

ADwin-Pro, -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
Jäger Computergesteuerte
Messtechnik GmbH
Rheinstraße 2-4
D-64653 Lorsch

Inhaltsverzeichnis

Typografische Konventionen	IV
1 Einführung	1
2 Das Programm <code>ADpro.exe</code>	2
3 ADbasic -Anweisungen für ADwin-Pro I -Module	3
3.1 Pro I: Alle Module	3
3.2 Pro I: Eingangsmodule	17
3.3 Pro I: Ausgangsmodule	70
3.4 Pro I: Digitalmodule	86
3.5 Pro I: Spezielle Module	165
4 ADbasic -Anweisungen für ADwin-Pro II -Module	226
4.1 Pro II: Alle Module	226
4.2 Pro II: Eingangsmodule	243
4.3 Pro II: Ausgangsmodule	317
5 Programmbeispiele	331
5.1 Online-Auswertung von Messwerten	331
5.2 Digitaler Proportional-Regler	332
5.3 Datenaustausch mit DATA-Feldern	332
5.4 Digitaler PID-Regler	333
5.5 Datenaustausch mit dem Feldbus	335
5.6 Beispiele für RS232 und RS485	336
5.7 Kontinuierliche Messwertwandlung	342
Anhang	A-1
A.1 Alphabetische Befehlsübersicht	A-1
A.2 Befehlsübersicht nach Modulen	A-5
A.3 Thematische Befehlsübersicht	A-17
A.4 Abkürzungsverzeichnis	A-27

Typografische Konventionen



<C:\ADwin\ ...>

Programmtext

Var_1

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.

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

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

Elemente eines Quelltextes wie **BEFEHLE**, Variablen, Kommentar und sonstiger Text werden im Schrifttyp Courier New und farbig dargestellt (wie im Editor der Entwicklungsumgebung *ADbasic*).

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

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

1 Einführung

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

Dieses Handbuch beschreibt die *ADbasic*-Befehle zum Ansprechen der verschiedenen Module ([Befehlsübersicht nach Modulen](#) im Anhang).

Darüber hinaus beschreibt das *ADbasic*-Handbuch grundlegende Befehle z.B. für Berechnungen, den Aufbau der Programmstruktur oder das Steuern von Prozessen.

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

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

```
#INCLUDE ADwinPRO_ALL.inc
```

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



Bitte beachten Sie folgende Hinweise

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

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 ICE 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.

Änderungen vorbehalten.

Hotline-Adresse siehe vordere Umschlagseite, innen.

Einschränkung der Anwendergruppe

Verfügbarkeit der Unterlagen



Rechtliche Grundlagen

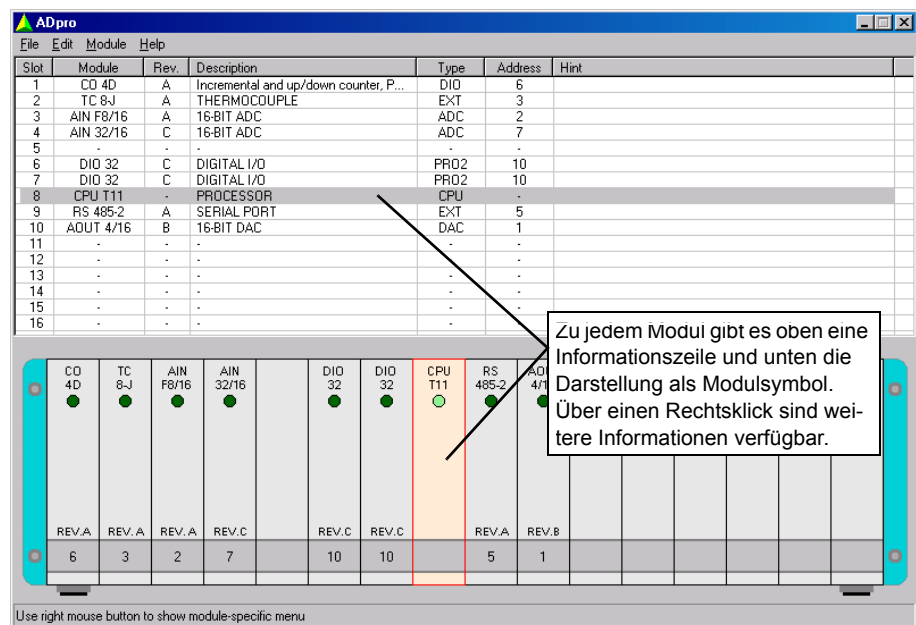
2 Das Programm ADpro.exe

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 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 <.NET Framework 1.1> installiert ist.

3 ADbasic-Anweisungen für ADwin-Pro I-Module

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

Im Anhang finden Sie außerdem sortierte Befehls-Übersichten:

- [Alphabetische Befehlsübersicht](#)
- [Befehlsübersicht nach Modulen](#)

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

- [Thematische Befehlsübersicht](#)

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

```
#INCLUDE ADwinPRO_ALL.INC
```

Zu jeder Befehlsbeschreibung gehören

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

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

3.1 Pro I: Alle Module

Die Include-Datei `<ADwinpro.inc>` beinhaltet allgemeine System-Funktionen und Prozeduren, die zum Ansprechen der *ADwin-Pro I-Module* benötigt werden. Wenn Sie diese Datei mit der *ADbasic*-Anweisung

```
#INCLUDE ADwinPRO_ALL.inc
```

in Ihr *ADbasic*-Programm einbinden, können Sie sämtliche Funktionen und Prozeduren daraus verwenden.

