, 250, 75, 1, 0)
```

Event:

```
Rem Status des Masters abfragen, bis er nicht mehr aktiv ist
Do
    Par_80 = P2_SPI_Master_Status(mod_no, master_no)
Until(Par_80 <> 1)
Rem SPI-Nachricht einlesen
Rem Par_13 enthält einen 32 Bit-Wert, in dem die Bits 0..18
Rem die SPI-Nachricht enthalten, die Bits 19..31 sind 0.
Par_13 = P2_SPI_Master_Get_Value32(mod_no, master_no)
```

P2_SPI_Master_Get_Value32

P2_SPI_Master_Get_Value64

P2_SPI_Master_Get_Value64 liest eine (bereits empfangene) SPI-Nachricht mit bis zu 64 Bit aus dem Eingangsregister der SPI-Schnittstelle.

Syntax

```
#Include ADwinPro_All.inc

P2_SPI_Master_Get_Value64(module, channel,
                           mosi_high, mosi_low)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Masters.	LONG
mosi_high	Obere 32 Bit der empfangenen SPI-Nachricht.	LONG
mosi_low	Untere 32 Bit der empfangenen SPI-Nachricht.	LONG

Bemerkungen

Die Pins DataIn für den Eingang der MISO-Datensignale finden Sie im Hardware-Handbuch.

Die Anzahl der übertragenen Bits in der SPI-Nachricht stellen Sie mit **P2_SPI_Config** ein. Sind weniger als 64 Bit eingestellt, werden ungenutzte Bits in **miso_high** (und ggf. auch in **miso_low**) auf Null gesetzt.

Sie können die SPI-Nachricht erst lesen, wenn die Datenübertragung beendet ist. Den Status fragen Sie mit **SPI_Master_Status** ab.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Master_Set_Value64](#), [P2_SPI_Master_Status](#), [P2_SPI_Master_Get_Value32](#)

Gültig für

SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 1

Init:
    P2_SPI_Mode(mod_no, 4)          '2 Master-Schnittstellen
    Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 48 Bit
    P2_SPI_Config(mod_no, master_no, 0, 48, 0, 0)
    Rem Taktfrequenz 1 MHz etc. einstellen
    P2_SPI_Master_Config(mod_no, master_no, 250, 75, 1, 0)

Event:
    Rem Status des Masters abfragen, bis er nicht mehr aktiv ist
    Do
        Par_80 = P2_SPI_Master_Status(mod_no, master_no)
    Until (Par_80 <> 1)
    Rem SPI-Nachricht einlesen
    Rem Par_12 enthält die unteren 32 Bit, Par_13 die restlichen
    Rem 16 Bit der SPI-Nachricht; die übrigen 16 Bit in Par_13
    Rem sind 0.
    P2_SPI_Master_Get_Value64(mod_no, master_no, Par_13, Par_12)
```

P2_SPI_Master_Get_Static_Input liest den Pegel auf der Datenleitung des SPI-Bus.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SPI_Master_Get_Static_Input(module,
                                           channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Masters.	LONG
ret_val	Pegel auf der Datenleitung: 0: Pegel low. 1: Pegel high.	LONG

Bemerkungen

Manche SPI-Slaves verwenden die Datenleitung nicht nur zur Datenübertragung, sondern übermitteln darüber auch Signale an den SPI-Master. Für diesen Fall können Sie mit **P2_SPI_Master_Get_Static_Input** den Pegel der Datenleitung lesen und je nach SPI-Slave darauf entsprechend reagieren.

Siehe auch

[P2_SPI_Mode](#), [P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Master_Status](#), [P2_SPI_Master_Get_Value32](#), [P2_SPI_Master_Get_Value64](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

- / -

P2_SPI_Master_Get_Static_Input

P2_SPI_Master_Set_Clk_Wait

P2_SPI_Master_Set_Clk_Wait fügt mehrere Wartezeiten nach einer wählbaren Anzahl von Takten in das Taktsignal eines SPI-Masters ein.

Syntax

```
#Include ADwinPro_All.inc

P2_SPI_Master_Set_Clk_Wait(module, channel, clocks,
    half_clk_wait[])
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Masters.	LONG
clocks	Anzahl (0...16) der Perioden im Taktsignal, nach denen eine Wartezeit eingefügt wird.	LONG
	Mit 0 werden die Wartezeiten deaktiviert.	
half_clk_wait[]	Feld mit 5 Wartezeiten, angegeben in halben Perioden (1...16).	ARRAY LONG
	Eine Wartezeit berechnet sich zu: $t_{\text{wait}} = 0,5 \times \text{half_clk_wait}[n] / f_{\text{SPI}}$	

Bemerkungen

Die Periodenlänge ergibt sich aus der Taktfrequenz f_{SPI} , die mit **P2_SPI_Master_Config** eingestellt wird.

Beachten Sie, dass eine Wartezeit mindestens eine halbe Periode lang ist.

Nach der letzten Periode des Taktsignals wird keine Wartezeit mehr eingefügt.

Wenn sich aus der SPI-Bitlänge und der Anzahl **clocks** ergibt, dass 6 oder mehr Wartezeiten in das Taktsignal eingefügt werden, wird die fünfte Wartezeit aus dem Feld **half_clk_wait** entsprechend oft wiederholt.

Siehe auch

[P2_SPI_Mode](#), [P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Slave_Config](#)

Gültig für

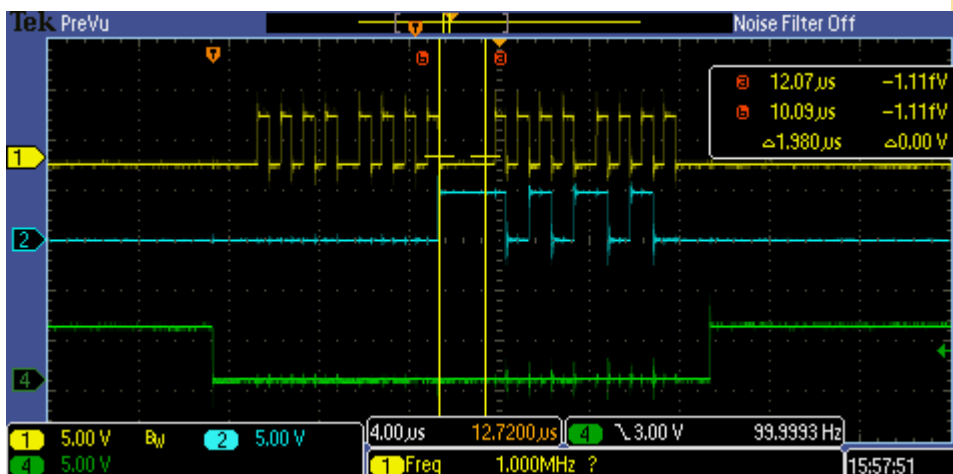
SPI-2-D Rev. E, SPI-2-T Rev. E

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define master_no 1
Dim clk_wait_arr[5] As Long

Init:
    P2_SPI_Mode(mod_no, 4)          '2 Master-Schnittstellen
    Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 16 Bit
    P2_SPI_Config(mod_no, master_no, 0, 16, 0, 0)
    Rem Taktfrequenz 1 MHz etc. einstellen
    P2_SPI_Master_Config(mod_no, master_no, 250, 75, 1, 0)
    Rem Wartezeiten definieren
    clk_wait_arr[1] = 1              'Wartezeit 1: 0,5µs
    clk_wait_arr[2] = 4              'Wartezeit 2: 2,0µs
    clk_wait_arr[3] = 1              'Wartezeit 3: 0,5µs
    clk_wait_arr[4] = 2              'Wartezeit 4: 1,0µs
    clk_wait_arr[5] = 1              'Wartezeit 5, 6, ...: 0,5µs
    Rem Wartezeiten nach jeder 4. Periode in das Taktsignal
    Rem einfügen
    P2_SPI_Master_Set_Clk_Wait(mod_no, master_no, 4, clk_wait_arr)

Event:
    Rem SPI-Nachricht zur Ausgabe bereitstellen
    P2_SPI_Master_Set_Value64(mod_no, master_no, 0, 0AAh)
    Rem Datenübertragung starten
    P2_SPI_Master_Start(mod_no, master_no)
```



Wartezeiten im Taktsignal (gelb) nach jeweils 4 Perioden;
hervorgehoben ist Wartezeit 2 mit 2,0µs

P2_SPI_Slave_Config

P2_SPI_Slave_Config legt (zusätzliche) Eigenschaften einer SPI-Slave-Schnittstelle des Moduls fest.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SPI_Slave_Config(module, channel, mode)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) der Schnittstelle, die als SPI-Slave konfiguriert ist.	LONG
mode	Modus „Bitanzahl speichern“ einstellen: 0: nur SPI-Nachrichten im Eingangs-Fifo speichern. 1: Anzahl der empfangenen Bits und SPI-Nachricht im Eingangs-Fifo speichern.	LONG

Bemerkungen

Sobald der Master die Slave-Select-Leitung deaktiviert, bricht der SPI-Slave den Datenempfang ab, auch wenn die SPI-Nachricht noch nicht vollständig übertragen wurde.

Die Anzahl der empfangenen Bits wird – wenn die Option aktiv ist – jeweils als 32 Bit-Wert vor der zugehörigen SPI-Nachricht im Eingangs-Fifo abgelegt.

Siehe auch

[P2_SPI_Mode](#), [P2_SPI_Config](#), [P2_SPI_Master_Config](#), [P2_SPI_Slave_OutFifo_Write](#), [P2_SPI_Slave_OutFifo_Empty](#), [P2_SPI_Slave_InFifo_Full](#), [P2_SPI_Slave_InFifo_Read](#), [P2_SPI_Slave_Clear_Fifo](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

- / -

P2_SPI_Slave_OutFifo_Write schreibt mehrere 32 Bit-Werte als SPI-Nachrichten in den Ausgangs-Fifo eines SPI-Slaves.

Syntax

```
#Include ADwinPro_All.inc

P2_SPI_Slave_OutFifo_Write(module, channel, count,
                           array[], array_idx)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Slaves.	LONG
count	Anzahl (1...18) der zu schreibenden 32 Bit-Werte.	LONG
array[]	Feld, das die zu schreibenden SPI-Nachrichten enthält.	ARRAY LONG
array_idx	Index (> 0) des ersten 32 Bit-Werts in array[] , der geschrieben wird.	LONG

Bemerkungen

Der SPI-Slave hält SPI-Nachrichten in einem Ausgangs-Fifo zur Ausgabe bereit. Damit kann der SPI-Slave sofort auf Signale des SPI-Masters reagieren.

Prüfen Sie erst mit **P2_SPI_Slave_OutFifo_Empty**, ob noch genügend Platz im Ausgangs-FIFO frei ist, bevor Sie neue SPI-Nachrichten hineinschreiben.

Der Ausgangs-Fifo kann 18 Werte zu 32 Bit aufnehmen. Je nach SPI-Bitlänge können daher entweder 18 SPI-Nachrichten mit bis zu 32 Bit oder 9 SPI-Nachrichten mit bis zu 64 Bit im Fifo gespeichert werden.

Im Feld **array[]** müssen die SPI-Nachrichten folgendermaßen angeordnet sein:

Index im Feld array[]	SPI-Bitlänge 1...32	SPI-Bitlänge 33...64
array_idx	Nachricht 1	Nachricht 1, unteres Wort
array_idx + 1	Nachricht 2	Nachricht 1, oberes Wort
array_idx + 2	Nachricht 3	Nachricht 2, unteres Wort
array_idx + 3	Nachricht 4	Nachricht 2, oberes Wort
...
array_idx + 14	Nachricht 15	Nachricht 8, unteres Wort
array_idx + 15	Nachricht 16	Nachricht 8, oberes Wort
array_idx + 16	Nachricht 17	Nachricht 9, unteres Wort
array_idx + 17	Nachricht 18	Nachricht 9, oberes Wort

Berücksichtigen Sie bei einer SPI-Bitlänge größer 32 Bit, dass

- in einer SPI-Nachricht 32 Bit-Werte immer paarweise verwendet werden. In der Regel wird daher **count** nur gerade Werte annehmen und **array_idx** nur ungerade Werte.
- im Feld **array[]** erst die unteren, dann die oberen 32 Bit einer SPI-Nachricht erwartet werden.

P2_SPI_Slave_OutFifo_Write

Das Feld `array[]` muss mindestens mit `array_idx + count - 1` Elementen dimensioniert werden.

Wenn der Ausgangs-Fifo leer läuft, wird die letzte geschriebene SPI-Nachricht so lange wiederholt, bis Sie neue Werte in den Ausgangs-Fifo geschrieben haben.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Slave_Config](#), [P2_SPI_Slave_OutFifo_Empty](#), [P2_SPI_Slave_InFifo_Full](#), [P2_SPI_Slave_InFifo_Read](#), [P2_SPI_Slave_Clear_Fifo](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1
Dim array[100] As Long

Init:
    P2_SPI_Mode(mod_no, 2)          '2 Slave-Schnittstellen
    Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 16 Bit
    P2_SPI_Config(mod_no, slave_no, 0, 16, 0, 0)
    P2_SPI_Slave_Config(mod_no, slave_no, 0)

Event:
    Rem wenn Platz im Ausgangs-Fifo vorhanden ist ..
    If (P2_SPI_Slave_OutFifo_Empty(mod_no, slave_no) > 0) Then
        Rem .. eine SPI-Nachricht zur Ausgabe bereitstellen
        array[1] = 50
        P2_SPI_Slave_OutFifo_Write(mod_no, slave_no, 1, array, 1)
    EndIf
```


P2_SPI_Slave_OutFifo_Empty gibt die Anzahl der freien Plätze im Ausgangs-Fifo zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val=P2_SPI_Slave_OutFifo_Empty(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Slaves.	LONG
ret_val	Anzahl der freien Plätze im Ausgangs-Fifo.	LONG

Bemerkungen

Wenn Sie Daten in das Ausgangs-FIFO schreiben wollen, sollten Sie vorher mit **P2_SPI_Slave_OutFifo_Empty** prüfen, ob noch genügend Platz im FIFO frei ist.

Ein freier Platz im Ausgangs-Fifo hat 32 Bit Länge. Bei einer SPI-Bitlänge kleiner gleich 32 Bit kann jeder freie Platz eine SPI-Nachricht aufnehmen; bei einer SPI-Bitlänge größer 32 Bit sind jeweils 2 Plätze für eine SPI-Nachricht nötig.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Slave_Config](#), [P2_SPI_Slave_OutFifo_Write](#), [P2_SPI_Slave_InFifo_Full](#), [P2_SPI_Slave_InFifo_Read](#), [P2_SPI_Slave_Clear_Fifo](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1
Dim array[100] As Long

Init:
    P2_SPI_Mode(mod_no, 2)          '2 Slave-Schnittstellen
    Rem CPOL = 0, CPHA = 0; Nachrichtenlänge 16 Bit
    P2_SPI_Config(mod_no, slave_no, 0, 16, 0, 0)
    P2_SPI_Slave_Config(mod_no, slave_no, 0)

Event:
    Rem wenn freie Plätze vorhanden sind ..
    If (P2_SPI_Slave_OutFifo_Empty(mod_no, slave_no) > 0) Then
        array[1] = 50
        Rem .. eine SPI-Nachricht zur Ausgabe bereitstellen
        P2_SPI_Slave_OutFifo_Write(mod_no, slave_no, 1, array, 1)
    EndIf
```

P2_SPI_Slave_OutFifo_Empty

P2_SPI_Slave_InFifo_Full

P2_SPI_Slave_InFifo_Full gibt die Anzahl der belegten Plätze (=eingegangene 32 Bit-Werte) im Eingangs-Fifo zurück.

Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_SPI_Slave_InFifo_Full(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...15).	__LONG
channel	Nummer (1, 2) des SPI-Slaves.	LONG
ret_val	Anzahl der belegten Plätze im Eingangs-Fifo.	LONG

Bemerkungen

Wenn Sie Daten aus dem Eingangs-FIFO lesen wollen, sollten Sie vorher mit diesem Befehl prüfen, ob im FIFO noch Daten enthalten sind. Falls keine Daten mehr vorhanden sind, wird aus dem FIFO-Feld ein undefinierter Wert gelesen.

Ein freier Platz im Eingangs-Fifo hat 32 Bit Länge. Bei einer SPI-Bitlänge kleiner gleich 32 Bit enthält jeder belegte Platz eine vollständige SPI-Nachricht; bei einer SPI-Bitlänge größer 32 Bit sind jeweils 2 Plätze für eine SPI-Nachricht nötig.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Slave_Config](#), [P2_SPI_Slave_OutFifo_Write](#), [P2_SPI_Slave_OutFifo_Empty](#), [P2_SPI_Slave_InFifo_Read](#), [P2_SPI_Slave_Clear_Fifo](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1
Dim array[100] As Long
Dim index As Long

Init:
    P2_SPI_Mode(mod_no, 2)          '2 Slave-Schnittstellen
    Rem CPOL = 0, CPHA = 1; Nachrichtenlänge 8 Bit
    P2_SPI_Config(mod_no, slave_no, 1, 8, 0, 0)
    P2_SPI_Slave_Config(mod_no, slave_no, 0)
    index = 1

Event:
    Par_1 = P2_SPI_Slave_InFifo_Full(mod_no, slave_no)
    If (Par_1 > 0) Then
        Rem eine SPI-Nachricht lesen
        P2_SPI_Slave_InFifo_Read(mod_no, slave_no, Par_1, array,
index)
        index = index + Par_1
        If (index > 100) Then index = 1
    EndIf
```

P2_SPI_Slave_InFifo_Read liest mehrere 32 Bit-Werte als SPI-Nachrichten aus dem Eingangs-Fifo eines SPI-Slaves.

Syntax

```
#Include ADwinPro_All.inc

P2_SPI_Slave_InFifo_Read(module, channel, count,
    array[], array_idx)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Slaves.	LONG
count	Anzahl (1...18) der zu lesenden 32 Bit-Werte.	LONG
array[]	Feld, das die gelesenen SPI-Nachrichten enthält.	ARRAY
		LONG
array_idx	Index des ersten 32 Bit-Werts in array[] , der gelesen wurde.	LONG

Bemerkungen

Der SPI-Slave sammelt eingehende SPI-Nachrichten in einem Eingangs-Fifo. Eine Nachricht wird im Eingangs-Fifo gespeichert, sobald die eingestellte Bitlänge erreicht ist oder wenn vorher die Slave-Select-Leitung deaktiviert wird.