Die Befehle für Pro I-Module benötigen u. a. die Angabe des Parameters `mod_class`, der die Modulkategorie eines Moduls angibt. Sie können für diesen Parameter die folgenden Konstanten verwenden, die in der Datei `<ADwinpro.inc>` definiert sind:

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



CHECKLED

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = CHECKLED(module, mod_class)
```

Parameter

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

Bemerkungen

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

Siehe auch

[SETLED](#)

Gültig für Module

(LP)SH-4/8(-FI), Aln-16/14-C Rev. A, Aln-32/12 Rev. A, Aln-32/12 Rev. B, Aln-32/14 Rev. A, Aln-32/16 Rev. B, Aln-32/16 Rev. C, Aln-8/12 Rev. A, Aln-8/12 Rev. B, Aln-8/14 Rev. A, Aln-8/16 Rev. A, Aln-8/16 Rev. B, Aln-8/16 Rev. C, Aln-F-4/12 Rev. A, Aln-F-4/14 Rev. B, Aln-F-4/16 Rev. A, Aln-F-8/12 Rev. A, Aln-F-8/14 Rev. B, Aln-F-8/16 Rev. A, AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, AOut-8/16 Rev. C, CAN-1, CAN-2, CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR4(-I), CNT-VR4L(-I), CO4, COMP16 Rev. A, CPU-T10, CPU-T9, DIO-32, DIO-32 Rev. B, Inter-SL, LS2 Rev. A, OPT-16 Rev. A, Profi-SL, PT100-4, PT100-8, PWM-4(-I), REL-16, RS232-2, RS232-4, RS485-2, RS485-4, Storage Rev. A, TC-16, TC-4, TC-8, TC-8-ISO, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
  IF (CHECKLED(1,dio)=0) THEN 'Falls LED aus ist ...
    SETLED(1,dio,1)           '... dann LED einschalten
  ENDIF
```


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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = CPU_DIGIN()
```

Parameter

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

Bemerkungen

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

Am Eingang Digin 0 werden TTL-Signale erwartet.

Siehe auch

[CPU_DIGIN](#) (T11)

Gültig für Module

CPU-T10, CPU-T9

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM dummy AS LONG  
  
INIT:  
    REM Statusmeldung auslesen und dadurch zurücksetzen  
    dummy = CPU_DIGIN()  
  
EVENT:  
    ...  
    IF (CPU_DIGIN() = 1) THEN 'Falls fallende Flanke aufgetreten ...  
        END                '... dann das Programm beenden  
    ENDIF  
    ...
```

CPU_DIGIN

EVENTENABLE

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

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
EVENTENABLE (module, mod_class, value)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>mod_class</code>	Modulkategorie: Eine der Konstanten <code>ad</code> , <code>da</code> , <code>dio</code> , <code>cpu</code> oder <code>ext</code> .	LONG
<code>value</code>	Signalstatus: 0 : externes Event-Signal sperren (Default). 1 : externes Event-Signal zulassen.	LONG

Bemerkungen

Sie können einen hochprioritären *ADbasic*-Prozess (d.h. dessen zyklischen Abschnitt Event:) durch ein externes Event-Signal aufrufen lassen und damit z.B. mit einem externen Prozess synchronisieren (vgl. *ADbasic*-Handbuch).

Die meisten Module verfügen über einen Event-Eingang. Sobald Sie mit `EventEnable` einen Eingang aktiviert haben, wird das anliegende Signal an das Prozessormodul weitergeleitet. Das Prozessormodul erkennt eine steigende Flanke als Event-Signal und der eingestellte Prozess reagiert.

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

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

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
INIT:
    REM Externes Event am Digital-Modul 1 zulassen
    EVENTENABLE (1, dio, 1)

EVENT:
    ...
    '16)
```



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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

RESETWATCHDOGTIMER()
```

Parameter

Keine

Bemerkungen

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

Prozessortyp	Dauer
T9, T10, T11	1600ms

Dauer des Zählvorgangs vom Startwert bis auf 0

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

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



Siehe auch

[STARTWATCHDOG](#), [STOPWATCHDOG](#)

Gültig für Module

CPU-T10, CPU-T11, CPU-T9

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    STARTWATCHDOG()           'Watchdog aktivieren
EVENT:
    RESETWATCHDOGTIMER()      'Watchdog regelmäßig zurücksetzen
...
FINISH:
    STOPWATCHDOG()           'Watchdog deaktivieren
```

SETLED

CAN-1, CAN-2**SETLED** schaltet die LED (oben auf der Frontplatte) auf dem angegebenen Modul ein oder aus.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
SETLED (module, mod_class, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
mod_class	Modulkategorie: Eine der Konstanten <code>ad</code> , <code>da</code> , <code>dio</code> , <code>cpu</code> oder <code>ext</code> .	LONG
pattern	Gewünschter Schaltzustand der LED. 0: ausschalten. 1: einschalten.	LONG

Bemerkungen

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

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

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

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

Siehe auch

CHECKLED

Gültig für Module

(LP)SH-4/8(-FI), Aln-16/14-C Rev. A, Aln-32/12 Rev. A, Aln-32/12 Rev. B, Aln-32/14 Rev. A, Aln-32/16 Rev. B, Aln-32/16 Rev. C, Aln-8/12 Rev. A, Aln-8/12 Rev. B, Aln-8/14 Rev. A, Aln-8/16 Rev. A, Aln-8/16 Rev. B, Aln-8/16 Rev. C, Aln-F-4/12 Rev. A, Aln-F-4/14 Rev. B, Aln-F-4/16 Rev. A, Aln-F-8/12 Rev. A, Aln-F-8/14 Rev. B, Aln-F-8/16 Rev. A, AOut-16/8-12, AOut-4/16 Rev. A, AOut-4/16 Rev. B, AOut-4/16 Rev. C, AOut-8/16 Rev. A, AOut-8/16 Rev. B, AOut-8/16 Rev. C, CAN-1, CAN-2, CNT-16/16(-I), CNT-16/32(-I), CNT-8/32(-I), CNT-PW4(-I), CNT-VR4(-I), CNT-VR4L(-I), CO4, COMP16 Rev. A, CPU-T10, CPU-T9, DIO-32, DIO-32 Rev. B, Inter-SL, LS2 Rev. A, OPT-16 Rev. A, Profi-SL, PT100-4, PT100-8, PWM-4(-I), REL-16, RS232-2, RS232-4, RS485-2, RS485-4, Storage Rev. A, TC-16, TC-4, TC-8, TC-8-ISO, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
INIT:
    SETLED(1,cpu,1)          'LED am Prozessor-Modul 1 einschalten
    SETLED(1,ad,1)           'LED am A/D-Modul 1 einschalten
    SETLED(1,da,1)           'LED am D/A-Modul 1 einschalten
    SETLED(1,dio,1)          'LED am DIO-Modul 1 einschalten
    SETLED(2,dio,0110b)      'Am Modul Pro-Storage (DIO Nr. 2)
                             'obere
                             'LED rot und untere LED grün schalten

EVENT:
    ...

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

STARTWATCH- DOG

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
STARTWATCHDOG ()
```