Wenn die entsprechende Option mit **P2_SPI_Slave_Config** aktiviert wurde, wird die Anzahl der empfangenen Bits zusätzlich zu jeder SPI-Nachricht im Eingangs-Fifo abgelegt.

Prüfen Sie erst mit **P2_SPI_Slave_InFifo_Full**, ob bereits 32 Bit-Werte im Eingangs-FIFO enthalten ist, bevor Sie neue SPI-Nachrichten lesen.

Der Eingangs-Fifo kann 18 Werte zu 32 Bit aufnehmen. Je nach SPI-Bitlänge können daher entweder 18 SPI-Nachrichten mit bis zu 32 Bit oder 9 SPI-Nachrichten mit bis zu 64 Bit im Fifo gespeichert werden. Wenn auch die Anzahl der übertragenen Bits (siehe **P2_SPI_Config**) im Eingangs-Fifo abgelegt wird, können 9 SPI-Nachrichten mit bis zu 32 Bit oder 6 SPI-Nachrichten mit bis zu 64 Bit gespeichert werden.

In der folgenden Tabelle ist gezeigt, wie die Daten im Feld **array[]** gespeichert sind. Das Feld **array[]** muss mindestens mit **array_idx + count - 1** Elementen dimensioniert werden.

Index im Feld array[]	Ohne Anzahl übertragener Bits		Mit Anzahl übertragener Bits	
	Bitlänge 1...32	Bitlänge 33...64	Bitlänge 1...32	Bitlänge 33...64
array_idx	Nachricht 1	Nachricht 1, unteres Wort	Bitanzahl 1	Bitanzahl 1
array_idx + 1	Nachricht 2	Nachricht 1, oberes Wort	Nachricht 1	Nachricht 1, unteres Wort
array_idx + 2	Nachricht 3	Nachricht 2, unteres Wort	Bitanzahl 2	Nachricht 1, oberes Wort
array_idx + 3	Nachricht 4	Nachricht 2, oberes Wort	Nachricht 2	Bitanzahl 2
...
array_idx + 14	Nachricht 15	Nachricht 8, unteres Wort	Bitanzahl 8	Nachricht 5, oberes Wort
array_idx + 15	Nachricht 16	Nachricht 8, oberes Wort	SPI-Nach- richt 8	Bitanzahl 6
array_idx + 16	Nachricht 17	Nachricht 9, unteres Wort	Bitanzahl 9	Nachricht 6, unteres Wort

Index im Feld <code>array[]</code>	Ohne Anzahl übertragener Bits		Mit Anzahl übertragener Bits	
	Bitlänge 1...32	Bitlänge 33...64	Bitlänge 1...32	Bitlänge 33...64
<code>array_idx + 17</code>	Nachricht 18	Nachricht 9, oberes Wort	SPI-Nach- richt 9	Nachricht 6, oberes Wort

Berücksichtigen Sie bei einer SPI-Bitlänge größer 32 Bit, dass

- in einer SPI-Nachricht 32 Bit-Werte immer paarweise verwendet werden.
- in `array[]` erst die unteren, dann die oberen 32 Bit einer SPI-Nachricht gespeichert werden.

Wenn der Eingangs-Fifo voll ist und neue SPI-Nachrichten eingehen, werden sie nicht gespeichert und gehen damit verloren.

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Slave_Config](#), [P2_SPI_Slave_OutFifo_Write](#), [P2_SPI_Slave_OutFifo_Empty](#), [P2_SPI_Slave_InFifo_Full](#), [P2_SPI_Slave_Clear_Fifo](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
#Define mod_no 4
#Define slave_no 1
Dim array[100] As Long
Dim index As Long

Init:
    P2_SPI_Mode(mod_no, 2)          '2 Slave-Schnittstellen
    Rem CPOL = 0, CPHA = 1; Nachrichtenlänge 8 Bit
    P2_SPI_Config(mod_no, slave_no, 1, 8, 0, 0)
    P2_SPI_Slave_Config(mod_no, slave_no, 0)
    index = 1

Event:
    Par_1 = P2_SPI_Slave_InFifo_Full(mod_no, slave_no)
    If (Par_1 > 0) Then
        Rem eine SPI-Nachricht lesen
        P2_SPI_Slave_InFifo_Read(mod_no, slave_no, Par_1, array,
index)
        index = index + Par_1
        If (index > 100) Then index = 1
    EndIf
```

P2_SPI_Slave_Clear_Fifo löscht den Eingangs- und/oder den Ausgangs-Fifo eines SPI-Slaves.

Syntax

```
#Include ADwinPro_All.inc
```

```
P2_SPI_Slave_Clear_Fifo(module, channel, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
channel	Nummer (1, 2) des SPI-Slaves.	LONG
pattern	Bitmuster, das angibt, welcher Fifo des SPI-Slaves gelöscht werden soll: Bit 0: Eingangs-Fifo (MOSI) Bit = 0: Fifo unverändert lassen. Bit = 1: Fifo-Inhalt löschen. Bit 1: Ausgangs-Fifo (MOSI) Bit = 0: Fifo unverändert lassen. Bit = 1: Fifo-Inhalt löschen. Bits 31:2: keine Funktion.	LONG

Bemerkungen

- / -

Siehe auch

[P2_SPI_Config](#), [P2_SPI_Slave_Config](#), [P2_SPI_Slave_OutFifo_Write](#), [P2_SPI_Slave_OutFifo_Empty](#), [P2_SPI_Slave_InFifo_Full](#), [P2_SPI_Slave_InFifo_Read](#)

Gültig für

[SPI-2-D Rev. E](#), [SPI-2-T Rev. E](#)

Beispiel

```
#Include ADwinPro_All.inc
#define mod_no 4
#define slave_no 1
```

Init:

```
P2_SPI_Mode(mod_no, 2)          '2 Slave-Schnittstellen
Rem CPOL = 0, CPHA = 1; Nachrichtenlänge 8 Bit
P2_SPI_Config(mod_no, slave_no, 1, 8, 0, 0)
P2_SPI_Slave_Config(mod_no, slave_no, 0)

Rem beide Fifos löschen
P2_SPI_Slave_Clear_Fifo(mod_no, slave_no, 11b)
```

P2_SPI_Slave_Clear_Fifo

4 Programmbeispiele

Folgende Beispiele stehen zur Verfügung:

- [Online-Auswertung von Messwerten \(Pro II\)](#), Seite 458
- [Digitaler Proportional-Regler \(Pro II\)](#), Seite 459
- [Datenaustausch mit DATA-Feldern \(Pro II\)](#), Seite 459
- [Digitaler PID-Regler \(Pro II\)](#), Seite 460
- [Beispiele für RS232 und RS485 \(Pro II\)](#):
 - [RS232: Empfangen und senden](#), Seite 462
 - [RS232: String-Befehl senden](#), Seite 463
 - [RS232: String-Befehl empfangen](#), Seite 464
 - [RS485: Empfangen und senden](#), Seite 465
- [Kontinuierliche Messwertwandlung \(Pro II\): 1 Kanal wandeln](#), Seite 466

Die meisten Beispiele sind als Programmdateien im Verzeichnis C:\ADwin\ADbasic\samples_ADwin_PROII abgelegt.

4.1 Online-Auswertung von Messwerten (Pro II)

Das Programm ProII_DM01.BAS sucht den Maximal- und Minimalwert aus 1000 Messungen von ADC1 und schreibt das Ergebnis in die Variablen `Par_1` und `Par_2`.

Benötigt wird ein A/D-Modul Pro II-AIn-x/x mit Moduladresse 1 und ein Signal am Eingang 1 des Moduls.

```
#Include ADwinPro_All.Inc 'Include file
#Define limit 65535 'max. 16 bit ADC value
#Define module 1 'module number
#Define input 1 'input number
Dim il, iw, max, min As Long

Init:
    il = 1 'reset sample counter
    max = 0 'initial maximum value
    min = limit 'initial minimum value
    Par_10 = 0 'init End-Flag
    Processdelay = 1 * 3E5 'cycle-time of 1ms

Event:
    iw = P2_ADC(module, 1) 'get sample
    Rem for modules Pro II-AIn-F-x/x, delete the previous line and
    Rem use the following line instead (without comment char ')
    'iw = P2_ADCF(module, 1)
    If (iw > max) Then max = iw 'new maximum sample?
    If (iw < min) Then min = iw 'new minimum sample?
    Inc il 'increment index
    If (il > 1000) Then '1000 samples done?
        il = 1 'reset index
        Par_1 = min 'write minimum value
        Par_2 = max 'write maximum value
        max = 0 'reset minimum value
        min = 65535 'reset maximum value
        Par_10 = 1 'set End-Flag
    EndIf
```

4.2 Digitaler Proportional-Regler (Pro II)

Das Programm ProII_DMO2.BAS ist ein digitaler P-Regler. Der Sollwert wird mit `Par_1` vorgegeben, die Verstärkung mit `Par_2`.

Benötigt werden:

- A/D-Modul Pro II-AIn-x/x mit Moduladresse 1.
- D/A-Modul Pro II-AOut-x/x mit Moduladresse 2.
- Eine Regelstrecke, die das Signal des D/A-Moduls vom Ausgang 1 aufnimmt und ein Signal an den Eingang 1 des A/D-Moduls zurückgibt.

```
#Include ADwinPro_All.Inc 'Include file
#Define offset 32768      '0V for 16 bit ADC/DAC-systems
```

```
Rem cd: control deviation; av: actuating value
Dim cd, av As Long
```

Init:

```
Par_1 = offset      'initial setpoint
Par_2 = 10          'initial gain
Processdelay = 1 * 3E5 'cycle-time of 1ms
```

Event:

```
Rem Read signal from input 1 of A/D module
cd = Par_1 - P2_ADC(1, 1) 'compute control deviation (cd)
Rem for modules Pro II-AIn-F-x/x, delete the previous line and
Rem use the following line instead (without comment char ')
'cd = Par_1 - P2_ADCF(1, 1)
av = cd * Par_2 + offset 'compute actuating value (av)
Rem Write actuating value to output 1 of D/A module
P2_DAC(2, 1, av)        'output actuating value on DAC #1
```

4.3 Datenaustausch mit DATA-Feldern (Pro II)

Das Programm ProII_DMO3.BAS misst den analogen Eingang 1 des A/D-Moduls mit Adresse 1 und setzt nach 1000 Messungen eine Ende-Markierung, damit man am PC erkennen kann, wann die Messwerte abholbereit sind. Die Daten werden mit Hilfe des Felds `Data_1` übertragen.

Benötigt wird ein A/D-Modul Pro II-AIn-x/x mit Moduladresse 1 und ein Signal am Eingang 1 des Moduls.

```
#Include ADwinPro_All.Inc 'Include file
```

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

```
Dim index As Long
```

Init:

```
Par_10 = 0
index = 0 'reset array pointer
Processdelay = 40000 'cycle-time of 1ms (T9)
```

Event:

```
index = index + 1 'increment array pointer
If (index > 1000) Then '1000 samples done?
    Par_10 = 1 'set End-Flag
End 'terminate process
```

EndIf

```
Data_1[index] = P2_ADC(1, 1) 'acquire sample and save in array
Rem for modules Pro II-AIn-F-x/x, delete the previous line and
Rem use the following line instead (without comment char ')
'Data_1[index] = P2_ADCF(1, 1)
```



4.4 Digitaler PID-Regler (Pro II)

Das Programm `ProII_DMO6.BAS` ist ein digitaler PID-Regler.

Vor dem Starten des Reglers müssen die Reglereinstellungen in den globalen Variablen stehen.

Benötigt werden:

- A/D-Modul Pro II-AIn-x/x mit Moduladresse 1.
- D/A-Modul Pro II-AOut-x/x mit Moduladresse 2.
- Eine Regelstrecke, die das Signal des D/A-Moduls aufnimmt vom Ausgang 1 und ein Signal an den Eingang 1 des A/D-Moduls zurückgibt.

Berechnungen im PC:

Die Reglerkoeffizienten, der Sollwert und die Abtastrate werden auf dem PC berechnet und in globalen Variablen an den Prozessor des ADwin-Pro-Systems übergeben. Auf dem gleichen Weg werden Informationen aus dem Programm an den PC zurück gegeben:

Einstellparameter des Reglers

<code>FPar_2</code>	Verstärkung des Reglers
<code>FPar_3</code>	Integrationszeit des Reglers
<code>FPar_4</code>	Differentiationszeit des Reglers
<code>Par_1</code>	Sollwert in Digits
<code>Par_6</code>	Abtastrate des Reglers in Einheiten zu 3,3ns

Informationen aus dem Programm

<code>Par_5</code>	Feld-Index (zu <code>Data_1</code>) der Regelabweichung
<code>Par_9</code>	Mittlere Regelabweichung
<code>Par_10</code>	Flag: Alle Messungen sind durchgeführt
<code>Data_1 []</code>	Feld, das die Regelabweichungen enthält

ADbasic-Programm:

Die Adresse des analogen Eingangsmoduls ist im Beispiel auf 1, die Adresse des analogen Ausgangsmoduls auf 2 eingestellt.

Beachten Sie, dass im Programm eine Zeitersparnis dadurch erreicht wird, dass während den erforderlichen Wartezeiten beim Einlesen der Regelabweichung (nach `P2_Set_Mux` und `P2_Start_Conv`) die Berechnung und Ausgabe des Stellwerts durchgeführt wird.

Als Folge ergibt sich, dass jeweils der ausgegebene Stellwert aus der Regelabweichung des vorigen Prozessaufrufs berechnet wird.


```
#Include ADwinPro_All.Inc 'Include file

#Define offset 32768      '0V output
#Define module_ad 1      'module number
#Define module_da 2      'module number

Dim Data_1[4000] As Long
Dim av, cd, cdo, sum As Long
Dim diff As Float

Init:
    sum = 0                'initial value of integral part
    cd = P2_ADC(module_ad, 1) 'initial value of control deviation
                                '(cd) & MUX to Ch #1
    Par_5 = 1              'set array index
    If (FPar_3 < 75E2) Then FPar_3 = 75E2 'check min. integration time
    If (Par_6 < 3E5) Then Par_6 = 3E5 'allow only cycle times >= 1ms
    Processdelay = Par_6      'set cycle-time

Event:
    Rem compute actuating value
    av = FPar_2 * (cd + sum / FPar_3 + diff * FPar_4)
    P2_Start_Conv(module_ad) 'start conversion ADC #1
    Rem while conversion is running ...
    P2_DAC(module_da, 1, av + offset) 'output actuating value at DAC #1
    cdo = cd                        'keep control deviation in mind
    P2_Wait_EOC(module_ad)         'wait until end-of-conversion of ADC
    cd = Par_1 - P2_Read_ADC(module_ad) 'compute control deviation
    FPar_9 = FPar_9 * 0.99 + cd * 0.01 'mean value of control
                                        'deviation
    sum = sum + cd                  'calculate integral
    If (sum > 2E6) Then sum = 2E6 'positive limit of integral
    If (sum < -2E6) Then sum = -2E6 'negative limit of integral
    diff = (cd - cdo)              'calculate deviation difference
    Data_1[Par_5] = cd             'write control deviation in a buffer
    Inc Par_5                      'increment buffer index
    If (Par_5 >= 4000) Then        '4000 samples done?
        Par_10 = 1                'set End-flag
        Par_5 = 1                'reset array index
    EndIf

Finish:
    P2_DAC(module_da, 1, av + offset) 'analog output #1 to 0V

Rem Note: For modules Pro II-AIn-F-x/x, the A/D instructions
Rem must be renamed, parameters are left unchanged:
Rem * P2_ADC -> P2_ADCF
Rem * P2_Start_Conv -> P2_Start_ConvF
Rem * P2_Wait_EOC -> P2_Wait_EOCF
Rem * P2_Read_ADC -> P2_Read_ADCF
```

RS232: Empfangen und senden

4.5 Beispiele für RS232 und RS485 (Pro II)

Die folgenden Beispiele sind vollständige Programme für das Senden und Empfangen von Daten und Strings mit RS232 oder RS485.

Benötigt wird ein Modul Pro II-RSxxx mit Moduladresse 1.