Bemerkungen

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

Prozessortyp	Dauer
T9, T10, T11	1600ms

Dauer des Zählvorgangs vom Startwert bis auf 0

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

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



Siehe auch

[RESETWATCHDOGTIMER](#), [STOPWATCHDOG](#)

Gültig für Module

CPU-T10, CPU-T11, CPU-T9

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
INIT:
    STARTWATCHDOG ()           'Watchdog aktivieren
```

STOPWATCHDOG deaktiviert den Watchdog-Zähler des CPU-Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
STOPWATCHDOG ()
```

Bemerkungen

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

Siehe auch

[RESETWATCHDOGTIMER](#), [STARTWATCHDOG](#)

Gültig für Module

CPU-T10, CPU-T11, CPU-T9

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
INIT:  
    STARTWATCHDOG ()          'Watchdog aktivieren  
    ...  
  
EVENT:  
    RESETWATCHDOGTIMER ()     'Watchdog zurücksetzen  
    ...  
  
FINISH:  
    STOPWATCHDOG ()           'Watchdog deaktivieren
```

STOPWATCHDOG



SYNCALL

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
SYNCALL ( )
```

Bemerkungen

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

Modultyp	Aktion
Analog-Eingang	A/D-Wandlung auf allen ADC starten (SYNCENABLE (..., ..., 1) oder Burst-Messreihe starten (SYNCENABLE (..., ..., 3) und nur für Module Pro-Aln-F-x/14).
Analog-Ausgang	D/A-Wandlung starten mit dem Wert aus dem DAC-Register (siehe WRITEDAC).
Digital-Eingang	Aktuellen Zustand der Eingänge in das Eingangs- Zwischenregister übertragen (von dort auslesen mit DIG_READLATCH1 , DIG_READLATCH2).
Digital-Ausgang	Wert aus dem Ausgangs-Zwischenregister lesen und auf die digitalen Ausgänge schalten (siehe DIG_WRITELATCH1 , DIG_WRITELATCH2).
Zähler	Aktuelle Zählerstände in die Zähler-Zwischenre- gister übertragen (von dort auslesen mit CNT_READLATCH16 , CNT_ READLATCH32 , CO4_READLATCH).

Siehe auch

SYNCENABLE, **SYNCSTAT**

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM i AS LONG
DIM DATA_1[1000], DATA_2[1000], DATA_3[1000] AS LONG
INIT:
    REM Multiplexer der A/D Module 1-3 auf Eingang 1 setzen
    SET_MUX(1,0)
    SET_MUX(2,0)
    SET_MUX(3,0)
    REM Synchronisation der A/D Module 1-3 aktivieren
    SYNCENABLE(1,ad,1)
    SYNCENABLE(2,ad,1)
    SYNCENABLE(3,ad,1)
    i=1                                'Index initialisieren

EVENT:
    REM Wandlung für alle 3 Module synchron starten
    SYNCALL()                        'Wandlung aller aktiven Module
                                    'starten
    WAIT_EOC(1)                      'Auf des Ende der Wandlung warten
    DATA_1[i]=READADC(1)            'A/D Wandler Modul 1 auslesen
    DATA_2[i]=READADC(2)            'A/D Wandler Modul 2 auslesen
    DATA_3[i]=READADC(3)            'A/D Wandler Modul 3 auslesen
    IF (i=1000) THEN END             'Nach 1000 Durchläufen Prozess
                                    'beenden
    INC(i)                          'Index erhöhen
```

SYNCENABLE

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
SYNCENABLE (module, mod_class, enable)
```

Parameter

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

Bemerkungen

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

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

Siehe auch

[SYNCALL](#), [SYNCSTAT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM i AS LONG
DIM DATA_1[1000], DATA_2[1000], DATA_3[1000] AS LONG
DIM DATA_4[1000] AS LONG

INIT:
  REM Multiplexer der A/D Module 1-3 auf Eingang 1 setzen
  SET_MUX(1,0)          'Multiplexer auf Eingang 1 setzen
  SET_MUX(2,0)
  SET_MUX(3,0)
  REM Synchronisation aktivieren: A/D Modul 1, DIO Module 1+2
  SYNCENABLE(1,ad,1)
  SYNCENABLE(1,dio,1)
  SYNCENABLE(2,ad,1)
  i=1                    'Index initialisieren

EVENT:
  REM - Wandlung A/D Modul 1 starten
  REM - Aktuellen Zustand der dig. Eingänge des Digital-
  REM   I/O-Moduls 1 in das Eingangs-Zwischenregister
  REM   übernehmen bzw. Wert aus dem Ausgangs-Zwischen-
  REM   register auf die dig. Ausgänge geben
  REM - Aktuelle Zählerstände der Zähler von Modul 2 in
  REM   die Zähler-Zwischenregister übernehmen
  SYNCALL()              'Wandlung A/D Modul starten
  WAIT_EOC(1)             'Auf des Ende der Wandlung warten
  DATA_1[i]=READADC(1)    'A/D Wandler Modul 1 auslesen
  REM Zwischenregister der DIO-Module auslesen
  DATA_2[i]=DIG_READLATCH1(1)
  DATA_3[i]=CNT_READ32(2,1) 'Zwischenregister Zähler 1
  DATA_4[i]=CNT_READ32(2,2) 'Zwischenregister Zähler 2
  IF (i=1000) THEN END    'Nach 1000 Durchläufen
                           ' den Prozess beenden
  INC(i)                  'Index erhöhen
```

SYNCSTAT

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = SYNCSTAT(module,mod_class)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
mod_class	Modulklass: Eine der Konstanten <code>ad</code> , <code>da</code> , <code>dio</code> , <code>cpu</code> oder <code>ext</code> .	LONG
ret_val	Einstellung der Synchron-Option: 0 : deaktiviert. 1 : aktiviert (für "normale" Aktion; siehe SYNCALL). 3 : aktiviert für Burst-Messreihe (nur für Module Pro-Aln-F-x/14).	LONG

Siehe auch

[SYNCALL](#), [SYNCENABLE](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM i AS LONG
DIM DATA_1[1000], DATA_2[1000] AS LONG

INIT:
  IF (SYNCSTAT(1,da)=0) THEN 'Ist Synchron-Option deaktiviert?
    SYNCENABLE(1,da,1)      'Dann Synchronisation aktivieren für
    SYNCENABLE(2,da,1)      'D/A Module 1+2
  ENDIF
  i=1                        'Index initialisieren

EVENT:
  WRITEDAC(1,1,DATA_1[i]) 'Ausgabe vorbereiten
  WRITEDAC(2,1,DATA_2[i])
  'Ausgabe auf den beiden Modulen synchron starten
  SYNCALL()
  IF (i=1000) THEN END      'Nach 1000 Durchläufen Prozess
                             'beenden
  INC(i)                    'Index erhöhen
```

3.2 Pro I: Eingangsmodule

Die Include-Datei `ADWPAD.INC` beinhaltet alle Funktionen und Prozeduren, die zum Ansprechen der *ADwin-Pro I* A/D-Module benötigt werden. Wenn Sie diese Datei mit der *ADbasic*-Anweisung

```
#INCLUDE ADwinPRO_ALL.inc
```

in Ihr *ADbasic*-Programm einbinden, können Sie sämtliche Funktionen und Prozeduren daraus verwenden.

Auf den folgenden Seiten werden alle Funktionen aus der Datei `ADWPAD.INC` ausführlich erklärt. Zusätzlich ist zu jeder Funktion ein Anwendungsbeispiel aufgeführt.

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

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



ADC

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

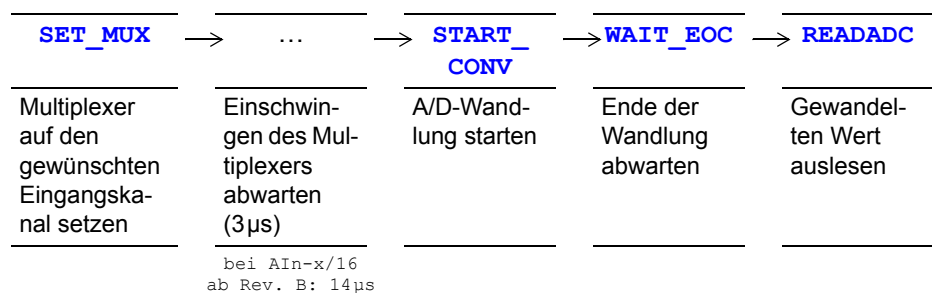
ret_val = ADC(module, input_no)
```

Parameter

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

Bemerkungen

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



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

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



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

Ein niedrig priorisierter Prozess kann inmitten der **ADC**-Sequenz von einem höher priorisierten Prozess unterbrochen werden. Wenn der höher priorisierte Prozess den Multiplexer umstellt, liefert die Funktion im niedrig priorisierten Prozess den Messwert des falschen Kanals.

Mehrere parallel laufende hoch priorisierte Prozesse können die Anweisung **ADC** problemlos verwenden, da hoch priorisierte Prozesse nicht unterbrochen werden können.

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

Siehe auch

[SET_MUX](#), [START_CONV](#), [WAIT_EOC](#), [READADC](#), [ADC16](#), [SE_DIFF](#)

Gültig für Module

(LP)SH-4/8(-FI), Aln-16/14-C Rev. A, Aln-32/12 Rev. A, Aln-32/12 Rev. B, Aln-32/14 Rev. A, Aln-32/16 Rev. B, Aln-32/16 Rev. C, Aln-8/12 Rev. A, Aln-8/12 Rev. B, Aln-8/14 Rev. A, Aln-8/16 Rev. B, Aln-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
DIM value AS LONG
```

```
EVENT:
```

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

ADC16

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = ADC16(module, input_no)
```

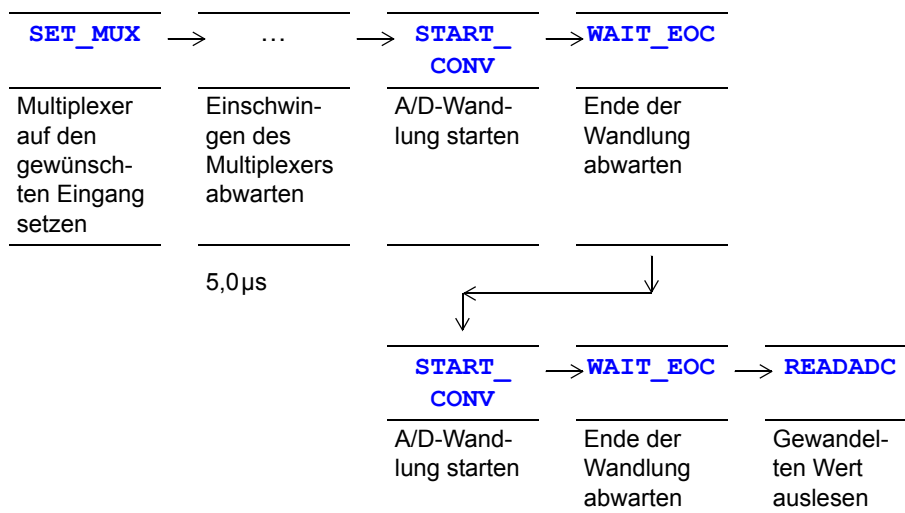
Parameter

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

Bemerkungen

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

Die Anweisung **ADC16** ist eine Zusammenstellung von aufeinander folgenden Funktionen:



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

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

Wandlung 2fach starten

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

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

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

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



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

Siehe auch

[SET_MUX](#), [START_CONV](#), [WAIT_EOC](#), [READADC](#), [ADC](#)

Gültig für Module

(LP)SH-4/8(-FI), Aln-8/16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
DIM value AS LONG
```

```
EVENT:
```

```
value = ADC16(1, 7)      'Misst einen Wert am analogen Eingg. 7
```