Das Programm zeigt die Initialisierung der seriellen RS232-Schnittstelle im Abschnitt **Init:** und das zyklische Lesen und Schreiben von Daten im Abschnitt **Event:**. Der Prozess ist zeitgesteuert.

```
Rem The program initializes the serial interfaces in
Rem the INIT: section.
Rem In the EVENT: section data are exchanged between interfaces
Rem 1 & 2 of the RSxxx module.
Rem With this program both interfaces can be tested.
Rem To do so, connect the interfaces with each other before
Rem starting the program.
```

```
#Include ADwinPro_All.Inc
#Define num_data 1000      'number of send and receive data
#Define module 1          'Module address
Dim Data_1[num_data] As Long 'send data
Dim Data_2[num_data] As Long 'receive data
Dim i As Long             'count variable
```

Init:

```
P2_RS_Reset(module)
For i = 1 To num_data      'initialize send data
    Data_1[i] = i And 0FFh
Next i
Rem Initialize interfaces 1 and 2:
Rem 9600 Baud, no parity bit, 8 data bits, 2 stop bits,
Rem no handshake
P2_RS_Init(module, 1, 9600, 0, 8, 1, 0)
P2_RS_Init(module, 2, 9600, 0, 8, 1, 0)
Par_1 = 1
Par_4 = 1
```

Event:

```
Rem read and write a data set
If (Par_1 <= num_data) Then 'send data
    Par_2 = P2_Write_FIFO(module, 1, Data_1[Par_1])
    If (Par_2 = 0) Then Inc Par_1
EndIf

Par_3 = P2_Read_FIFO(module, 2) 'read data
If (Par_3 <> -1) Then
    Data_2[Par_4] = Par_3
    Inc Par_4
EndIf
If (Par_4 > num_data) Then End 'all data are transferred
```

Benötigt wird ein Modul Pro II-RSxxx mit Moduladresse 1.

Viele Geräte mit RS232-Schnittstelle können mit String-Befehlen gesteuert werden. Die beiden folgenden Programme zeigen, wie man mit einem Prozess eine Zeichenfolge sendet und mit einem anderen Prozess die Zeichenfolge empfängt. Die Programme sind im ADwin-Software-Paket verfügbar.

Die Programme können auf dem gleichen Modul, jedoch mit verschiedenen Schnittstellen eingesetzt werden. Beachten Sie bitte die Kommentare im Programm.

Das Programm RS232_send_string.BAS initialisiert zuerst die RS232-Schnittstelle. Im Abschnitt **Event:** sendet die Schnittstelle 1 des RS-Moduls eine Zeichenfolge. Im Abschnitt **Finish:** wird das Zeichen „#“ als Ende-Markierung gesendet. Es kann durch ein beliebiges anderes Zeichen ersetzt werden.

```
' Process for RS232 communication: sending a string
' +-----+
' The program may run together with RS232_receive_string.BAS
' on the same module. If so, please follow these instructions:
' - connect the interfaces with each other
' - compile and start RS232_receive_string.BAS
' - compile and start RS232_send_string.BAS

#include ADwinPro_All.Inc

Rem import string library
Import string.lib

#define rs_adr 1           'module address
#define rs_no 1           'interface number
#define s_endchar "#"     'end marker "#"
#define s_send Data_1
#define str_len 50        'length of send string

Dim s_send[str_len] As String 'send string
Dim s_temp[1] As String      'single char
Dim sp As Long               'send pointer

Init:
    Rem 0.25 s
    Processdelay = 75E6
    Rem A reset is allowed only once on a module!
    'P2_RS_Reset(rs_adr)      'reset RS module
    P2_RS_Init(rs_adr,rs_no,9600,0,8,0,0) 'init RS interface
    sp=1                      'initialize pointer
    s_send = „This is a TESTSTRING“ 'send string

Event:
    StrMid(s_send, sp, 1, s_temp) 'read next char of string
    Par_11 = Asc(s_temp)           'get ascii code of char
    If (Par_11 = 0) Then End       'quit when all chars are sent
    Par_12 = P2_Write_FIFO(rs_adr, rs_no, Par_11) 'send code
    Rem increase pointer, else send again
    If (Par_12 = 0) Then Inc sp
    Rem quit when all chars are sent
    If (sp > str_len) Then End

Finish:
    Do                            'send End marker "#"
        Par_11 = Asc(s_endchar) 'get ascii code
        Par_12 = P2_Write_FIFO(rs_adr, rs_no, Par_11) 'send code
    Until (Par_12 = 0)
```

RS232: String-Befehl senden

RS232: String-Befehl empfangen

Benötigt wird ein RS232-Modul mit Moduladresse 1 (Gruppe EXT-Module).

Das Programm RS232_receive_string.BAS initialisiert zuerst das Modul und die Schnittstelle 2. Im Abschnitt **Event:** wird eine Zeichenfolge über die Schnittstelle 2 empfangen, bis die Ende-Markierung empfangen wird (oder der Empfangs-String voll ist).

```
' Process for RS232-communication: Receiving a string.
' +-----+
' The program may run together with RS232_send_string.BAS
' on the same module. If so, please follow these instructions:
' - connect the interfaces with each other
' - compile and start RS232_receive_string.BAS
' - compile and start RS232_send_string.BAS
#include ADwinPro_All.Inc

Rem import string library
Import string.lib

#Define rs_adr 1           'module address
#Define rs_no 2           'interface number
#Define s_receive Data_2
#Define str_len 50        'max. length of received string

Dim s_receive[str_len] As String 'received string
Dim s_temp[1] As String 'single char
Dim s_endchar[1] As String 'end marker
Dim endflag As Long
Dim rp As Long           'receive pointer

Init:
    Rem 0.25 s
    Processdelay = 75000000

    Rem A reset is allowed only once on a module!
    P2_RS_Reset(rs_adr) 'reset RS module
    P2_RS_Init(rs_adr,rs_no,9600,0,8,0,0) 'init RS interface
    rp = 0 'initialize receive pointer
    s_receive = "" 'initialize receive string
    s_endchar = "#" 'end marker

Event:
    Par_21 = P2_Read_FIFO(rs_adr, rs_no) 'receive status / char
    If (Par_21 <> -1) Then
        Chr(Par_21,s_temp) 'get char from ascii value
        Inc rp 'increase receive pointer
        Rem End marker received or string full?
        endflag = StrComp(s_temp, s_endchar)
        If ((endflag=0) Or (rp>str_len)) Then End
        s_receive = s_receive + s_temp 'save char to string
    EndIf
```

Benötigt wird ein Modul Pro II-RSxxx mit Moduladresse 1.

In diesem Beispiel wird eine RS485-Schnittstelle als passiver Teilnehmer verwendet, der alle Daten liest, die an seinem Eingang anliegen. Wenn ein bestimmter Wert (55) empfangen wird, wird die Schnittstelle aktiv und sendet dann ihrerseits fortlaufend z.B. den Wert 44.

```
#Include ADwinPro_All.Inc
#Define rs_adr 1
#Define rs_no 2

#Define val_to_send Par_1
#Define received_val Par_2
#Define status Par_3

Init:
  P2_RS_Reset(rs_adr)
  P2_RS_Init(rs_adr,rs_no,38400,0,8,0,3)
  P2_RS485_Send(rs_adr,rs_no,0) 'set channel 2 as receiving
  val_to_send = 44

Event:
  received_val = P2_Read_FIFO(rs_adr,rs_no) 'read data
  If (received_val = 55) Then
    P2_RS485_Send(rs_adr,rs_no,1) 'set channel 2 as sending
    status = P2_Write_FIFO(rs_adr,rs_no,val_to_send) 'send value
  EndIf
```

RS485:
Empfangen und senden

4.6 Kontinuierliche Messwertwandlung (Pro II)

Die Module Pro II AIn-F-4/14 Rev. E und Pro II AIn-F-8/14 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.

Beispieldateien sind in C:\ADwin\ADbasic\samples_ADwin_ProII gespeichert.