ADCF

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

Syntax

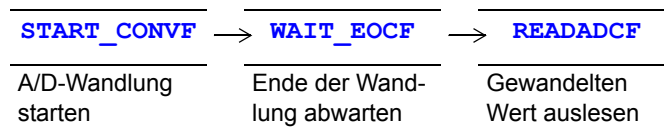
```
#INCLUDE ADwinPRO_ALL.inc
ret_val = ADCF(module, input_no)
```

Parameter

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

Bemerkungen

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



Siehe auch

[START_CONV](#), [WAIT_EOCF](#), [READADCF](#), [READ_ADCF4](#), [READ_ADCF8](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

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

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = BURST_ABORT(module)
```

Parameter

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

Bemerkungen

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

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

Siehe auch

[BURST_INIT](#), [BURST_READ](#), [BURST_START](#), [BURST_STATUS](#)

Gültig für Module

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

BURST_ABORT

Beispiel

```
REM Messung im hochprioren Prozess, Auslesen im niederprioren
REM Abschnitt FINISH:. Abbruch der Messung bei
REM einem Triggersignal am DIO32 Modul Nr.1
#include ADwinPRO_ALL.inc

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

DIM DATA_1[samples] AS LONG
DIM DATA_2[samples] AS LONG
DIM DATA_num AS LONG      'Anzahl der gewandelten Messwerte
DIM run_state AS LONG      'Ablaufstatus der Messreihe:
                           'steht/läuft/abgeschlossen
DIM cancel AS LONG         'Merker, ob Abbruch gewünscht

INIT:
  BURST_INIT(ainadr,1,sampleperiod,samples)
                           'Adresse, Modus, Messrate, Anzahl
                           'Messungen setzen
  run_state=0              'Status setzen: Messreihe steht
  DIGPROG1(dioadr,0)       'DIO 32: Bit 0...15 als Eingang

EVENT:
  IF (run_state=0) THEN    'wenn keine Messreihe läuft
    BURST_START(ainadr)    'dann Messreihe starten
    run_state=1            'Messreihe läuft
  ENDIF
  IF (run_state=1) THEN    'wenn Messreihe läuft
    DATA_num=BURST_STATUS(ainadr) 'Anzahl restl. Messg. ermitteln
    cancel=DIGIN_WORD1(dioadr) 'Abbruch von extern gewünscht?
    IF (cancel AND 1=1) THEN 'Abbruchkriterium erfüllt
      DATA_num=BURST_ABORT(ainadr) 'Messreihe abbr./Anz. Messwerte
    END
    ENDIF
    IF (DATA_num=0) THEN END 'wenn alle Messungen fertig: beenden
  ENDIF

FINISH: 'Messwerte abholen im niederprioren Abschnitt
  BURST_READ(ainadr,1,1,DATA_num,DATA_1,1)
```

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
BURST_CREAD(module, channel, count, array[], arr_idx)
```

Parameter

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

Bemerkungen

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

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

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

Siehe auch

[BURST_INIT](#), [BURST_CSTART](#), [BURST_STATUS](#), [SET_GAIN](#), [SYNC_MODE](#)

Gültig für Module

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

BURST_CREAD



Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE count 10000
#DEFINE module 1
#DEFINE sampleperiod 20    '2000 kHz Messrate [=1/(25ns*20)]
DIM DATA_1[count] AS LONG

INIT:
    REM Adresse, Modus, Messrate, Anzahl Messungen
    BURST_INIT(module,1,sampleperiod,count)
    REM kontinuierliche Burst-Messung starten
    BURST_CSTART(module)
    PROCESSDELAY=80          'Mit 500 kHz den Trigger-Punkt finden

EVENT:
    PAR_1=READADCF(module,1) 'Den aktuellsten Messwert holen
    IF (PAR_1>49152) THEN    'Sind +5V überschritten?
        PAR_9=BURST_ABORT(module) 'Messung abbrechen und
        END                      'Prozess beenden (=FINISH ausführen)
    ENDIF

FINISH:
    REM Die zuletzt gesammelten Daten in DATA_1 kopieren
    BURST_CREAD(module,1,count,DATA_1,1)
```

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
BURST_CSTART(module)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

Bemerkungen

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

Die gespeicherten Messwerte werden nach Abschluss der Messreihe mit **BURST_CREAD** ausgelesen (nicht mit **BURST_READ**).

Mit dem Befehl ist es beispielsweise möglich, eine Anzahl von Messwerten direkt vor oder nach dem Eintreten einer bestimmten Trigger-Bedingung zu erfassen (und weiter zu verarbeiten).

Hierzu wird die Messung mit **BURST_ABORT** sofort beim (oder um eine bestimmte Zeitspanne nach) Eintreten der Bedingung gestoppt.



Siehe auch

[BURST_INIT](#), [BURST_CREAD](#), [BURST_STATUS](#), [SET_GAIN](#), [SYNC_MODE](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE count 10000
#DEFINE module 1
#DEFINE sampleperiod 20      '2000 kHz Messrate [=1/(25ns*20)]
DIM DATA_1[count] AS LONG

INIT:
    BURST_INIT(module,1,sampleperiod,count) 'Adresse, Modus,
                                           'Messrate, Anzahl Messungen
    BURST_CSTART(module)                   'kontin. Burst-Messreihe starten
    PROCESSDELAY=80                        'Mit 500 kHz den Trigger-Punkt finden

EVENT:
    PAR_1=READADCF(module,1) 'Den aktuellsten Messwert holen
    IF (PAR_1>49152) THEN      'Sind +5V überschritten?
        PAR_9=BURST_ABORT(module) 'Messung abbrechen und
        END                        'Prozess beenden (=FINISH ausführen)
    ENDIF

FINISH:
    REM Die zuletzt gesammelten Daten in DATA_1 kopieren
    BURST_CREAD(module,1,count,DATA_1,1)
```

BURST_INIT

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

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
BURST_INIT (module, mode, pulses, samples)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