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-AIn-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 verloren, weil sie schneller überschrieben als gelesen werden, und Sie müssen die Abstimmung von Wandlungsrate und Leserate neu durchführen.

```
#Include ADwinPro_All.inc
#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

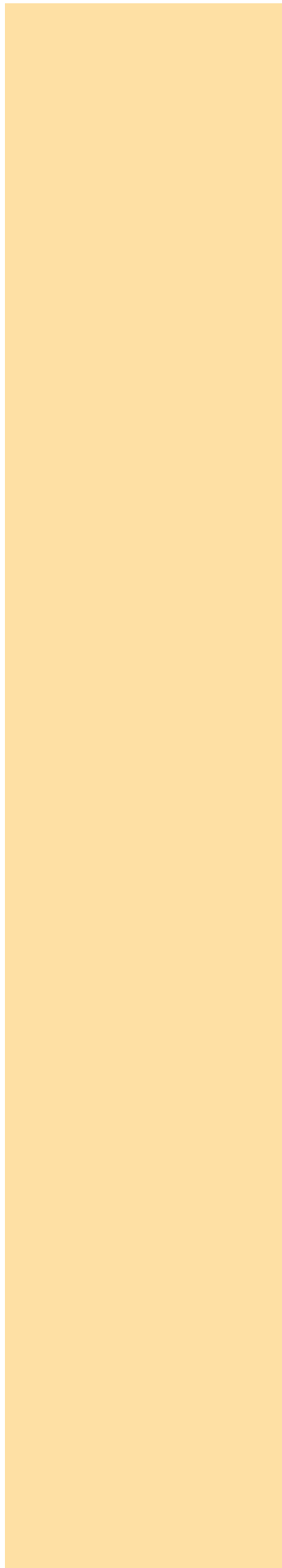
Init:
  Rem 1 channel continuous, mem for max_val values, 25 MHz
  P2_Burst_Init(module,1,0,max_val,2,010b)
  pattern = Shift_Left(1,module-1) 'address this module only
  P2_Burst_Start(pattern)
  segment = 1                    'start with memory segment 1
  Processdelay = 20000          'cycle time 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 · 37
P2_ADC24 · 38
P2_ADCF · 98
P2_ADCF24 · 99
P2_ADCF_Mode · 100
P2_ADCF_Read_Limit · 102
P2_ADCF_Read_Min_Max4 · 105
P2_ADCF_Read_Min_Max8 · 107
P2_ADCF_Reset_Min_Max · 104
P2_ADCF_Set_Limit · 103
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
P2_ARINC_Config_Receive · 361
P2_ARINC_Config_Transmit · 360
ARINC_Create_Value32 · 366
P2_ARINC_Read_Receive_Fifo · 369
P2_ARINC_Receive_Fifo_Empty · 368
P2_ARINC_Reset · 359
P2_ARINC_Set_Labels · 371
ARINC_Split_Value32 · 370
P2_ARINC_Transmit_Enable · 367
P2_ARINC_Transmit_Fifo_Empty · 364
P2_ARINC_Transmit_Fifo_Full · 363
P2_ARINC_Write_Transmit_Fifo · 365
P2_CANFD_Get_BDIAG0 · 266
P2_CANFD_Get_BDIAG1 · 267
P2_CANFD_Get_BRS · 255
P2_CANFD_Get_DLC · 258
P2_CANFD_Get_ESI · 253
P2_CANFD_Get_FDF · 254
P2_CANFD_Get_Fifo_State · 247
P2_CANFD_Get_Header_Parts · 250
P2_CANFD_Get_ID · 252
P2_CANFD_Get_IDE · 257
P2_CANFD_Get_RTR · 256
P2_CANFD_Get_SEQ · 259
P2_CANFD_Get_TREC · 265
P2_CANFD_Init_Datatable · 234
P2_CANFD_Init_Controller · 236
P2_CANFD_Read_EFO · 260
P2_CANFD_Read_RMO · 249
P2_CANFD_Set_Baudrate_Data · 243
P2_CANFD_Set_Baudrate_Nominal · 242
P2_CANFD_Set_LED · 233
P2_CANFD_Set_Mode · 246
P2_CANFD_Set_Reg · 264
P2_CANFD_Set_SID11 · 244
P2_CANFD_Set_TDC · 245
P2_CANFD_Transmit_Msg · 263
P2_CANFD_Transmit_Multi_Msg · 264
P2_CANFD_Write_TMO · 261
P2_CAN_Interrupt_Source · 214
CAN_Msg (Pro II) · 213
P2_CAN_Set_LED · 216
P2_Check_LED · 4
P2_Check_Shift_Reg · 325
P2_Cnt_Clear · 187
P2_Cnt_Enable · 188
P2_Cnt_Get_PW · 192
P2_Cnt_Get_PW_HL · 193
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 194
P2_Cnt_Mode · 195
P2_Cnt_PW_Enable · 189
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read4 · 199
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Read_Latch4 · 202
P2_Cnt_Sync_Latch · 203
P2_Comp_Filter_Init · 147
P2_Comp_Init · 145
P2_Comp_Set · 148
P2_Comp_Set_Voltage · 149
CPU_Digin (T11, T12) · 22
CPU_Digout · 23
CPU_Dig_IO_Config · 24
CPU_Event_Config · 25

B

P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Pos_Unpacked8 · 74
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_CRead_Unpacked8 · 66
P2_Burst_Init · 76
P2_Burst_Read · 81
P2_Burst_Read_Index · 79
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Read_Unpacked8 · 90
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 96

C

Calc_Processdelay · 6
Calc_TicksToNs · 7
P2_CANFD_Enable_Receive_Fifo · 237
P2_CANFD_Enable_Transmit_Event_Fifo · 241
P2_CANFD_Enable_Transmit_Fifo · 239
P2_CANFD_Enable_Transmit_Queue · 240

D

P2_DAC · 125
P2_DAC1_DIO · 137
P2_DAC4 · 126
P2_DAC4_Packed · 127
P2_DAC8 · 128
P2_DAC8_Packed · 129
P2_DAC_Ramp_Buffer_Free · 142
P2_DAC_Ramp_Status · 140
P2_DAC_Ramp_Stop · 143
P2_DAC_Ramp_Write · 138
P2_Digin_Edge · 155
P2_Digin_Fifo_Clear · 157
P2_Digin_Fifo_Enable · 158
P2_Digin_Fifo_Full · 159
P2_Digin_Fifo_Read · 160
P2_Digin_Fifo_Read_Fast · 162
P2_Digin_Fifo_Read_Timer · 164
P2_Digin_Filter_Init · 165
P2_Digin_Long · 167
P2_Digout · 168
P2_Digout_Bits · 169
P2_Digout_Fifo_Clear · 171
P2_Digout_Fifo_Empty · 172
P2_Digout_Fifo_Enable · 173
P2_Digout_Fifo_Read_Timer · 174
P2_Digout_Fifo_Start · 175
P2_Digout_Fifo_Write · 176
P2_Digout_Long · 179
P2_Digout_Reset · 180
P2_Digout_Set · 181
P2_DigProg · 182
P2_DigProg_Bits · 183
P2_DigProg_Set_IO_Level · 184
P2_Dig_Fifo_Mode · 150
P2_Dig_Latch · 152
P2_Dig_Read_Latch · 153
P2_Dig_Write_Latch · 154

E

P2_ECAT_Get_State · 374
P2_ECAT_Get_Version · 373
P2_ECAT_Init · 375
P2_ECAT_Read_Data_16F · 380
P2_ECAT_Read_Data_16L · 378
P2_ECAT_Set_Mode · 377
P2_ECAT_Write_Data_16F · 381
P2_ECAT_Write_Data_16L · 379
P2_En_Interrupt · 217
P2_En_Receive · 218
P2_En_Transmit · 219
P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12

F

P2_FlexRay_Get_Version · 386

P2_FlexRay_Init · 387
P2_FlexRay_Read_Word · 388
P2_FlexRay_Reset · 389
P2_FlexRay_Set_LED · 390
P2_FlexRay_Write_Word · 391

G

P2_Get_CAN_Reg · 220
P2_Get_Digout_Long · 185
P2_Get_RS · 326

I

P2_Init_CAN · 221
P2_Init_EtherCAT · 382
P2_Init_Profibus · 339
P2_Init_Profibus_M40 · 342
P2_Init_ProfinetIO · 345

L

P2_LIN_Ch_Read_Cnt · 278
P2_LIN_Get_Version · 275
P2_LIN_Init · 270
P2_LIN_Init_Apply · 273
P2_LIN_Init_Write · 272
P2_LIN_Msg_Read_Status · 279
P2_LIN_Msg_Transmit · 281
P2_LIN_Msg_Write · 280
P2_LIN_Read_Dat · 276
P2_LIN_Reset · 274
P2_LIN_Set_LED · 282

M

P2_MIL_Get_Register · 357
P2_MIL_Reset · 349
P2_MIL_Set_LED · 355
P2_MIL_Set_Register · 356
P2_MIL_SMT_Init · 350
P2_MIL_SMT_Message_Read · 351
P2_MIL_SMT_Set_All_Filters · 353
P2_MIL_SMT_Set_Filter · 354
P2_MIO_Digin_Long · 30
P2_MIO_Digout · 31
P2_MIO_Digout_Long · 32
P2_MIO_DigProg · 33
P2_MIO_Dig_Latch · 27
P2_MIO_Dig_Read_Latch · 28
P2_MIO_Dig_Write_Latch · 29
P2_MIO_Get_Digout_Long · 34

P

P2_PWM_Enable · 284
P2_PWM_Get_Status · 285
P2_PWM_Init · 286
P2_PWM_Latch · 288
P2_PWM_Reset · 289
P2_PWM_Standby_Value · 290
P2_PWM_Write_Latch · 291
P2_PWM_Write_Latch_Block · 292

R

P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADCF · 109
P2_Read_ADCF24 · 110
P2_Read_ADCF32 · 117
P2_Read_ADCF4 · 111
P2_Read_ADCF4_24B · 112
P2_Read_ADCF4_Packed · 115
P2_Read_ADCF8 · 113
P2_Read_ADCF8_24B · 114
P2_Read_ADCF8_Packed · 116
P2_Read_ADCF_SConv · 118
P2_Read_ADCF_SConv24 · 119
P2_Read_ADCF_SConv32 · 120
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
P2_Read_Fifo · 327
P2_Read_Msg · 222
P2_Read_Msg_Con · 224
P2_RS485_Send · 331
P2_RS_Init · 328
P2_RS_Reset · 330
P2_RS_Set_LED · 332
P2_RTD_Channel_Config · 295
P2_RTD_Config · 297
P2_RTD_Convert · 298
P2_RTD_Read · 299
P2_RTD_Read8 · 300
P2_RTD_Start · 301
P2_RTD_Status · 303
P2_Run_EtherCAT · 384
P2_Run_Profibus · 341
P2_Run_Profibus_M40 · 344
P2_Run_ProfinetIO · 347

S

P2_SENT_Clear_Serial_Message_Array · 409
P2_SENT_Command_Ready · 396
P2_SENT_Get_ChannelState · 397
P2_SENT_Get_ClockTick · 398
P2_SENT_Get_Fast_Channel1 · 401
P2_SENT_Get_Fast_Channel2 · 403
P2_SENT_Get_Fast_Channel_CRC_OK · 400
P2_SENT_Get_Msg_Counter · 395
P2_SENT_Get_PulseCount · 399
P2_SENT_Get_Serial_Message_Array · 407
P2_SENT_Get_Serial_Message_CRC_OK · 404
P2_SENT_Get_Serial_Message_Data · 406
P2_SENT_Get_Serial_Message_Id · 405
P2_SENT_Get_Version · 394
P2_SENT_Init · 393
P2_SENT_Set_ClockTick · 412
P2_SENT_Set_CRC_Implementation · 410
P2_SENT_Set_Detection · 411
P2_SENT_Set_PulseCount · 413
P2_SENT_Check_Latch · 416
P2_SENT_Config_Output · 421
P2_SENT_Config_Serial_Messages · 422

P2_SENT_Enable_Channel · 423
P2_SENT_Fifo_Clear · 431
P2_SENT_Fifo_Empty · 430
P2_SENT_Get_Latch_Data · 417
P2_SENT_Invert_Channel · 424
P2_SENT_Request_Latch · 415
P2_SENT_Set_Fast_Channel1 · 426
P2_SENT_Set_Fast_Channel2 · 427
P2_SENT_Set_Fifo · 432
P2_SENT_Set_Output_Mode · 420
P2_SENT_Set_Reserved_Bits · 425
P2_SENT_Set_Sensor_Type · 414
P2_SENT_Set_Serial_Message_Data · 429
P2_SENT_Set_Serial_Message_Pattern · 428
P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_Average_Filter · 97
P2_Set_CAN_Baudrate · 226
P2_Set_CAN_Reg · 227
P2_Set_Gain · 121
P2_Set_LED · 5
P2_Set_Mux · 56
P2_Set_RS · 333
P2_SE_Diff · 46
P2_SG_Convert · 318
P2_SG_Init · 319
P2_SG_Mode · 313
P2_SG_Read · 317
P2_SG_Set_Gain · 322
P2_SG_Start · 315
P2_SG_Wait · 316
P2_SG_Zero · 321
P2_SPI_Config · 435
P2_SPI_Master_Config · 437
P2_SPI_Master_Get_Static_Input · 447
P2_SPI_Master_Get_Value32 · 445
P2_SPI_Master_Get_Value64 · 446
P2_SPI_Master_Set_Clk_Wait · 448
P2_SPI_Master_Set_Value32 · 441
P2_SPI_Master_Set_Value64 · 442
P2_SPI_Master_Start · 443
P2_SPI_Master_Status · 444
P2_SPI_Mode · 434
P2_SPI_Slave_Clear_Fifo · 457
P2_SPI_Slave_Config · 450
P2_SPI_Slave_InFifo_Full · 454
P2_SPI_Slave_InFifo_Read · 455
P2_SPI_Slave_OutFifo_Empty · 453
P2_SPI_Slave_OutFifo_Write · 451
P2_SSI_Mode · 204
P2_SSI_Read · 205
P2_SSI_Read2 · 206
P2_SSI_Set_Bits · 207
P2_SSI_Set_Clock · 208
P2_SSI_Set_Delay · 209

[P2_SSI_Start · 210](#)
[P2_SSI_Status · 211](#)
[P2_Start_Conv · 57](#)
[P2_Start_ConvF · 122](#)
[P2_Start_DAC · 130](#)
[P2_Sync_All · 13](#)
[P2_Sync_Enable · 15](#)
[P2_Sync_Mode · 17](#)
[P2_Sync_Stat · 19](#)

T

[P2_TC_Latch · 304](#)
[P2_TC_Read_Latch · 305](#)
[P2_TC_Read_Latch4 · 307](#)
[P2_TC_Read_Latch8 · 309](#)
[P2_TC_Set_Rate · 311](#)
[P2_Transmit · 228](#)
[P2_Transmit_Status · 230](#)

W

[P2_Wait_EOC · 58](#)
[P2_Wait_EOCF · 123](#)
[P2_Wait_Mux · 59](#)
[P2_Write_DAC · 131](#)
[P2_Write_DAC32 · 136](#)
[P2_Write_DAC4 · 132](#)
[P2_Write_DAC4_Packed · 133](#)
[P2_Write_DAC8 · 134](#)
[P2_Write_DAC8_Packed · 135](#)
[P2_Write_Fifo · 334](#)
[P2_Write_Fifo_Full · 335](#)

A.2 Befehlsübersicht nach Modulen

Sie finden die Befehlsübersichten der Module auf den folgenden Seiten. Die Module sind alphabetisch nach Namen sortiert:

Modulname	Seite
Aln-16/18-8B Rev. E	A-6
Aln-16/18-C Rev. E	A-6
Aln-32/18-D Rev. E	A-6
Aln-32/18-D-TiCo Rev. E	A-6
Aln-8/18 Rev. E	A-7
Aln-8/18-8B Rev. E	A-7
Aln-8/18-TiCo Rev. E	A-7
Aln-F-4/14 Rev. E	A-8
Aln-F-4/16 Rev. E	A-8
Aln-F-4/18 Rev. E	A-9
Aln-F-8/14 Rev. E	A-9
Aln-F-8/16 Rev. E	A-10
Aln-F-8/18 Rev. E	A-10
AOut-1/16 Rev. E	A-11
AOut-4/16 Rev. E	A-11
AOut-4/16-TiCo Rev. E	A-11
AOut-8/16 Rev. E	A-11
AOut-8/16-TiCo Rev. E	A-12
ARINC-429 Rev. E	A-12
CAN-2 Rev. E	A-12
CAN-FD-2 Rev. E	A-12
CNT-D Rev. E	A-13
CNT-I Rev. E	A-13
CNT-T Rev. E	A-13
Comp-16 Rev. E	A-13
CPU-T10	A-13
CPU-T11	A-14
CPU-T12	A-14
CPU-T9	A-14
DIO-32 Rev. E	A-14
DIO-32-TiCo Rev. E	A-14
DIO-32-TiCo2 Rev. E	A-15
DIO-32/1-TiCo Rev. E	A-15
DIO-8-D12 Rev. E	A-16
EtherCAT-SL Rev. E	A-16
EtherCAT-SL-40 Rev. E	A-16
FlexRay-2 Rev. E	A-16
LIN-2 Rev. E	A-16
LS-2 Rev. E	A-16
MIL-1553 Rev. E	A-16

Modulname	Seite
MIO-4 Rev. E	A-17
MIO-4-ET1 Rev. E	A-18
MIO-D12 Rev. E	A-18
OPT-16 Rev. E	A-19
OPT-32-24V Rev. E	A-19
Profi-IRT-CU Rev. E	A-19
Profi-IRT-CU-40 Rev. E	A-19
Profi-IRT-FO Rev. E	A-19
Profi-IRT-FO-40 Rev. E	A-19
Profi-SL Rev. E	A-19
Profi-SL-40 Rev. E	A-19
PWM-16(-I) Rev. E	A-19
REL-16 Rev. E	A-19
RS422-4 Rev. E	A-20
RSxxx-2 Rev. E	A-20
RSxxx-4 Rev. E	A-20
RTD-8 Rev. E	A-20
SENT-4 Rev. E	A-20
SENT-4-Out Rev. E	A-21
SENT-6 Rev. E	A-21
SG-4/18 Rev. E	A-21
SPI-2-D Rev. E	A-22
SPI-2-T Rev. E	A-22
TC-8-ISO Rev. E	A-23
TRA-16 Rev. E	A-23

Aln-16/18-8B Rev. E

- A:** P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
- C:** P2_Check_LED · 4
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- R:** P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
- S:** P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 5
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_Start_Conv · 57
P2_Sync_All · 13
- W:** P2_Wait_EOC · 58
P2_Wait_Mux · 59

Aln-16/18-C Rev. E

- A:** P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
P2_Read_ADC_SConv · 44
- C:** P2_Check_LED · 4
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- R:** P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv24 · 45
- S:** P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 5
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_Start_Conv · 57
P2_Sync_All · 13
- W:** P2_Wait_EOC · 58
P2_Wait_Mux · 59

Aln-32/18-D Rev. E

- A:** P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
- C:** P2_Check_LED · 4
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- R:** P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
- S:** P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 5
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_Start_Conv · 57
P2_Sync_All · 13
- W:** P2_Wait_EOC · 58
P2_Wait_Mux · 59

Aln-32/18-D-TiCo Rev. E

- A:** P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
- C:** P2_Check_LED · 4
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- R:** P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
- S:** P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 5
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_Start_Conv · 57
P2_Sync_All · 13
- W:** P2_Wait_EOC · 58
P2_Wait_Mux · 59

Aln-8/18 Rev. E

- A:** [P2_ADC · 37](#)
[P2_ADC24 · 38](#)
[P2_ADC_Read_Limit · 39](#)
[P2_ADC_Set_Limit · 41](#)
- C:** [P2_Check_LED · 4](#)
- E:** [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- R:** [P2_Read_ADC · 42](#)
[P2_Read_ADC24 · 43](#)
[P2_Read_ADC_SConv · 44](#)
[P2_Read_ADC_SConv24 · 45](#)
- S:** [P2_Seq_Init · 47](#)
[P2_Seq_Read · 50](#)
[P2_Seq_Read24 · 51](#)
[P2_Seq_Read_Packed · 53](#)
[P2_Seq_Start · 54](#)
[P2_Seq_Wait · 55](#)
[P2_Set_LED · 5](#)
[P2_Set_Mux · 56](#)
[P2_Start_Conv · 57](#)
[P2_Sync_All · 13](#)
- W:** [P2_Wait_EOC · 58](#)
[P2_Wait_Mux · 59](#)

Aln-8/18-8B Rev. E

- A:** [P2_ADC · 37](#)
[P2_ADC24 · 38](#)
[P2_ADC_Read_Limit · 39](#)
[P2_ADC_Set_Limit · 41](#)
- C:** [P2_Check_LED · 4](#)
- E:** [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- R:** [P2_Read_ADC · 42](#)
[P2_Read_ADC24 · 43](#)
[P2_Read_ADC_SConv · 44](#)
[P2_Read_ADC_SConv24 · 45](#)
- S:** [P2_Seq_Init · 47](#)
[P2_Seq_Read · 50](#)
[P2_Seq_Read24 · 51](#)
[P2_Seq_Read_Packed · 53](#)
[P2_Seq_Start · 54](#)
[P2_Seq_Wait · 55](#)
[P2_Set_LED · 5](#)
[P2_Set_Mux · 56](#)
[P2_Start_Conv · 57](#)
[P2_Sync_All · 13](#)
- W:** [P2_Wait_EOC · 58](#)
[P2_Wait_Mux · 59](#)

Aln-8/18-TiCo Rev. E

- A:** [P2_ADC · 37](#)
[P2_ADC24 · 38](#)
[P2_ADC_Read_Limit · 39](#)
[P2_ADC_Set_Limit · 41](#)
- C:** [P2_Check_LED · 4](#)
- E:** [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- R:** [P2_Read_ADC · 42](#)
[P2_Read_ADC24 · 43](#)
[P2_Read_ADC_SConv · 44](#)
[P2_Read_ADC_SConv24 · 45](#)
- S:** [P2_Seq_Init · 47](#)
[P2_Seq_Read · 50](#)
[P2_Seq_Read24 · 51](#)
[P2_Seq_Read_Packed · 53](#)
[P2_Seq_Start · 54](#)
[P2_Seq_Wait · 55](#)
[P2_Set_LED · 5](#)
[P2_Set_Mux · 56](#)
[P2_Start_Conv · 57](#)
[P2_Sync_All · 13](#)
- W:** [P2_Wait_EOC · 58](#)
[P2_Wait_Mux · 59](#)

Aln-F-4/14 Rev. E

- A:** P2_ADCF · 98
P2_ADCF_Read_Limit · 102
P2_ADCF_Set_Limit · 103
- B:** P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_Init · 76
P2_Burst_Read · 81
P2_Burst_Read_Index · 79
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 96
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- R:** P2_Read_ADCF · 109
P2_Read_ADCF32 · 117
P2_Read_ADCF4 · 111
P2_Read_ADCF4_Packed · 115
- S:** P2_Set_Average_Filter · 97
P2_Set_LED · 5
P2_Sync_All · 13

Aln-F-4/16 Rev. E

- A:** P2_ADCF · 98
P2_ADCF_Mode · 100
P2_ADCF_Read_Limit · 102
P2_ADCF_Read_Min_Max4 · 105
P2_ADCF_Read_Min_Max8 · 107
P2_ADCF_Reset_Min_Max · 104
P2_ADCF_Set_Limit · 103
- B:** P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_Init · 76
P2_Burst_Read · 81
P2_Burst_Read_Index · 79
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 96
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- R:** P2_Read_ADCF · 109
P2_Read_ADCF32 · 117
P2_Read_ADCF4 · 111
P2_Read_ADCF4_Packed · 115
P2_Read_ADCF_SConv · 118
P2_Read_ADCF_SConv32 · 120
- S:** P2_Set_Average_Filter · 97
P2_Set_Gain · 121
P2_Set_LED · 5
P2_Start_ConvF · 122
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Mode · 17
P2_Sync_Stat · 19
- W:** P2_Wait_EOCF · 123

Aln-F-4/18 Rev. E

- A:** P2_ADCF · 98
P2_ADCF24 · 99
P2_ADCF_Mode · 100
P2_ADCF_Read_Limit · 102
P2_ADCF_Set_Limit · 103
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- R:** P2_Read_ADCF · 109
P2_Read_ADCF24 · 110
P2_Read_ADCF32 · 117
P2_Read_ADCF4 · 111
P2_Read_ADCF4_24B · 112
P2_Read_ADCF4_Packed · 115
P2_Read_ADCF_SConv · 118
P2_Read_ADCF_SConv24 · 119
P2_Read_ADCF_SConv32 · 120
- S:** P2_Set_LED · 5
P2_Start_ConvF · 122
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Mode · 17
P2_Sync_Stat · 19
- W:** P2_Wait_EOCF · 123

Aln-F-8/14 Rev. E

- A:** P2_ADCF · 98
P2_ADCF_Read_Limit · 102
P2_ADCF_Set_Limit · 103
- B:** P2_Burst_CRead_Pos_Unpacked1 · 68
P2_Burst_CRead_Pos_Unpacked2 · 70
P2_Burst_CRead_Pos_Unpacked4 · 72
P2_Burst_CRead_Pos_Unpacked8 · 74
P2_Burst_CRead_Unpacked1 · 60
P2_Burst_CRead_Unpacked2 · 62
P2_Burst_CRead_Unpacked4 · 64
P2_Burst_CRead_Unpacked8 · 66
P2_Burst_Init · 76
P2_Burst_Read · 81
P2_Burst_Read_Index · 79
P2_Burst_Read_Unpacked1 · 84
P2_Burst_Read_Unpacked2 · 86
P2_Burst_Read_Unpacked4 · 88
P2_Burst_Read_Unpacked8 · 90
P2_Burst_Reset · 92
P2_Burst_Start · 94
P2_Burst_Status · 95
P2_Burst_Stop · 96
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 10
P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- R:** P2_Read_ADCF · 109
P2_Read_ADCF32 · 117
P2_Read_ADCF4 · 111
P2_Read_ADCF4_Packed · 115
P2_Read_ADCF8 · 113
P2_Read_ADCF8_Packed · 116
- S:** P2_Set_Average_Filter · 97
P2_Set_LED · 5
P2_Sync_All · 13

Aln-F-8/16 Rev. E

- A:** P2_ADCF · 98
 P2_ADCF_Mode · 100
 P2_ADCF_Read_Limit · 102
 P2_ADCF_Read_Min_Max4 · 105
 P2_ADCF_Read_Min_Max8 · 107
 P2_ADCF_Reset_Min_Max · 104
 P2_ADCF_Set_Limit · 103
- B:** P2_Burst_CRead_Pos_Unpacked1 · 68
 P2_Burst_CRead_Pos_Unpacked2 · 70
 P2_Burst_CRead_Pos_Unpacked4 · 72
 P2_Burst_CRead_Pos_Unpacked8 · 74
 P2_Burst_CRead_Unpacked1 · 60
 P2_Burst_CRead_Unpacked2 · 62
 P2_Burst_CRead_Unpacked4 · 64
 P2_Burst_CRead_Unpacked8 · 66
 P2_Burst_Init · 76
 P2_Burst_Read · 81
 P2_Burst_Read_Index · 79
 P2_Burst_Read_Unpacked1 · 84
 P2_Burst_Read_Unpacked2 · 86
 P2_Burst_Read_Unpacked4 · 88
 P2_Burst_Read_Unpacked8 · 90
 P2_Burst_Reset · 92
 P2_Burst_Start · 94
 P2_Burst_Status · 95
 P2_Burst_Stop · 96
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 10
 P2_Event_Config · 9
 P2_Event_Enable · 8
 P2_Event_Read · 12
- R:** P2_Read_ADCF · 109
 P2_Read_ADCF32 · 117
 P2_Read_ADCF4 · 111
 P2_Read_ADCF4_Packed · 115
 P2_Read_ADCF8 · 113
 P2_Read_ADCF8_Packed · 116
 P2_Read_ADCF_SConv · 118
 P2_Read_ADCF_SConv32 · 120
- S:** P2_Set_Average_Filter · 97
 P2_Set_Gain · 121
 P2_Set_LED · 5
 P2_Start_ConvF · 122
 P2_Sync_All · 13
 P2_Sync_Enable · 15
 P2_Sync_Mode · 17
 P2_Sync_Stat · 19
- W:** P2_Wait_EOCF · 123

Aln-F-8/18 Rev. E

- A:** P2_ADCF · 98
 P2_ADCF24 · 99
 P2_ADCF_Mode · 100
 P2_ADCF_Read_Limit · 102
 P2_ADCF_Set_Limit · 103
- C:** P2_Check_LED · 4
- E:** P2_Event2_Config · 10
 P2_Event_Config · 9
 P2_Event_Enable · 8
 P2_Event_Read · 12
- R:** P2_Read_ADCF · 109
 P2_Read_ADCF24 · 110
 P2_Read_ADCF32 · 117
 P2_Read_ADCF4 · 111
 P2_Read_ADCF4_24B · 112
 P2_Read_ADCF4_Packed · 115
 P2_Read_ADCF8 · 113
 P2_Read_ADCF8_24B · 114
 P2_Read_ADCF8_Packed · 116
 P2_Read_ADCF_SConv · 118
 P2_Read_ADCF_SConv24 · 119
 P2_Read_ADCF_SConv32 · 120
- S:** P2_Set_LED · 5
 P2_Start_ConvF · 122
 P2_Sync_All · 13
 P2_Sync_Enable · 15
 P2_Sync_Mode · 17
 P2_Sync_Stat · 19
- W:** P2_Wait_EOCF · 123

AOut-1/16 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_DAC · 125
 - P2_DAC1_DIO · 137
 - P2_DAC_Ramp_Buffer_Free · 142
 - P2_DAC_Ramp_Status · 140
 - P2_DAC_Ramp_Stop · 143
 - P2_DAC_Ramp_Write · 138
 - P2_Digin_Edge · 155
 - P2_Digin_Fifo_Clear · 157
 - P2_Digin_Fifo_Enable · 158
 - P2_Digin_Fifo_Full · 159
 - P2_Digin_Fifo_Read · 160
 - P2_Digin_Fifo_Read_Fast · 162
 - P2_Digin_Fifo_Read_Timer · 164
 - P2_Digin_Long · 167
 - P2_Digout · 168
 - P2_Digout_Bits · 169
 - P2_Digout_Fifo_Clear · 171
 - P2_Digout_Fifo_Empty · 172
 - P2_Digout_Fifo_Enable · 173
 - P2_Digout_Fifo_Read_Timer · 174
 - P2_Digout_Fifo_Start · 175
 - P2_Digout_Fifo_Write · 176
 - P2_Digout_Long · 179
 - P2_Digout_Reset · 180
 - P2_Digout_Set · 181
 - P2_Dig_Fifo_Mode · 150
 - P2_Dig_Latch · 152
 - P2_Dig_Read_Latch · 153
 - P2_Dig_Write_Latch · 154
 - P2_Get_Digout_Long · 185
- E:** P2_Event_Config · 9
 - P2_Event_Enable · 8
 - P2_Event_Read · 12
- S:** P2_Set_LED · 5
 - P2_Start_DAC · 130
 - P2_Sync_All · 13
- W:** P2_Write_DAC · 131

AOut-4/16 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_DAC · 125
 - P2_DAC4 · 126
 - P2_DAC4_Packed · 127
- E:** P2_Event_Config · 9
 - P2_Event_Enable · 8
 - P2_Event_Read · 12
- S:** P2_Set_LED · 5
 - P2_Start_DAC · 130
 - P2_Sync_All · 13
 - P2_Sync_Enable · 15
 - P2_Sync_Stat · 19
- W:** P2_Write_DAC · 131
 - P2_Write_DAC32 · 136
 - P2_Write_DAC4 · 132
 - P2_Write_DAC4_Packed · 133

AOut-4/16-TiCo Rev. E

- C:** P2_Check_LED · 4
- D:** P2_DAC · 125
 - P2_DAC4 · 126
 - P2_DAC4_Packed · 127
- E:** P2_Event_Config · 9
 - P2_Event_Enable · 8
 - P2_Event_Read · 12
- S:** P2_Set_LED · 5
 - P2_Start_DAC · 130
 - P2_Sync_All · 13
 - P2_Sync_Enable · 15
 - P2_Sync_Stat · 19
- W:** P2_Write_DAC · 131
 - P2_Write_DAC32 · 136
 - P2_Write_DAC4 · 132
 - P2_Write_DAC4_Packed · 133

AOut-8/16 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_DAC · 125
 - P2_DAC4 · 126
 - P2_DAC4_Packed · 127
 - P2_DAC8 · 128
 - P2_DAC8_Packed · 129
- E:** P2_Event_Config · 9
 - P2_Event_Enable · 8
 - P2_Event_Read · 12
- S:** P2_Set_LED · 5
 - P2_Start_DAC · 130
 - P2_Sync_All · 13
 - P2_Sync_Enable · 15
 - P2_Sync_Stat · 19
- W:** P2_Write_DAC · 131
 - P2_Write_DAC32 · 136
 - P2_Write_DAC4 · 132
 - P2_Write_DAC4_Packed · 133
 - P2_Write_DAC8 · 134
 - P2_Write_DAC8_Packed · 135

AOut-8/16-TiCo Rev. E

- C:** P2_Check_LED · 4
- D:** P2_DAC · 125
P2_DAC4 · 126
P2_DAC4_Packed · 127
P2_DAC8 · 128
P2_DAC8_Packed · 129
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- S:** P2_Set_LED · 5
P2_Start_DAC · 130
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19
- W:** P2_Write_DAC · 131
P2_Write_DAC32 · 136
P2_Write_DAC4 · 132
P2_Write_DAC4_Packed · 133
P2_Write_DAC8 · 134
P2_Write_DAC8_Packed · 135

ARINC-429 Rev. E

- A:** ARINC_Create_Value32 · 366
ARINC_Split_Value32 · 370
P2_ARINC_Config_Receive · 361
P2_ARINC_Config_Transmit · 360
P2_ARINC_Read_Receive_Fifo · 369
P2_ARINC_Receive_Fifo_Empty · 368
P2_ARINC_Reset · 359
P2_ARINC_Set_Labels · 371
P2_ARINC_Transmit_Enable · 367
P2_ARINC_Transmit_Fifo_Empty · 364
P2_ARINC_Transmit_Fifo_Full · 363
P2_ARINC_Write_Transmit_Fifo · 365
- C:** P2_Check_LED · 4
- S:** P2_Set_LED · 5

CAN-2 Rev. E

- C:** CAN_Msg · 213
P2_CAN_Set_LED · 216
P2_Check_LED · 4
- E:** P2_CAN_Interrupt_Source · 214
P2_En_Interrupt · 217
P2_En_Receive · 218
P2_En_Transmit · 219
P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- G:** P2_Get_CAN_Reg · 220
- I:** P2_Init_CAN · 221
- R:** P2_Read_Msg · 222
P2_Read_Msg_Con · 224
- S:** P2_Set_CAN_Baudrate · 226
P2_Set_CAN_Reg · 227
P2_Set_LED · 5
- T:** P2_Transmit · 228
P2_Transmit_Status · 230

CAN-FD-2 Rev. E

- C:** P2_CANFD_Enable_Receive_Fifo · 237
P2_CANFD_Enable_Transmit_Event_Fifo · 241
P2_CANFD_Enable_Transmit_Fifo · 239
P2_CANFD_Enable_Transmit_Queue · 240
P2_CANFD_Get_BDIAG0 · 266
P2_CANFD_Get_BDIAG1 · 267
P2_CANFD_Get_BRS · 255
P2_CANFD_Get_DLC · 258
P2_CANFD_Get_ESI · 253
P2_CANFD_Get_FDF · 254
P2_CANFD_Get_Fifo_State · 247
P2_CANFD_Get_Header_Parts · 250
P2_CANFD_Get_ID · 252
P2_CANFD_Get_IDE · 257
P2_CANFD_Get_RTR · 256
P2_CANFD_Get_SEQ · 259
P2_CANFD_Get_TREC · 265
P2_CANFD_Init_Controller · 236
P2_CANFD_Init_Datatable · 234
P2_CANFD_Read_EFO · 260
P2_CANFD_Read_RMO · 249
P2_CANFD_Set_Baudrate_Data · 243
P2_CANFD_Set_Baudrate_Nominal · 242
P2_CANFD_Set_LED · 233
P2_CANFD_Set_Mode · 246
P2_CANFD_Set_Reg · 264
P2_CANFD_Set_SID11 · 244
P2_CANFD_Set_TDC · 245
P2_CANFD_Transmit_Msg · 263
P2_CANFD_Transmit_Multi_Msg · 264
P2_CANFD_Write_TMO · 261
P2_Check_LED · 4
- S:** P2_Set_LED · 5

CNT-D Rev. E

- C:** P2_Check_LED · 4
P2_Cnt_Clear · 187
P2_Cnt_Enable · 188
P2_Cnt_Get_PW · 192
P2_Cnt_Get_PW_HL · 193
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 194
P2_Cnt_Mode · 195
P2_Cnt_PW_Enable · 189
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read4 · 199
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Read_Latch4 · 202
P2_Cnt_Sync_Latch · 203
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- S:** P2_Set_LED · 5
P2_SSI_Mode · 204
P2_SSI_Read · 205
P2_SSI_Read2 · 206
P2_SSI_Set_Bits · 207
P2_SSI_Set_Clock · 208
P2_SSI_Set_Delay · 209
P2_SSI_Start · 210
P2_SSI_Status · 211
P2_Sync_All · 13

CNT-I Rev. E

- C:** P2_Check_LED · 4
P2_Cnt_Clear · 187
P2_Cnt_Enable · 188
P2_Cnt_Get_PW · 192
P2_Cnt_Get_PW_HL · 193
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 194
P2_Cnt_Mode · 195
P2_Cnt_PW_Enable · 189
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read4 · 199
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Read_Latch4 · 202
P2_Cnt_Sync_Latch · 203
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- S:** P2_Set_LED · 5
P2_Sync_All · 13

CNT-T Rev. E

- C:** P2_Check_LED · 4
P2_Cnt_Clear · 187
P2_Cnt_Enable · 188
P2_Cnt_Get_PW · 192
P2_Cnt_Get_PW_HL · 193
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 194
P2_Cnt_Mode · 195
P2_Cnt_PW_Enable · 189
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read4 · 199
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Read_Latch4 · 202
P2_Cnt_Sync_Latch · 203
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- S:** P2_Set_LED · 5
P2_Sync_All · 13

Comp-16 Rev. E

- C:** P2_Check_LED · 4
P2_Comp_Filter_Init · 147
P2_Comp_Init · 145
P2_Comp_Set · 148
P2_Comp_Set_Voltage · 149
- D:** P2_Digin_Edge · 155
P2_Digin_Fifo_Clear · 157
P2_Digin_Fifo_Enable · 158
P2_Digin_Fifo_Full · 159
P2_Digin_Fifo_Read · 160
P2_Digin_Fifo_Read_Fast · 162
P2_Digin_Fifo_Read_Timer · 164
P2_Digin_Long · 167
P2_Dig_Latch · 152
P2_Dig_Read_Latch · 153
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- S:** P2_Set_LED · 5

CPU-T10

- C:** Calc_Processdelay · 6
Calc_TicksToNs · 7

CPU-T11

- C:** Calc_Processdelay · 6
Calc_TicksToNs · 7
CPU_Digin · 22
CPU_Digout · 23
CPU_Dig_IO_Config · 24
CPU_Event_Config · 25
P2_Check_LED · 4
- S:** P2_Set_LED · 5

CPU-T12

- C:** Calc_Processdelay · 6
Calc_TicksToNs · 7
CPU_Digin · 22
CPU_Digout · 23
CPU_Dig_IO_Config · 24
CPU_Event_Config · 25
P2_Check_LED · 4
- S:** P2_Set_LED · 5

CPU-T9

- C:** Calc_Processdelay · 6
Calc_TicksToNs · 7

DIO-32 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 155
P2_Digin_Fifo_Clear · 157
P2_Digin_Fifo_Enable · 158
P2_Digin_Fifo_Full · 159
P2_Digin_Fifo_Read · 160
P2_Digin_Fifo_Read_Fast · 162
P2_Digin_Fifo_Read_Timer · 164
P2_Digin_Long · 167
P2_Digout · 168
P2_Digout_Bits · 169
P2_Digout_Long · 179
P2_Digout_Reset · 180
P2_Digout_Set · 181
P2_DigProg · 182
P2_Dig_Latch · 152
P2_Dig_Read_Latch · 153
P2_Dig_Write_Latch · 154
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- G:** P2_Get_Digout_Long · 185
- S:** P2_Set_LED · 5
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19

DIO-32-TiCo Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 155
P2_Digin_Fifo_Clear · 157
P2_Digin_Fifo_Enable · 158
P2_Digin_Fifo_Full · 159
P2_Digin_Fifo_Read · 160