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

mode	Kanalnr.	max. Anzahl der Messwerte pro Kanal:
n	$1 \dots 2^{(n-1)}$	$2^{(21-n)} - 1$
1	1	$220 - 1 = 1048575 = 0FFFFFFH$
2	1...2	$219 - 1 = 524287 = 7FFFFFFH$
3	1...4	$218 - 1 = 262143 = 3FFFFFFH$
4	1...8	$217 - 1 = 131071 = 1FFFFFFH$

pulses legt die Periodendauer für eine Messreihe fest als Anzahl der Zeittakte: Periodendauer = **pulses** * 25ns (Mind.wert 20, entspricht 2MHz). Eine Periodendauer ist die Zeit vom Beginn einer Messung bis zum Beginn der nächsten Messung. LONG

samples Anzahl der durchzuführenden Messungen pro Kanal (der Maximalwert für **samples** wird durch **mode** bestimmt). LONG

Bemerkungen

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

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

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

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



Siehe auch

BURST_ABORT, BURST_READ, BURST_READ_PACKED, BURST_START, BURST_STATUS, READADCF, SET_GAIN, SYNC_MODE

Gültig für Module

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

Beispiel

REM Messung im hochprioren Prozess; sobald eine Spannung größer
REM 5V gemessen wird, Umschaltung in Burst-Modus und messen
REM mit 1,0 MHz.

REM Messwerte auslesen im niederprioren Abschnitt FINISH:

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE samples 10000      'Anzahl durchzuführende Messungen
#DEFINE ainadr 1           'Adresse AIn Modul
#DEFINE sampleperiod 40    '1,0 MHz Messrate [=1/(25ns*40)]
DIM DATA_1[samples] AS LONG
DIM DATA_2[samples] AS LONG
DIM dig_value AS LONG      'Anliegender Spannungswert
DIM remaining AS LONG      'Anzahl der restlichen Messwerte
DIM run_state AS LONG      'Ablaufstatus der Messreihe:
                           'steht/läuft/abgeschlossen

INIT:
  run_state=0              'Status setzen: Messreihe steht

EVENT:
  IF (run_state=0) THEN    'keine Messreihe läuft?
    dig_value =ADCF(ainadr,1)
    IF (dig_value>49151) THEN 'Spannung >5 V
      'Adresse, Modus, Messrate, Anzahl Messungen
      BURST_INIT(ainadr,2,sampleperiod,samples)
      BURST_START(ainadr) 'dann Messreihe starten
      run_state=1         'Messreihe läuft
    ENDIF
  ENDIF
  IF (run_state=1) THEN    'Messreihe läuft?
    remaining=BURST_STATUS(ainadr) 'Anzahl restliche Messungen
    'ermitteln
    IF (remaining=0) THEN END 'alle Messungen durchgeführt?
  ENDIF

FINISH: 'Daten auslesen im niederprioren Abschnitt FINISH:
  BURST_READ(ainadr,1,1,samples,DATA_1,1) 'Messwerte von Kanal 1
    'auslesen
  BURST_READ(ainadr,2,1,samples,DATA_2,1) 'Messwerte von Kanal 2
    'auslesen
```

BURST_READ

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

Die Anzahl der zu kopierenden Messwerte muss angegeben werden.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
BURST_READ(module, channel, startadr, count,  
            array[], array_idx)
```

Parameter

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

Bemerkungen

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

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

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

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

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

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

Siehe auch

[BURST_ABORT](#), [BURST_INIT](#), [BURST_READ_PACKED](#), [BURST_START](#), [BURST_STATUS](#), [SET_GAIN](#), [SYNC_MODE](#)

Gültig für Module

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



Beispiel

```

REM Achtung: Messung im hochprioren Prozess, Auslesen im
REM niederprioren Abschnitt!
REM Startet bei positiver Flanke an DI0 Bit 0 eine Analogmessung
REM an Kanal 1
#include ADwinPRO_ALL.inc
#define samples 10000      'Anzahl durchzuführende Messungen
#define sampleperiod 30    '1,33 MHz Messrate [=1/(25ns*30)]
#define dioadr 1           'Moduladresse DIO Modul
#define ainadr 1           'Moduladresse AIN Modul
#define run_state PAR_1    'Ablaufstatus der Messreihe:
                           'steht/läuft/abgeschlossen

DIM DATA_1[samples] AS LONG
DIM remaining AS LONG      'Anzahl der restlichen Messwerte
DIM i AS LONG
DIM trigger AS LONG        'Merker für externes Trigger-Signal

INIT:
  DIGPROG1(dioadr,0)       'DIO Bit 0...15 als Input
  run_state=0              'Status setzen: Messreihe steht
  BURST_INIT(ainadr,1,sampleperiod,samples)
                           'Adresse,Modus,Messrate,Anz. Messng.

EVENT:
  IF (run_state=0) THEN    'wenn keine Messreihe läuft
    trigger=DIGIN_WORD1(dioadr) 'Trigger-Signal lesen
    IF (trigger AND 1=1) THEN 'Trigger vorhanden?
      BURST_START(ainadr)    'dann Messreihe starten
      run_state=1           'Messreihe läuft
    ENDIF
  ENDIF
  IF (run_state=1) THEN    'wenn Messreihe läuft
    remaining=BURST_STATUS(ainadr) 'Anzahl restliche Messungen
    IF (remaining=0) THEN END 'alle Messungen durchgeführt?
    '... dann Ende
  ENDIF

FINISH: 'Daten auslesen im niederprioren Abschnitt FINISH:
  BURST_READ(ainadr,1,1,samples,DATA_1,1)
                           'Moduladr., Kanal, Startadr., Länge,
                           'Zielfeld, Startindex im Zielfeld

```

BURST_READ_PACKED

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

Die Anzahl der zu kopierenden Messwerte muss angegeben werden.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

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

Parameter

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

Bemerkungen

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

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

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

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

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

In einem hoch priorisierten Prozess dürfen Sie mit **BURST_READ_PACKED** nur so viele Daten auf einmal vom Modul lesen, dass die Auslastung des ADwin-Systems nicht über 100% steigt. Falls dies dennoch geschieht, kann die Kommunikation mit dem PC in einen undefinierten



Zustand geraten (Time-out). Diese Einschränkung besteht nicht im Abschnitt **LOWINIT**: und bei niederprioren Prozessen.