P2_Digin_Fifo_Read_Fast · 162
P2_Digin_Fifo_Read_Timer · 164
P2_Digin_Filter_Init · 165
P2_Digin_Long · 167
P2_Digout · 168
P2_Digout_Bits · 169
P2_Digout_Fifo_Clear (Rev. E03) · 171
P2_Digout_Fifo_Empty (Rev. E03) · 172
P2_Digout_Fifo_Enable (Rev. E03) · 173
P2_Digout_Fifo_Read_Timer (Rev. E03) · 174
P2_Digout_Fifo_Start (Rev. E03) · 175
P2_Digout_Fifo_Write (Rev. E03) · 176
P2_Digout_Long · 179
P2_Digout_Reset · 180
P2_Digout_Set · 181
P2_DigProg · 182
P2_Dig_Fifo_Mode (Rev. E03) · 150
P2_Dig_Latch · 152
P2_Dig_Read_Latch · 153
P2_Dig_Write_Latch · 154
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- G:** P2_Get_Digout_Long · 185
- S:** P2_Set_LED · 5
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19

DIO-32-TiCo2 Rev. E

- C:** [P2_Check_LED · 4](#)
- D:** [P2_Digin_Edge · 155](#)
 - [P2_Digin_Fifo_Clear · 157](#)
 - [P2_Digin_Fifo_Enable · 158](#)
 - [P2_Digin_Fifo_Full · 159](#)
 - [P2_Digin_Fifo_Read · 160](#)
 - [P2_Digin_Fifo_Read_Fast · 162](#)
 - [P2_Digin_Fifo_Read_Timer · 164](#)
 - [P2_Digin_Filter_Init · 165](#)
 - [P2_Digin_Long · 167](#)
 - [P2_Digout · 168](#)
 - [P2_Digout_Bits · 169](#)
 - [P2_Digout_Fifo_Clear · 171](#)
 - [P2_Digout_Fifo_Empty · 172](#)
 - [P2_Digout_Fifo_Enable · 173](#)
 - [P2_Digout_Fifo_Read_Timer · 174](#)
 - [P2_Digout_Fifo_Start · 175](#)
 - [P2_Digout_Fifo_Write · 176](#)
 - [P2_Digout_Long · 179](#)
 - [P2_Digout_Reset · 180](#)
 - [P2_Digout_Set · 181](#)
 - [P2_DigProg · 182](#)
 - [P2_DigProg_Set_IO_Level · 184](#)
 - [P2_Dig_Fifo_Mode · 150](#)
 - [P2_Dig_Latch · 152](#)
 - [P2_Dig_Read_Latch · 153](#)
 - [P2_Dig_Write_Latch · 154](#)
- E:** [P2_Event_Config · 9](#)
 - [P2_Event_Enable · 8](#)
 - [P2_Event_Read · 12](#)
- G:** [P2_Get_Digout_Long · 185](#)
- S:** [P2_Set_LED · 5](#)
 - [P2_Sync_All · 13](#)
 - [P2_Sync_Enable · 15](#)
 - [P2_Sync_Stat · 19](#)

DIO-32/1-TiCo Rev. E

- C:** [P2_Check_LED · 4](#)
- D:** [P2_Digin_Edge · 155](#)
 - [P2_Digin_Fifo_Clear · 157](#)
 - [P2_Digin_Fifo_Enable · 158](#)
 - [P2_Digin_Fifo_Full · 159](#)
 - [P2_Digin_Fifo_Read · 160](#)
 - [P2_Digin_Fifo_Read_Fast · 162](#)
 - [P2_Digin_Fifo_Read_Timer · 164](#)
 - [P2_Digin_Filter_Init · 165](#)
 - [P2_Digin_Long · 167](#)
 - [P2_Digout · 168](#)
 - [P2_Digout_Bits · 169](#)
 - [P2_Digout_Fifo_Clear · 171](#)
 - [P2_Digout_Fifo_Empty · 172](#)
 - [P2_Digout_Fifo_Enable · 173](#)
 - [P2_Digout_Fifo_Read_Timer · 174](#)
 - [P2_Digout_Fifo_Start · 175](#)
 - [P2_Digout_Fifo_Write · 176](#)
 - [P2_Digout_Long · 179](#)
 - [P2_Digout_Reset · 180](#)
 - [P2_Digout_Set · 181](#)
 - [P2_DigProg · 182](#)
 - [P2_DigProg_Bits · 183](#)
 - [P2_Dig_Fifo_Mode · 150](#)
 - [P2_Dig_Latch · 152](#)
 - [P2_Dig_Read_Latch · 153](#)
 - [P2_Dig_Write_Latch · 154](#)
- E:** [P2_Event_Config · 9](#)
 - [P2_Event_Enable · 8](#)
 - [P2_Event_Read · 12](#)
- G:** [P2_Get_Digout_Long · 185](#)
- S:** [P2_Set_LED · 5](#)
 - [P2_Sync_All · 13](#)
 - [P2_Sync_Enable · 15](#)
 - [P2_Sync_Stat · 19](#)

DIO-8-D12 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 155
 - P2_Digin_Fifo_Clear · 157
 - P2_Digin_Fifo_Enable · 158
 - P2_Digin_Fifo_Full · 159
 - P2_Digin_Fifo_Read · 160
 - P2_Digin_Fifo_Read_Fast · 162
 - P2_Digin_Fifo_Read_Timer · 164
 - P2_Digin_Filter_Init · 165
 - P2_Digin_Long · 167
 - P2_Digout · 168
 - P2_Digout_Bits · 169
 - P2_Digout_Fifo_Clear · 171
 - P2_Digout_Fifo_Empty · 172
 - P2_Digout_Fifo_Enable · 173
 - P2_Digout_Fifo_Read_Timer · 174
 - P2_Digout_Fifo_Start · 175
 - P2_Digout_Fifo_Write · 176
 - P2_Digout_Long · 179
 - P2_Digout_Reset · 180
 - P2_Digout_Set · 181
 - P2_DigProg · 182
 - P2_DigProg_Bits · 183
 - P2_Dig_Fifo_Mode · 150
 - P2_Dig_Latch · 152
 - P2_Dig_Read_Latch · 153
 - P2_Dig_Write_Latch · 154
- E:** P2_Event_Config · 9
 - P2_Event_Enable · 8
 - P2_Event_Read · 12
- G:** P2_Get_Digout_Long · 185
- S:** P2_Set_LED · 5
 - P2_Sync_All · 13
 - P2_Sync_Enable · 15
 - P2_Sync_Stat · 19

EtherCAT-SL Rev. E

- C:** P2_Check_LED · 4
- E:** P2_ECAC_Get_State · 374
 - P2_ECAC_Get_Version · 373
 - P2_ECAC_Read_Data_16F · 380
 - P2_ECAC_Read_Data_16L · 378
 - P2_ECAC_Set_Mode · 377
 - P2_ECAC_Write_Data_16F · 381
 - P2_ECAC_Write_Data_16L · 379
- I:** P2_ECAC_Init · 375
- S:** P2_Set_LED · 5

EtherCAT-SL-40 Rev. E

- C:** P2_Check_LED · 4
- I:** P2_Init_EtherCAT · 382
- R:** P2_Run_EtherCAT · 384
- S:** P2_Set_LED · 5

FlexRay-2 Rev. E

- C:** P2_Check_LED · 4
- F:** P2_FlexRay_Get_Version · 386
 - P2_FlexRay_Init · 387
 - P2_FlexRay_Read_Word · 388
 - P2_FlexRay_Reset · 389
 - P2_FlexRay_Set_LED · 390
 - P2_FlexRay_Write_Word · 391
- S:** P2_Set_LED · 5

LIN-2 Rev. E

- C:** P2_Check_LED · 4
- L:** P2_LIN_Ch_Read_Cnt · 278
 - P2_LIN_Get_Version · 275
 - P2_LIN_Init · 270
 - P2_LIN_Init_Apply · 273
 - P2_LIN_Init_Write · 272
 - P2_LIN_Msg_Read_Status · 279
 - P2_LIN_Msg_Transmit · 281
 - P2_LIN_Msg_Write · 280
 - P2_LIN_Read_Dat · 276
 - P2_LIN_Reset · 274
 - P2_LIN_Set_LED · 282
- S:** P2_Set_LED · 5

LS-2 Rev. E

- C:** P2_Check_LED · 4
- S:** P2_Set_LED · 5

MIL-1553 Rev. E

- C:** P2_Check_LED · 4
- M:** P2_MIL_Get_Register · 357
 - P2_MIL_Reset · 349
 - P2_MIL_Set_LED · 355
 - P2_MIL_Set_Register · 356
 - P2_MIL_SMT_Init · 350
 - P2_MIL_SMT_Message_Read · 351
 - P2_MIL_SMT_Set_All_Filters · 353
 - P2_MIL_SMT_Set_Filter · 354
- S:** P2_Set_LED · 5

MIO-4 Rev. E

- A:** [P2_ADC · 37](#)
[P2_ADC24 · 38](#)
[P2_ADC_Read_Limit · 39](#)
[P2_ADC_Set_Limit · 41](#)
- C:** [P2_Check_LED · 4](#)
- D:** [P2_DAC · 125](#)
[P2_DAC4 · 126](#)
[P2_DAC4_Packed · 127](#)
- E:** [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- M:** [P2_MIO_Digin_Long · 30](#)
[P2_MIO_Digout · 31](#)
[P2_MIO_Digout_Long · 32](#)
[P2_MIO_DigProg · 33](#)
[P2_MIO_Dig_Latch · 27](#)
[P2_MIO_Dig_Read_Latch · 28](#)
[P2_MIO_Dig_Write_Latch · 29](#)
[P2_MIO_Get_Digout_Long · 34](#)
- R:** [P2_Read_ADC · 42](#)
[P2_Read_ADC24 · 43](#)
[P2_Read_ADC_SConv · 44](#)
[P2_Read_ADC_SConv24 · 45](#)
- S:** [P2_Seq_Init · 47](#)
[P2_Seq_Read · 50](#)
[P2_Seq_Read24 · 51](#)
[P2_Seq_Read_Packed · 53](#)
[P2_Seq_Start · 54](#)
[P2_Seq_Wait · 55](#)
[P2_Set_LED · 5](#)
[P2_Set_Mux · 56](#)
[P2_SE_Diff · 46](#)
[P2_Start_Conv · 57](#)
[P2_Start_DAC · 130](#)
[P2_Sync_All · 13](#)
[P2_Sync_Enable · 15](#)
[P2_Sync_Stat · 19](#)
- W:** [P2_Wait_EOC · 58](#)
[P2_Wait_Mux · 59](#)
[P2_Write_DAC · 131](#)
[P2_Write_DAC32 · 136](#)
[P2_Write_DAC4 · 132](#)
[P2_Write_DAC4_Packed · 133](#)

MIO-4-ET1 Rev. E

- A:** P2_ADC · 37
P2_ADC24 · 38
P2_ADC_Read_Limit · 39
P2_ADC_Set_Limit · 41
- C:** P2_Check_LED · 4
P2_Cnt_Clear · 187
P2_Cnt_Enable · 188
P2_Cnt_Get_PW · 192
P2_Cnt_Get_PW_HL · 193
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 194
P2_Cnt_Mode · 195
P2_Cnt_PW_Enable · 189
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Sync_Latch · 203
- D:** P2_DAC · 125
P2_DAC4 · 126
P2_DAC4_Packed · 127
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- M:** P2_MIO_Digin_Long · 30
P2_MIO_Digout · 31
P2_MIO_Digout_Long · 32
P2_MIO_DigProg · 33
P2_MIO_Dig_Latch · 27
P2_MIO_Dig_Read_Latch · 28
P2_MIO_Dig_Write_Latch · 29
P2_MIO_Get_Digout_Long · 34
- R:** P2_Read_ADC · 42
P2_Read_ADC24 · 43
P2_Read_ADC_SConv · 44
P2_Read_ADC_SConv24 · 45
- S:** P2_Seq_Init · 47
P2_Seq_Read · 50
P2_Seq_Read24 · 51
P2_Seq_Read_Packed · 53
P2_Seq_Start · 54
P2_Seq_Wait · 55
P2_Set_LED · 5
P2_Set_Mux · 56
P2_SE_Diff · 46
P2_SSI_Mode · 204
P2_SSI_Read · 205
P2_SSI_Set_Bits · 207
P2_SSI_Set_Clock · 208
P2_SSI_Set_Delay · 209
P2_SSI_Start · 210
P2_SSI_Status · 211
P2_Start_Conv · 57
P2_Start_DAC · 130
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19

- W:** P2_Wait_EOC · 58
P2_Wait_Mux · 59
P2_Write_DAC · 131
P2_Write_DAC32 · 136
P2_Write_DAC4 · 132
P2_Write_DAC4_Packed · 133

MIO-D12 Rev. E

- C:** P2_Check_LED · 4
P2_Cnt_Clear · 187
P2_Cnt_Enable · 188
P2_Cnt_Get_PW · 192
P2_Cnt_Get_PW_HL · 193
P2_Cnt_Get_Status · 190
P2_Cnt_Latch · 194
P2_Cnt_Mode · 195
P2_Cnt_PW_Enable · 189
P2_Cnt_PW_Latch · 197
P2_Cnt_Read · 198
P2_Cnt_Read_Int_Register · 200
P2_Cnt_Read_Latch · 201
P2_Cnt_Sync_Latch · 203
- D:** P2_Digin_Edge · 155
P2_Digin_Fifo_Clear · 157
P2_Digin_Fifo_Enable · 158
P2_Digin_Fifo_Full · 159
P2_Digin_Fifo_Read · 160
P2_Digin_Fifo_Read_Fast · 162
P2_Digin_Fifo_Read_Timer · 164
P2_Digout_Fifo_Clear · 171
P2_Digout_Fifo_Empty · 172
P2_Digout_Fifo_Enable · 173
P2_Digout_Fifo_Read_Timer · 174
P2_Digout_Fifo_Start · 175
P2_Digout_Fifo_Write · 176
P2_Dig_Fifo_Mode · 150
- E:** P2_Event_Config · 9
P2_Event_Enable · 8
P2_Event_Read · 12
- M:** P2_MIO_Digin_Long · 30
P2_MIO_Digout · 31
P2_MIO_Digout_Long · 32
P2_MIO_Dig_Latch · 27
P2_MIO_Dig_Read_Latch · 28
P2_MIO_Dig_Write_Latch · 29
P2_MIO_Get_Digout_Long · 34
- S:** P2_Set_LED · 5
P2_SSI_Mode · 204
P2_SSI_Read · 205
P2_SSI_Set_Bits · 207
P2_SSI_Set_Clock · 208
P2_SSI_Set_Delay · 209
P2_SSI_Start · 210
P2_SSI_Status · 211
P2_Sync_All · 13
P2_Sync_Enable · 15
P2_Sync_Stat · 19

OPT-16 Rev. E

- C: [P2_Check_LED · 4](#)
- D: [P2_Digin_Edge · 155](#)
[P2_Digin_Fifo_Clear · 157](#)
[P2_Digin_Fifo_Enable · 158](#)
[P2_Digin_Fifo_Full · 159](#)
[P2_Digin_Fifo_Read · 160](#)
[P2_Digin_Fifo_Read_Fast · 162](#)
[P2_Digin_Fifo_Read_Timer · 164](#)
[P2_Digin_Long · 167](#)
[P2_Dig_Latch · 152](#)
[P2_Dig_Read_Latch · 153](#)
- E: [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- S: [P2_Set_LED · 5](#)
[P2_Sync_All · 13](#)
[P2_Sync_Enable · 15](#)
[P2_Sync_Stat · 19](#)

OPT-32-24V Rev. E

- C: [P2_Check_LED · 4](#)
- D: [P2_Digin_Edge · 155](#)
[P2_Digin_Fifo_Clear · 157](#)
[P2_Digin_Fifo_Enable · 158](#)
[P2_Digin_Fifo_Full · 159](#)
[P2_Digin_Fifo_Read · 160](#)
[P2_Digin_Fifo_Read_Fast · 162](#)
[P2_Digin_Fifo_Read_Timer · 164](#)
[P2_Digin_Long · 167](#)
[P2_Dig_Latch · 152](#)
[P2_Dig_Read_Latch · 153](#)
- E: [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- S: [P2_Set_LED · 5](#)
[P2_Sync_All · 13](#)
[P2_Sync_Enable · 15](#)
[P2_Sync_Stat · 19](#)

Profi-IRT-CU Rev. E

- C: [P2_Check_LED · 4](#)
- S: [P2_Set_LED · 5](#)

Profi-IRT-CU-40 Rev. E

- I: [P2_Init_ProfinetIO · 345](#)
- R: [P2_Run_ProfinetIO · 347](#)

Profi-IRT-FO Rev. E

- C: [P2_Check_LED · 4](#)
- S: [P2_Set_LED · 5](#)

Profi-IRT-FO-40 Rev. E

- I: [P2_Init_ProfinetIO · 345](#)
- R: [P2_Run_ProfinetIO · 347](#)

Profi-SL Rev. E

- C: [P2_Check_LED · 4](#)
- I: [P2_Init_Profibus · 339](#)
- R: [P2_Run_Profibus · 341](#)
- S: [P2_Set_LED · 5](#)

Profi-SL-40 Rev. E

- C: [P2_Check_LED · 4](#)
- I: [P2_Init_Profibus_M40 · 342](#)
- R: [P2_Run_Profibus_M40 · 344](#)
- S: [P2_Set_LED · 5](#)

PWM-16(-I) Rev. E

- C: [P2_Check_LED · 4](#)
- E: [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- P: [P2_PWM_Enable · 284](#)
[P2_PWM_Get_Status · 285](#)
[P2_PWM_Init · 286](#)
[P2_PWM_Latch · 288](#)
[P2_PWM_Reset · 289](#)
[P2_PWM_Standby_Value · 290](#)
[P2_PWM_Write_Latch · 291](#)
[P2_PWM_Write_Latch_Block · 292](#)
- S: [P2_Set_LED · 5](#)
[P2_Sync_All · 13](#)
[P2_Sync_Enable · 15](#)
[P2_Sync_Stat · 19](#)

REL-16 Rev. E

- C: [P2_Check_LED · 4](#)
- D: [P2_Digout · 168](#)
[P2_Digout_Bits · 169](#)
[P2_Digout_Long · 179](#)
[P2_Digout_Reset · 180](#)
[P2_Digout_Set · 181](#)
[P2_Dig_Latch · 152](#)
[P2_Dig_Write_Latch · 154](#)
- E: [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- G: [P2_Get_Digout_Long · 185](#)
- S: [P2_Set_LED · 5](#)
[P2_Sync_All · 13](#)
[P2_Sync_Enable · 15](#)
[P2_Sync_Stat · 19](#)

RS422-4 Rev. E

- C:** [P2_Check_LED · 4](#)
[P2_Check_Shift_Reg · 325](#)
- E:** [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- G:** [P2_Get_RS · 326](#)
- R:** [P2_Read_Fifo · 327](#)
[P2_RS485_Send · 331](#)
[P2_RS_Init · 328](#)
[P2_RS_Reset · 330](#)
[P2_RS_Set_LED · 332](#)
- S:** [P2_Set_LED · 5](#)
[P2_Set_RS · 333](#)
- W:** [P2_Write_Fifo · 334](#)
[P2_Write_Fifo_Full · 335](#)

RSxxx-2 Rev. E

- C:** [P2_Check_LED · 4](#)
[P2_Check_Shift_Reg · 325](#)
- E:** [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- G:** [P2_Get_RS · 326](#)
- R:** [P2_Read_Fifo · 327](#)
[P2_RS485_Send · 331](#)
[P2_RS_Init · 328](#)
[P2_RS_Reset · 330](#)
[P2_RS_Set_LED · 332](#)
- S:** [P2_Set_LED · 5](#)
[P2_Set_RS · 333](#)
- W:** [P2_Write_Fifo · 334](#)
[P2_Write_Fifo_Full · 335](#)

RSxxx-4 Rev. E

- C:** [P2_Check_LED · 4](#)
[P2_Check_Shift_Reg · 325](#)
- E:** [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
- G:** [P2_Get_RS · 326](#)
- R:** [P2_Read_Fifo · 327](#)
[P2_RS485_Send · 331](#)
[P2_RS_Init · 328](#)
[P2_RS_Reset · 330](#)
[P2_RS_Set_LED · 332](#)
- S:** [P2_Set_LED · 5](#)
[P2_Set_RS · 333](#)
- W:** [P2_Write_Fifo · 334](#)
[P2_Write_Fifo_Full · 335](#)

RTD-8 Rev. E

- C:** [P2_Check_LED · 4](#)
- R:** [P2_RTD_Channel_Config · 295](#)
[P2_RTD_Config · 297](#)
[P2_RTD_Convert · 298](#)
[P2_RTD_Read · 299](#)
[P2_RTD_Read8 · 300](#)
[P2_RTD_Start · 301](#)
[P2_RTD_Status · 303](#)
- S:** [P2_Set_LED · 5](#)

SENT-4 Rev. E

- C:** [P2_Check_LED · 4](#)
- S:** [P2_SENT_Check_Latch · 416](#)
[P2_SENT_Clear_Serial_Message_Array · 409](#)
[P2_SENT_Command_Ready · 396](#)
[P2_SENT_Get_ChannelState · 397](#)
[P2_SENT_Get_ClockTick · 398](#)
[P2_SENT_Get_Fast_Channel1 · 401](#)
[P2_SENT_Get_Fast_Channel2 · 403](#)
[P2_SENT_Get_Fast_Channel_CRC_OK · 400](#)
[P2_SENT_Get_Latch_Data · 417](#)
[P2_SENT_Get_Msg_Counter · 395](#)
[P2_SENT_Get_PulseCount · 399](#)
[P2_SENT_Get_Serial_Message_Array · 407](#)
[P2_SENT_Get_Serial_Message_CRC_OK · 404](#)
[P2_SENT_Get_Serial_Message_Data · 406](#)
[P2_SENT_Get_Serial_Message_Id · 405](#)
[P2_SENT_Get_Version · 394](#)
[P2_SENT_Init · 393](#)
[P2_SENT_Request_Latch · 415](#)
[P2_SENT_Set_ClockTick · 412](#)
[P2_SENT_Set_CRC_Implementation · 410](#)
[P2_SENT_Set_Detection · 411](#)
[P2_SENT_Set_PulseCount · 413](#)
[P2_SENT_Set_Sensor_Type · 414](#)
[P2_Set_LED · 5](#)

SENT-4-Out Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 155
P2_Digin_Long · 167
P2_Digout · 168
P2_Digout_Bits · 169
P2_Digout_Long · 179
P2_Digout_Reset · 180
P2_Digout_Set · 181
P2_Dig_Latch · 152
P2_Dig_Read_Latch · 153
P2_Dig_Write_Latch · 154
- G:** P2_Get_Digout_Long · 185
- S:** P2_SENT_Command_Ready · 396
P2_SENT_Config_Output · 421
P2_SENT_Config_Serial_Messages · 422
P2_SENT_Enable_Channel · 423
P2_SENT_Fifo_Clear · 431
P2_SENT_Fifo_Empty · 430
P2_SENT_Get_Msg_Counter · 395
P2_SENT_Get_Version · 394
P2_SENT_Init · 393
P2_SENT_Invert_Channel · 424
P2_SENT_Set_Fast_Channel1 · 426
P2_SENT_Set_Fast_Channel2 · 427
P2_SENT_Set_Fifo · 432
P2_SENT_Set_Output_Mode · 420
P2_SENT_Set_Reserved_Bits · 425
P2_SENT_Set_Serial_Message_Data · 429
P2_SENT_Set_Serial_Message_Pattern · 428
P2_Set_LED · 5

SENT-6 Rev. E

- C:** P2_Check_LED · 4
- D:** P2_Digin_Edge · 155
P2_Digin_Long · 167
P2_Digout · 168
P2_Digout_Bits · 169
P2_Digout_Long · 179
P2_Digout_Reset · 180
P2_Digout_Set · 181
P2_Dig_Latch · 152
P2_Dig_Read_Latch · 153
P2_Dig_Write_Latch · 154
P2_Get_Digout_Long · 185
- S:** P2_SENT_Check_Latch · 416
P2_SENT_Clear_Serial_Message_Array · 409
P2_SENT_Command_Ready · 396
P2_SENT_Get_ChannelState · 397
P2_SENT_Get_ClockTick · 398
P2_SENT_Get_Fast_Channel1 · 401
P2_SENT_Get_Fast_Channel2 · 403
P2_SENT_Get_Fast_Channel_CRC_OK · 400
P2_SENT_Get_Latch_Data · 417
P2_SENT_Get_Msg_Counter · 395
P2_SENT_Get_PulseCount · 399
P2_SENT_Get_Serial_Message_Array · 407
P2_SENT_Get_Serial_Message_CRC_OK ·

404

- P2_SENT_Get_Serial_Message_Data · 406
- P2_SENT_Get_Serial_Message_Id · 405
- P2_SENT_Get_Version · 394
- P2_SENT_Init · 393
- P2_SENT_Request_Latch · 415
- P2_SENT_Set_ClockTick · 412
- P2_SENT_Set_CRC_Implementation · 410
- P2_SENT_Set_Detection · 411
- P2_SENT_Set_PulseCount · 413
- P2_SENT_Set_Sensor_Type · 414
- P2_Set_LED · 5

SG-4/18 Rev. E

- C:** P2_Check_LED · 4
- R:** P2_SG_Convert · 318
P2_SG_Init · 319
P2_SG_Mode · 313
P2_SG_Read · 317
P2_SG_Set_Gain · 322
P2_SG_Start · 315
P2_SG_Wait · 316
P2_SG_Zero · 321
- S:** P2_Set_LED · 5

SPI-2-D Rev. E

C: [P2_Check_LED · 4](#)
D: [P2_Digin_Edge · 155](#)
[P2_Digin_Fifo_Clear · 157](#)
[P2_Digin_Fifo_Enable · 158](#)
[P2_Digin_Fifo_Full · 159](#)
[P2_Digin_Fifo_Read · 160](#)
[P2_Digin_Fifo_Read_Fast · 162](#)
[P2_Digin_Fifo_Read_Timer · 164](#)
[P2_Digin_Long · 167](#)
[P2_Digout · 168](#)
[P2_Digout_Bits · 169](#)
[P2_Digout_Fifo_Clear · 171](#)
[P2_Digout_Fifo_Empty · 172](#)
[P2_Digout_Fifo_Enable · 173](#)
[P2_Digout_Fifo_Read_Timer · 174](#)
[P2_Digout_Fifo_Start · 175](#)
[P2_Digout_Fifo_Write · 176](#)
[P2_Digout_Long · 179](#)
[P2_Digout_Reset · 180](#)
[P2_Digout_Set · 181](#)
[P2_DigProg · 182](#)
[P2_DigProg_Bits · 183](#)
[P2_Dig_Fifo_Mode · 150](#)
[P2_Dig_Latch · 152](#)
[P2_Dig_Read_Latch · 153](#)
[P2_Dig_Write_Latch · 154](#)
E: [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
G: [P2_Get_Digout_Long · 185](#)
S: [P2_Set_LED · 5](#)
[P2_SPI_Config · 435](#)
[P2_SPI_Master_Config · 437](#)
[P2_SPI_Master_Get_Static_Input · 447](#)
[P2_SPI_Master_Get_Value32 · 445](#)
[P2_SPI_Master_Get_Value64 · 446](#)
[P2_SPI_Master_Set_Clk_Wait · 448](#)
[P2_SPI_Master_Set_Value32 · 441](#)
[P2_SPI_Master_Set_Value64 · 442](#)
[P2_SPI_Master_Start · 443](#)
[P2_SPI_Master_Status · 444](#)
[P2_SPI_Mode · 434](#)
[P2_SPI_Slave_Clear_Fifo · 457](#)
[P2_SPI_Slave_Config · 450](#)
[P2_SPI_Slave_InFifo_Full · 454](#)
[P2_SPI_Slave_InFifo_Read · 455](#)
[P2_SPI_Slave_OutFifo_Empty · 453](#)
[P2_SPI_Slave_OutFifo_Write · 451](#)
[P2_Sync_All · 13](#)
[P2_Sync_Enable · 15](#)
[P2_Sync_Stat · 19](#)

SPI-2-T Rev. E

C: [P2_Check_LED · 4](#)
D: [P2_Digin_Edge · 155](#)
[P2_Digin_Fifo_Clear · 157](#)
[P2_Digin_Fifo_Enable · 158](#)
[P2_Digin_Fifo_Full · 159](#)
[P2_Digin_Fifo_Read · 160](#)
[P2_Digin_Fifo_Read_Fast · 162](#)
[P2_Digin_Fifo_Read_Timer · 164](#)
[P2_Digin_Long · 167](#)
[P2_Digout · 168](#)
[P2_Digout_Bits · 169](#)
[P2_Digout_Fifo_Clear · 171](#)
[P2_Digout_Fifo_Empty · 172](#)
[P2_Digout_Fifo_Enable · 173](#)
[P2_Digout_Fifo_Read_Timer · 174](#)
[P2_Digout_Fifo_Start · 175](#)
[P2_Digout_Fifo_Write · 176](#)
[P2_Digout_Long · 179](#)
[P2_Digout_Reset · 180](#)
[P2_Digout_Set · 181](#)
[P2_DigProg · 182](#)
[P2_Dig_Fifo_Mode · 150](#)
[P2_Dig_Latch · 152](#)
[P2_Dig_Read_Latch · 153](#)
[P2_Dig_Write_Latch · 154](#)
E: [P2_Event_Config · 9](#)
[P2_Event_Enable · 8](#)
[P2_Event_Read · 12](#)
G: [P2_Get_Digout_Long · 185](#)
S: [P2_Set_LED · 5](#)
[P2_SPI_Config · 435](#)
[P2_SPI_Master_Config · 437](#)
[P2_SPI_Master_Get_Static_Input · 447](#)
[P2_SPI_Master_Get_Value32 · 445](#)
[P2_SPI_Master_Get_Value64 · 446](#)
[P2_SPI_Master_Set_Clk_Wait · 448](#)
[P2_SPI_Master_Set_Value32 · 441](#)
[P2_SPI_Master_Set_Value64 · 442](#)
[P2_SPI_Master_Start · 443](#)
[P2_SPI_Master_Status · 444](#)
[P2_SPI_Mode · 434](#)
[P2_SPI_Slave_Clear_Fifo · 457](#)
[P2_SPI_Slave_Config · 450](#)
[P2_SPI_Slave_InFifo_Full · 454](#)
[P2_SPI_Slave_InFifo_Read · 455](#)
[P2_SPI_Slave_OutFifo_Empty · 453](#)
[P2_SPI_Slave_OutFifo_Write · 451](#)
[P2_Sync_All · 13](#)
[P2_Sync_Enable · 15](#)
[P2_Sync_Stat · 19](#)

TC-8-ISO Rev. E

- S: [P2_Sync_All · 13](#)
- T: [P2_TC_Latch · 304](#)
 - [P2_TC_Read_Latch · 305](#)
 - [P2_TC_Read_Latch4 · 307](#)
 - [P2_TC_Read_Latch8 · 309](#)
 - [P2_TC_Set_Rate · 311](#)

TRA-16 Rev. E

- C: [P2_Check_LED · 4](#)
- D: [P2_Digout · 168](#)
 - [P2_Digout_Bits · 169](#)
 - [P2_Digout_Long · 179](#)
 - [P2_Digout_Reset · 180](#)
 - [P2_Digout_Set · 181](#)
 - [P2_Dig_Latch · 152](#)
 - [P2_Dig_Write_Latch · 154](#)
- E: [P2_Event_Config · 9](#)
 - [P2_Event_Enable · 8](#)
 - [P2_Event_Read · 12](#)
- G: [P2_Get_Digout_Long · 185](#)
- S: [P2_Set_LED · 5](#)
 - [P2_Sync_All · 13](#)
 - [P2_Sync_Enable · 15](#)
 - [P2_Sync_Stat · 19](#)

A.3 Thematische Befehlsübersicht

Die Befehle sind in die folgenden Themengruppen aufgeteilt. Innerhalb der Themengruppen sind die Befehle alphabetisch sortiert.

Analoge Ausgänge:	Seite A-24
Analoge Eingänge (Fast-ADC):	Seite A-25
Analoge Eingänge (Fast-ADC, Burst):	Seite A-26
Analoge Eingänge (Multiplexer):	Seite A-26
Bustyp: ARINC 429:	Seite A-27
Bustyp: CAN:	Seite A-27
Bustyp: CAN-FD:	Seite A-28
Bustyp: EtherCAT:	Seite A-29
Bustyp: FlexRay:	Seite A-29
Bustyp: LIN:	Seite A-29
Bustyp: MIL STD 1553:	Seite A-29
Bustyp: Profibus:	Seite A-30
Bustyp: Profinet:	Seite A-30
Bustyp: RSxxx:	Seite A-30
Bustyp: SENT:	Seite A-30
Bustyp: SENT-Ausgang:	Seite A-30
Bustyp: SENT-Eingang:	Seite A-31
Bustyp: SPI:	Seite A-31
CPU-Digitalkanäle:	Seite A-32
Dehnungsmessstreifen:	Seite A-32
Digitale Ein-/Ausgänge:	Seite A-32
Multi-I/O:	Seite A-33
PWM-Ausgänge:	Seite A-34
SSI-Decoder:	Seite A-34
Status-Variablen:	Seite A-34
System:	Seite A-34
Temperatur-Eingänge:	Seite A-34
Zähler:	Seite A-35

Analoge Ausgänge

P2_DAC	gibt auf einem Kanal des angegebenen Moduls eine (analoge) Spannung aus, die dem angegebenen Digitalwert entspricht.
P2_DAC1_DIO	gibt auf dem DAC-Kanal 1 eine (analoge) Spannung aus und setzt oder löscht die digitalen Ausgänge des angegebenen Moduls.
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.

ge) Spannung aus.

P2_DAC_Ramp_Buffer_Free gibt an, ob der Zwischenpuffer für die Rampenausgabe frei ist.

P2_DAC_Ramp_Status gibt zurück, ob eine Rampenausgabe aktiv ist.

P2_DAC_Ramp_Stop stoppt eine Rampenausgabe sofort.

P2_DAC_Ramp_Write definiert die Parameter für die Ausgabe der nächsten Rampe und startet die DAC-Ausgabe.

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 Auflösung.

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_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_Gain setzt für einen Kanal des angegebenen Moduls die Betriebsart fest und damit auch den Ver-

	stärkungsfaktor 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 (Fast-ADC, Burst)

P2_Burst_CRead_Pos_Unpacked1	kopiert eine Anzahl Messwerte von einem Kanal ab einer Speicherposition im Modulspeicher in ein Feld.
P2_Burst_CRead_Pos_Unpacked2	kopiert eine Anzahl Messwerte von 2 Kanälen ab einer Speicherposition im Modulspeicher in 2 Felder.
P2_Burst_CRead_Pos_Unpacked4	kopiert eine Anzahl Messwerte von 4 Kanälen ab einer Speicherposition im Modulspeicher in 4 Felder.
P2_Burst_CRead_Pos_Unpacked8	kopiert eine Anzahl Messwerte von 8 Kanälen ab einer Speicherposition im Modulspeicher in 8 Felder.
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_Set_Average_Filter	legt fest, ob und aus wievielen Messwerten das angegebene Modul einen Mittelwert berechnet.

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 Messgruppe 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.

Bustyp: ARINC 429

ARINC_Create_Value32	erzeugt einen 32 Bit-Wert aus den Komponenten SSM, SDI, Daten und Label.
ARINC_Split_Value32	teilt einen 32 Bit-Wert in seine Komponenten Label, SSM, SDI, Daten und Parity-Bit auf.
P2_ARINC_Config_Receive	konfiguriert die Empfangs-Einstellungen auf der ARINC-Schnittstelle des Moduls.
P2_ARINC_Config_Transmit	konfiguriert die Sende-Einstellungen auf der ARINC-Schnittstelle des Moduls.
P2_ARINC_Read_Receive_Fifo	P2_ARINC_Receive_Fifo_Empty gibt zurück, ob der Empfangs-Fifo auf der ARINC-Schnittstelle des Moduls leer ist.
P2_ARINC_Receive_Fifo_Empty	gibt zurück, ob der Empfangs-Fifo auf der ARINC-Schnittstelle des Moduls leer ist.
P2_ARINC_Reset	startet einen Master-Reset auf der ARINC-Schnittstelle des Moduls.
P2_ARINC_Set_Labels	P2_ARINC_Receive_Fifo_Empty gibt zurück, ob der Empfangs-Fifo auf der ARINC-Schnittstelle des Moduls leer ist.
P2_ARINC_Transmit_Enable	sperrt oder aktiviert das Senden aus dem Sende-Fifo auf der ARINC-Schnittstelle des Moduls.
P2_ARINC_Transmit_Fifo_Empty	gibt zurück, ob der Sende-Fifo auf der ARINC-Schnittstelle des Moduls leer ist.
P2_ARINC_Transmit_Fifo_Full	gibt zurück, ob der Sende-Fifo auf der ARINC-Schnittstelle des Moduls voll ist.
P2_ARINC_Write_Transmit_Fifo	schreibt einen 32 Bit-Wert in den Sende-Fifo auf der ARINC-Schnittstelle des Moduls.

Bustyp: CAN

CAN_Msg	ist ein eindimensionales Feld mit 9 Elementen, in dem 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 Eintreffen 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.

	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.
P2_Transmit_Status	gibt zurück, ob ein Message-Objekt bereit ist zum Senden.

Bustyp: CAN-FD

P2_CANFD_Enable_Receive_Fifo	definiert einen Fifo für eingehende CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.
P2_CANFD_Enable_Transmit_Event_Fifo	definiert einen Überwachungs-Fifo für gesendete CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.
P2_CANFD_Enable_Transmit_Fifo	definiert einen Fifo für ausgehende CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.
P2_CANFD_Enable_Transmit_Queue	definiert einen Ausgabepuffer für ausgehende CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.
P2_CANFD_Get_BDIAG0	gibt den Inhalt des Registers BDIAG0 auf einem CAN-Controller zurück.
P2_CANFD_Get_BDIAG1	gibt den Inhalt des Registers BDIAG0 auf einem CAN-Controller zurück.
P2_CANFD_Get_BRS	gibt aus einem CAN-Nachrichtenobjekt den Parameter BRS aus dem Nachrichten-Header zurück.
P2_CANFD_Get_DLC	gibt aus einem CAN-Nachrichtenobjekt den Parameter DLC aus dem Nachrichten-Header zurück.
P2_CANFD_Get_ESI	gibt aus einem CAN-Nachrichtenobjekt den Parameter ESI aus dem Nachrichten-Header zurück..
P2_CANFD_Get_FDF	gibt aus einem CAN-Nachrichtenobjekt den Parameter FDF aus dem Nachrichten-Header zurück..
P2_CANFD_Get_Fifo_State	P2_CANFD_Enable_Transmit_Queue definiert einen Ausgabepuffer für ausgehende CANFD-Nachrichten und speichert die Informationen in einem CANFD-Datenfeld.
P2_CANFD_Get_Header_Parts	gibt aus einem CAN-Nachrichtenobjekt die Bestandteile des Nachrichten-Headers zurück.
P2_CANFD_Get_ID	gibt den Parameter ID aus Header eines CAN-Nachrichtenobjekts zurück.
P2_CANFD_Get_IDE	gibt aus einem CAN-Nachrichtenobjekt den Parameter IDE aus dem Nachrichten-Header zurück.
P2_CANFD_Get_RTR	gibt aus einem CAN-Nachrichtenobjekt den Parameter RTR aus dem Nachrichten-Header zurück.
P2_CANFD_Get_SEQ	gibt aus einem Nachrichtenobjekt (EFO) den Parameter SEQ zurück.
P2_CANFD_Get_TREC	gibt den Inhalt des Registers TREC auf einem CAN-Controller zurück.
P2_CANFD_Init_Controller	initialisiert einen CAN-Controller auf dem angegebenen Modul und setzt die vorbereiteten Einstellungen.
P2_CANFD_Init_Datatable	initialisiert ein Datenfeld für Informationen des CAN FD-Controllers.
P2_CANFD_Read_EFO	prüft, ob im Überwachungs-Fifo ein Nachrichtenobjekt vorhanden ist.
P2_CANFD_Read_RMO	prüft, ob in einem Fifo eine Nachricht empfangen wurde.
P2_CANFD_Set_Baudrate_Data	stellt die Daten-Baudrate ein und speichert die Informationen in einem CANFD-Datenfeld.
P2_CANFD_Set_Baudrate_Nominal	stellt die nominale Baudrate ein und speichert die Informationen in einem CANFD-Datenfeld.
P2_CANFD_Set_LED	schaltet die Zusatz-LED für einen CAN FD-Kanal auf dem Modul ein oder aus.
P2_CANFD_Set_Mode	stellt den CAN-Betriebsmodus eines Controllers ein.