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

Siehe auch

BURST_ABORT, **BURST_INIT**, **BURST_READ**, **BURST_START**,
BURST_STATUS, **SET_GAIN**, **SYNC_MODE**

Gültig für Module

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

Beispiel

```
REM Achtung: Messung im hochprioren Prozess, Auslesen im
REM niederprioren Abschnitt!
#include ADwinPRO_ALL.inc
#define samples 10000      'Anzahl durchzuführende Messungen
#define ainadr 2           'Adresse AIN Modul
#define sampleperiod 40    '1 MHz Messrate [=1/(25ns*40)]
DIM DATA_1[samples] AS LONG
DIM DATA_2[samples] AS LONG
DIM remaining AS LONG      'Anzahl der restlichen Messwerte
DIM run_state AS LONG      'Ablaufstatus der Messreihe:
                           'steht/läuft/abgeschlossen

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

EVENT:
  IF (run_state=0) THEN    'wenn keine Messreihe läuft
    BURST_START(ainadr)    'dann Messreihe starten
    run_state=1            'Messreihe läuft
  ENDIF
  IF (run_state=1) THEN    'wenn Messreihe läuft
    remaining=BURST_STATUS(ainadr) 'Anzahl restliche Messungen
    IF (remaining=0) THEN  run_state=2 'Messreihe abgeschlossen,
                           'Merker setzen
  ENDIF

FINISH: 'Daten auslesen im niederprioren Abschnitt FINISH:
  BURST_READ_PACKED(ainadr,1,1,samples,DATA_1,1) 'Messwerte von
                           'Kanal 1 lesen
  BURST_READ_PACKED(ainadr,2,1,samples,DATA_2,1) 'Messwerte von
                           'Kanal 2 lesen
```

BURST_START

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
BURST_START(module)
```

Parameter

module Eingestellte Moduladresse (1...255).

LONG

Bemerkungen

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

1. Burst-Messreihe synchron zu anderen Messungen starten:

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

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

2. Die Messungen mehrerer Burst-Messreihen synchron ausführen:

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

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

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

Siehe auch

[BURST_ABORT](#), [BURST_INIT](#), [BURST_READ](#), [BURST_READ_PACKED](#), [BURST_STATUS](#), [SET_GAIN](#), [SYNCENABLE](#), [SYNC_MODE](#)

Gültig für Module

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



Beispiel

```

REM Messung im hochprioren Prozess; sobald eine Spannung größer
REM 5 V gemessen wird, Umschaltung in Burst-Modus und messen mit
REM 1,0 MHz.
REM Messwerte auslesen im niederprioren Abschnitt FINISH:
#INCLUDE ADwinPRO_ALL.inc
#DEFINE samples 10000      'Anzahl durchzuführende Messungen
#DEFINE ainadr 1           'Adresse AIn Modul
#DEFINE sampleperiod 40    '1 MHz Messrate [=1/(25ns*40)]
DIM DATA_1[samples] AS LONG
DIM DATA_2[samples] AS LONG
DIM dig_value AS LONG     'Anliegender Spannungswert
DIM remaining AS LONG     'Anzahl der restlichen Messwerte
DIM run_state AS LONG     'Ablaufstatus der Messreihe:
                             'steht/läuft/abgeschlossen

INIT:
    run_state=0              'Status setzen: Messreihe steht

EVENT:
    IF (run_state=0) THEN    'wenn keine Messreihe läuft
        volt_value =ADCF(ainadr,1)
        IF (volt_value>49151) THEN 'Spannung >5 V
            BURST_INIT(ainadr,2,sampleperiod,samples) 'Adresse, Modus,
                                                         'Messrate, Anzahl Messungen
            BURST_START(ainadr) 'dann Messreihe starten
            run_state=1        'Messreihe läuft
        ENDIF
    ELSE                    'wenn Messreihe läuft
        remaining=BURST_STATUS(ainadr) 'Anzahl restliche Messungen
                                         'ermitteln
        IF (remaining=0) THEN END 'alle Messungen durchgeführt
    ENDIF

FINISH: 'Daten auslesen im niederprioren Abschnitt FINISH:
    BURST_READ(ainadr,1,1,samples,DATA_1,1) 'Messwerte von
                                              'Kanal 1 auslesen
    BURST_READ(ainadr,2,1,samples,DATA_2,1) 'Messwerte von
                                              'Kanal 2 auslesen

```

BURST_STATUS

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = BURST_STATUS(module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>ret_val</code>	Anzahl der noch auszuführenden Messungen.	LONG

Bemerkungen

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

Siehe auch

[BURST_ABORT](#), [BURST_INIT](#), [BURST_CSTART](#), [BURST_CREAD](#),
[BURST_READ](#), [BURST_READ_PACKED](#), [BURST_START](#), [SET_GAIN](#), [SYNC_MODE](#)

Gültig für Module

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

Beispiel

```

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

DIM DATA_1[samples] AS LONG
DIM DATA_2[samples] AS LONG
DIM volt_value AS LONG 'Anliegender Spannungswert
DIM remaining AS LONG 'Anzahl der restlichen Messwerte
DIM run_state AS LONG 'Ablaufstatus der Messreihe:
                        'steht/läuft/abgeschlossen

INIT:
    run_state=0 'Status setzen: Messreihe steht

EVENT:
    IF (run_state=0) THEN 'wenn keine Messreihe läuft
        volt_value =ADCF(ainadr,1)
        IF (volt_value>49151) THEN 'Spannung >5 V
            BURST_INIT(ainadr,2,sampleperiod,samples) 'Adresse, Modus,
                'Messrate, Anzahl Messungen
            BURST_START(ainadr) 'dann Messreihe starten
            run_state=1 'Messreihe läuft
        ENDIF
    ENDIF
    IF (run_state=1) THEN 'wenn Messreihe läuft
        remaining=BURST_STATUS(ainadr) 'Anzahl restliche Messungen
        'ermitteln
        IF (remaining=0) THEN END 'alle Messungen durchgeführt
    ENDIF

FINISH: 'Daten auslesen im niederprioren Abschnitt FINISH:
    BURST_READ(ainadr,1,1,samples,DATA_1,1) 'Messwerte von
        'Kanal 1 auslesen
    BURST_READ(ainadr,2,1,samples,DATA_2,1) 'Messwerte von
        'Kanal 2 auslesen

```

READADC

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = READADC(module)
```