P2_CANFD_Set_Reg	schreibt einen Wert in ein Register des gewählten CAN-Controllers auf dem angegebenen Modul.
P2_CANFD_Set_SID11	stellt ein, ob CAN FD-Nachrichten mit einem Identifier von 11 Bit oder 12 Bit Länge gesendet werden und speichert die Information in einem CANFD-Datenfeld.
P2_CANFD_Set_TDC	stellt einen Ausgleich für Zeitverzögerungen beim Senden ein und speichert die Information

nen in einem CANFD-Datenfeld.

[P2_CANFD_Transmit_Msg](#) gibt das Senden von gespeicherten CAN-Nachrichten aus einem Fifo frei.

[P2_CANFD_Transmit_Multi_Msg](#) gibt das Senden von gespeicherten CAN-Nachrichten aus mehreren Fifos frei.

[P2_CANFD_Write_TMO](#) übergibt eine CAN-Nachricht an den Controller.

Bustyp: EtherCAT

[P2_ECAT_Get_State](#) gibt den Status der EtherCAT-Schnittstelle zurück.

[P2_ECAT_Get_Version](#) gibt die Version der EtherCAT-Schnittstelle zurück.

[P2_ECAT_Init](#) initialisiert den EtherCAT-Slave.

[P2_ECAT_Read_Data_16F](#) liest 16 Float-Werte vom EtherCAT-Slave und gibt sie in einem Feld zurück.

[P2_ECAT_Read_Data_16L](#) liest 16 Long-Werte vom EtherCAT-Slave und gibt sie in einem Feld zurück.

[P2_ECAT_Set_Mode](#) stellt den Datenübertragungs-Modus des EtherCAT-Slave ein.

[P2_ECAT_Write_Data_16F](#) schreibt 16 Float-Werte aus einem Feld zum EtherCAT-Slave.

[P2_ECAT_Write_Data_16L](#) schreibt 16 Long-Werte aus einem Feld zum EtherCAT-Slave.

[P2_Init_EtherCAT](#) initialisiert ein Feld für den Betrieb mit dem EtherCAT-Slave.

[P2_Run_EtherCAT](#) tauscht Daten mit dem EtherCAT-Slave aus.

Bustyp: 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.

Bustyp: LIN

[P2_LIN_Ch_Read_Cnt](#) gibt die Anzahl der übertragenen Nachrichten einer LIN-Schnittstelle zurück.

[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_Read_Status](#) gibt den Status einer Messagebox einer LIN-Schnittstelle zurück.

[P2_LIN_Msg_Transmit](#) sendet im Betriebsmodus LIN Master einen Header und den Identifier einer Messagebox auf den LIN-Bus.

[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.

Bustyp: MIL STD 1553

[P2_MIL_Get_Register](#) gibt den Wert eines Registers von der MIL-Schnittstelle auf dem angegebenen Modul zurück.

[P2_MIL_Reset](#) initialisiert die MIL-Schnittstelle auf einem bestimmten Modul und setzt alle Register auf die Vorgabewerte zurück.

[P2_MIL_Set_LED](#) schaltet die Zusatz-LEDs der MIL-Schnittstelle auf dem Modul ein oder aus.

[P2_MIL_Set_Register](#) setzt den Wert eines Registers auf der MIL-Schnittstelle auf dem angegebenen Modul.

[P2_MIL_SMT_Init](#) initialisiert den Modus SMT-Monitor 16 Bit (simple monitoring terminal) für beide Busse A und

B auf dem angegebenen Modul.

[P2_MIL_SMT_Message_Read](#) liest die Zwischenspeicher für Befehle und Daten der zuletzt gespeicherten MIL-Nachricht auf dem angegebenen Modul.

[P2_MIL_SMT_Set_All_Filters](#) sperrt oder aktiviert die Filter für alle Sende- und Empfangs-Unteradressen aller Terminals auf dem angegebenen Modul.

[P2_MIL_SMT_Set_Filter](#) sperrt oder aktiviert die Filter für alle Sende- und Empfangs-Unteradressen eines Terminals auf dem angegebenen Modul.

Bustyp: Profibus

[P2_Init_Profibus](#) initialisiert den Profibus-Slave.

[P2_Init_Profibus_M40](#) initialisiert den Profibus-Slave.

[P2_Run_Profibus](#) tauscht Daten mit dem Profibus-Slave aus.

[P2_Run_Profibus_M40](#) tauscht Daten mit dem Profibus-Slave aus.

Bustyp: Profinet

[P2_Init_ProfinetIO](#) initialisiert ein Feld für den Betrieb mit dem Profinet-Slave.

[P2_Run_ProfinetIO](#) tauscht Daten mit dem Profinet-Slave aus.

Bustyp: RSxxx

[P2_Check_Shift_Reg](#) gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO der Schnittstelle (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 einer bestimmten Schnittstelle auf dem angegebenen Modul.

[P2_RS485_Send](#) legt die Übertragungsrichtung für eine bestimmte Schnittstelle des angegebenen Moduls fest.

[P2_RS_Init](#) initialisiert eine bestimmte Schnittstelle 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 einer bestimmten Schnittstelle auf dem angegebenen Modul.

[P2_Write_Fifo_Full](#) gibt zurück, ob noch mindestens ein Speicherplatz im Sende-FIFO einer bestimmten Schnittstelle auf dem angegebenen Modul frei ist.

Bustyp: SENT

[P2_SENT_Command_Ready](#) gibt zurück, ob die SENT-Schnittstelle auf dem angegebenen Modul bereit ist zum Verarbeiten eines Befehls.

[P2_SENT_Get_Msg_Counter](#) gibt die Anzahl der empfangenen/gesendeten Nachrichten auf dem angegebenen SENT-Kanal zurück.

[P2_SENT_Get_Version](#) gibt die Version der SENT-Schnittstelle auf dem angegebenen Modul zurück.

[P2_SENT_Init](#) initialisiert die Datenübertragung zwischen ADwin CPU und der SENT-Schnittstelle auf einem bestimmten Modul.

Bustyp: SENT-Ausgang

[P2_SENT_Config_Output](#) setzt die Grundeinstellungen für einen SENT-Ausgabekanal auf dem angegebenen Modul.

[P2_SENT_Config_Serial_Messages](#) konfiguriert das Sendeformat für serielle Nachrichten auf einem SENT-Kanal.

[P2_SENT_Enable_Channel](#) sperrt mehrere SENT-Kanäle oder gibt sie frei.

[P2_SENT_Fifo_Clear](#) initialisiert den Schreib- und Lese-Zeiger des Ausgabe-FIFOs eines SENT-Kanals.

[P2_SENT_Fifo_Empty](#) ermittelt die Anzahl der freien Elemente im Ausgabe-FIFO eines SENT-Kanals.

[P2_SENT_Invert_Channel](#) invertiert alle Signalpegel auf einem SENT-Kanal des angegebenen Moduls.

[P2_SENT_Set_Fast_Channel1](#) setzt den ersten 12 Bit-Wert in der SENT-Nachricht für einen SENT-Kanal auf dem angegebenen Modul.

[P2_SENT_Set_Fast_Channel2](#) setzt den zweiten 12 Bit-Wert in der SENT-Nachricht für einen SENT-Kanal auf dem

angegebenen Modul.

P2_SENT_Set_Fifo schreibt neue Daten in den Ausgabe-Fifo eines SENT-Kanals auf dem angegebenen Modul.

P2_SENT_Set_Output_Mode stellt den Ausgabemodus für einen SENT-Kanal auf dem angegebenen Modul ein.

P2_SENT_Set_Reserved_Bits setzt die reservierten Bits im Status-Nibble eines SENT-Kanals auf dem angegebenen Modul.

P2_SENT_Set_Serial_Message_Data ändert einen Datenwert im Nachrichtensatz für serielle Nachrichten.

P2_SENT_Set_Serial_Message_Pattern definiert einen Nachrichtensatz für serielle Nachrichten auf einem SENT-Kanal.

Bustyp: SENT-Eingang

P2_SENT_Check_Latch gibt zurück, ob der Latch-Zwischenspeicher Daten der angeforderten SENT-Kanäle enthält.

P2_SENT_Clear_Serial_Message_Array setzt den zwischengespeicherten Nachrichtensatz von seriellen Nachrichten eines SENT-Kanals zurück.

P2_SENT_Get_ChannelState gibt den Empfangsmodus der SENT-Kanäle auf dem angegebenen Modul zurück.

P2_SENT_Get_ClockTick gibt den Basistakt eines SENT-Eingangskanals auf dem angegebenen Modul zurück.

P2_SENT_Get_Fast_Channel1 liest den ersten 12 Bit-Wert vom angegebenen SENT-Kanal auf dem angegebenen Modul.

P2_SENT_Get_Fast_Channel2 liest den zweiten 12 Bit-Wert vom angegebenen SENT-Kanal auf dem angegebenen Modul.

P2_SENT_Get_Fast_Channel_CRC_OK gibt das Ergebnis der CRC-Prüfung für die Signale einer SENT-Nachricht auf einem Kanal auf dem angegebenen Modul zurück.

P2_SENT_Get_Latch_Data liest die Daten einer SENT-Nachricht für einen SENT-Kanal aus dem Latch-Zwischenspeicher.

P2_SENT_Get_PulseCount gibt zurück, wie viele Pulse eine SENT-Nachricht an einem Eingangskanal auf dem angegebenen Modul enthält.

P2_SENT_Get_Serial_Message_Array gibt einen vollständigen Nachrichtensatz von seriellen Nachrichten auf einem SENT-Kanal zurück.

P2_SENT_Get_Serial_Message_CRC_OK gibt das Ergebnis der CRC-Prüfung einer seriellen Nachricht auf dem angegebenen Modul zurück.

P2_SENT_Get_Serial_Message_Data gibt den Datenwert einer seriellen Nachricht auf dem angegebenen Modul zurück.

P2_SENT_Get_Serial_Message_Id gibt die Kennung einer seriellen Nachricht auf dem angegebenen Modul zurück.

P2_SENT_Request_Latch fordert an, bestimmte SENT-Kanäle auf dem angegebenen Modul einmalig über den Latch-Zwischenspeicher zu puffern.

P2_SENT_Set_ClockTick schaltet einen SENT-Kanal auf dem angegebenen Modul in den Lesemodus mit einem definierten Basistakt.

P2_SENT_Set_CRC_Implementation setzt den Berechnungsalgorithmus für die CRC-Prüfsumme für einen SENT-Kanal auf dem angegebenen Modul.

P2_SENT_Set_Detection setzt einen SENT-Kanal auf dem angegebenen Modul in den Erkennungsmodus.

P2_SENT_Set_PulseCount stellt ein, ob ein Pausenpuls in SENT-Nachrichten eines Eingangskanals auf dem angegebenen Modul erwartet wird.

P2_SENT_Set_Sensor_Type stellt den erwarteten Sensortyp für die SENT-Nachrichten auf einem Eingangskanal auf dem angegebenen Modul ein.

Bustyp: SPI

P2_SPI_Config legt Eigenschaften einer SPI-Schnittstelle des angegebenen Moduls fest, für Master wie für Slave.

P2_SPI_Master_Config legt (zusätzliche) Eigenschaften einer SPI-Master-Schnittstelle des Moduls fest.

P2_SPI_Master_Get_Static_Input liest den Pegel auf der Datenleitung des SPI-Bus.

P2_SPI_Master_Get_Value32 liest eine (bereits empfangene) SPI-Nachricht mit bis zu 32 Bit aus dem Eingangsregister der SPI-Schnittstelle.

P2_SPI_Master_Get_Value64 liest eine (bereits empfangene) SPI-Nachricht mit bis zu 64 Bit aus dem Eingangsregister der SPI-Schnittstelle.

P2_SPI_Master_Set_Clk_Wait fügt mehrere Wartezeiten nach einer wählbaren Anzahl von Takten in das Taktsignal

eines SPI-Masters ein.

P2_SPI_Master_Set_Value32 stellt eine SPI-Nachricht bis 32 Bit Länge am Master-Ausgang zur Ausgabe bereit.

P2_SPI_Master_Set_Value64 stellt eine SPI-Nachricht bis 64 Bit Länge am Master-Ausgang zur Ausgabe bereit.

P2_SPI_Master_Start startet die Datenübertragung über den SPI-Bus. Falls konfiguriert, wird die Slave-Select-Leitung des SPI-Masters automatisch aktiviert.

P2_SPI_Master_Status gibt zurück, ob die Datenübertragung des SPI-Masters aktiv oder inaktiv ist.

P2_SPI_Mode legt den Betriebsmodus (SPI-Master / SPI-Slave / Digitalmodul) des angegebenen Moduls fest.

P2_SPI_Slave_Clear_Fifo löscht den Eingangs- und/oder den Ausgangs-Fifo eines SPI-Slaves.

P2_SPI_Slave_Config legt (zusätzliche) Eigenschaften einer SPI-Slave-Schnittstelle des Moduls fest.

P2_SPI_Slave_InFifo_Full gibt die Anzahl der belegten Plätze (=eingegangene 32 Bit-Werte) im Eingangs-Fifo zurück.

P2_SPI_Slave_InFifo_Read liest mehrere 32 Bit-Werte als SPI-Nachrichten aus dem Eingangs-Fifo eines SPI-Slaves.

P2_SPI_Slave_OutFifo_Empty gibt die Anzahl der freien Plätze im Ausgangs-Fifo zurück.

P2_SPI_Slave_OutFifo_Write schreibt mehrere 32 Bit-Werte als SPI-Nachrichten in den Ausgangs-Fifo eines SPI-Slaves.

CPU-Digitalkanäle

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.

Dehnungsmessstreifen

P2_SG_Convert berechnet aus dem Digitalwert eines DMS die zugehörige Spannung.

P2_SG_Init stellt Verstärkung und Filterfrequenz für die Brückenspannung sowie die Versorgungsspannung eines DMS ein.

P2_SG_Mode stellt den Betriebsmodus für Dehnungsmessstreifen auf dem angegebenen Modul ein und wählt die zu messenden Kanäle aus.

P2_SG_Read kopiert eine bestimmte Anzahl an Messwerten (24 Bit) von dem angegebenen Modul in ein Ziel-Feld.

P2_SG_Set_Gain führt den Abgleich des Verstärkungsfaktors für einen DMS-Kanal auf dem angegebenen Modul durch.

P2_SG_Start startet die Ablaufsteuerung auf allen angegebenen Modulen gleichzeitig.

P2_SG_Wait wartet, bis die Ablaufsteuerung auf dem angegebenen Modul alle Kanäle der Messgruppe gewandelt und gespeichert hat.

P2_SG_Zero führt den Nullpunktabgleich für einen DMS-Kanal auf dem angegebenen Modul durch.

Digitale Ein-/Ausgänge

P2_Comp_Filter_Init stellt die Filter-Prüfdauer für alle Komparatoren auf dem angegebenen Modul ein.

P2_Comp_Init stellt den Betriebsmodus für die Komparatoren einer Kanalgruppe auf dem angegebenen Modul ein.

P2_Comp_Set stellt die Schaltschwelle für die Komparatoren einer Kanalgruppe auf dem angegebenen Modul ein.

P2_Comp_Set_Voltage stellt die Schaltschwelle für die Komparatoren einer Kanalgruppe auf dem angegebenen Modul ein, angegeben in Volt.

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 2 Felder.

P2_Digin_Fifo_Read_Fast P2_Digin_Fifo_Read liest die Wertepaare aus dem FIFO der Flankenüberwachung und

schreibt sie in 2 Felder.

P2_Digin_Fifo_Read_Timer	gibt den aktuellen Stand des Zählers auf dem angegebenen Modul zurück.
P2_Digin_Filter_Init	stellt die Filter-Prüfdauer für alle Eingänge auf dem angegebenen Modul ein.
P2_Digin_Long	gibt den Zustand der Eingänge (Bits 31 0) 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 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_DigProg_Bits	programmiert die digitalen Kanäle des angegebenen Moduls einzeln als Ein- oder Ausgang.
P2_DigProg_Set_IO_Level	setzt die Spannungspegel in Gruppen zu je 8 auf einen definierten Wert.
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.

Multi-I/O

P2_MIO_Digin_Long	gibt den Zustand der Eingänge 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.

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	schreibt Frequenz und Tastverhältnis eines oder mehrerer PWM-Ausgänge in das Register 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.

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 (6,1 kHz bis 12,5 MHz) 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.

Status-Variablen

Calc_Processdelay	gibt die Anzahl der Zähler-Taktzyklen (Processdelay oder Zykluszeit) zu einer Prozessfrequenz zurück.
Calc_TicksToNs	berechnet aus eine Anzahl Zähler-Taktzyklen die zugehörige Zeit in Nanosekunden.

System

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