Parameter

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

Bemerkungen

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

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

Siehe auch

[ADC](#), [ADC16](#), [READADC_SCONV](#), [SET_MUX](#), [START_CONV](#), [SYNCALL](#), [WAIT_EOC](#)

Gültig für Module

(LP)SH-4/8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value1 AS LONG           'Deklaration

EVENT:
    SET_MUX(1,0)              'Multiplexer auf Eingang 1 setzen
    ...                       '3µs (12-Bit-ADC) bzw. 14µs
                                '(AIn-8/16 Rev.B, AIn-32/16 Rev.B)
                                'auf das Einschwingen des
                                'Multiplexers warten*

    START_CONV(1)              'Start AD-Wandlung
    WAIT_EOC(1)                'Warten auf Wandlung-Ende
    value1 = READADC(1)        'Wert vom ADC einlesen
```

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

Besteht keine Notwendigkeit, die Wartezeit mit Befehlen zu nutzen, so kann die folgende Warteschleife programmiert werden, welche jedoch nur in hoch priorisierten Prozessen (ab T9) angewendet werden darf:

```
START_CONV(1)                'Start AD-Wandlung
PAR_80 = READ_TIMER()
DO
UNTIL (READ_TIMER() - PAR_80 > 560) '25ns*560=14µs warten
```

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = READADC_SCONV(module)
```

Parameter

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

Bemerkungen

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

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

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



Siehe auch

[ADC](#), [ADC16](#), [READADC](#), [SE_DIFF](#), [SET_MUX](#), [START_CONV](#), [WAIT_EOC](#)

Gültig für Module

(LP)SH-4/8(-FI), Aln-16/14-C Rev. A, Aln-32/12 Rev. B, Aln-32/14 Rev. A, Aln-32/16 Rev. B, Aln-32/16 Rev. C, Aln-8/12 Rev. B, Aln-8/14 Rev. A, Aln-8/16 Rev. A, Aln-8/16 Rev. B, Aln-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM i AS LONG
DIM DATA_1[1000] AS LONG 'Deklaration

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

EVENT:
  WAIT_EOC(1) 'Wandlungsende abwarten
  DATA_1[i] = READADC_SCONV(1) 'A/D-Wandler auslesen und starten
  INC(i) 'Index erhöhen
  IF (i=1001) THEN END 'Nach 1000 Messwerten Prozess beenden
```

READADCF

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = READADCF(module, adc_no)
```

Parameter

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

Bemerkungen

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

Alternativ stehen die Befehle **READ_ADCF4**, **READ_ADCF8**, **READ_ADCF4_PACKED**, **READ_ADCF8_PACKED** zur Verfügung, die mehrere Ergebnisse sehr schnell auslesen.

Siehe auch

[ADCF](#), [START_CONV](#), [WAIT_EOCF](#), [READADCF_32](#), [READADCF_SCONV](#), [READADCF_SCONV_32](#), [READ_ADCF4](#), [READ_ADCF8](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM value1 AS LONG 'Deklaration  
  
EVENT:  
  START_CONV(1,1) 'Start AD-Wandlung  
  WAIT_EOCF(1,1) 'Warten auf Wandlung-Ende  
  value1 = READADCF(1,1) 'Wert vom ADC einlesen
```

READ_ADCF4 liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus.

Syntax

```
#INCLUDE ADWINPRO_ALL.INC

READ_ADCF4(module, array[], index)
```

Parameter

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

Bemerkungen

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

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

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

Siehe auch

[ADCF](#), [START_CONV](#), [READADCF](#), [READ_ADCF8](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#), [READADCF_SCONV](#), [WAIT_EOCF](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC
DIM value[4] AS LONG      'Feld für Messwerte

INIT:
    START_CONV(1, 0Fh)      'Start AD-Wandlung Kanäle 1...4

EVENT:
    WAIT_EOCF(1, 0Fh)      'Warten auf Wandlungsende
    READ_ADCF4(1, value, 1) 'Werte der ADC 1...4 lesen
    START_CONV(1, 0Fh)      'Neue AD-Wandlung starten
```

READ_ADCF4

READ_ADCF8

READ_ADCF8 liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus.

Syntax

```
#INCLUDE ADWINPRO_ALL.INC  
READ_ADCF8(module,array[],index)
```

Parameter

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

Bemerkungen

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

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

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

Siehe auch

[ADCF](#), [START_CONV](#), [WAIT_EOCF](#), [READ_ADCF4](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#), [READADCF_SCONV](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC  
DIM value[8] AS LONG      'Feld für Messwerte  
  
EVENT:  
    START_CONV(1,0FFh)    'Start AD-Wandlung Kanäle 1...8  
  
EVENT:  
    WAIT_EOCF(1,0FFh)     'Warten auf Wandlungsende  
    READ_ADCF8(1,value,1) 'Werte der ADC 1...8 lesen  
    START_CONV(1,0FFh)    'Neue AD-Wandlung starten
```

READ_ADCF4_PACKED liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus. Je 2 Messwerte werden gemeinsam in einem 32 Bit-Wert zurückgegeben.

Syntax

```
#INCLUDE ADWINPRO_ALL.INC

READ_ADCF4_PACKED(module, array[], index)
```

Parameter

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

Bemerkungen

Es werden immer die Messwerte der F-ADC 1...4 des Moduls gelesen. 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
<code>index</code>	F-ADC 2	F-ADC 1
<code>index+1</code>	F-ADC 4	F-ADC 3

Die Anweisung ist schneller als das Auslesen mit **READ_ADCF4**.

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

Siehe auch

[ADCF](#), [START_CONV](#), [WAIT_EOCF](#), [READ_ADCF4](#), [READ_ADCF8](#), [READ_ADCF8_PACKED](#), [READ_ADCF_SCONV](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC
DIM value[2] AS LONG      'Feld für Messwerte

INIT:
    START_CONV(1, 0Fh)      'Start AD-Wandlung Kanäle 1...4

EVENT:
    WAIT_EOCF(1, 0Fh)      'Warten auf Wandlungsende
    READ_ADCF4_PACKED(1, value, 1) 'Werte der ADC 1...4 lesen
    START_CONV(1, 0Fh)      'Neue AD-Wandlung starten
```

READ_ADCF4_PACKED

READ_ADCF8_PACKED

READ_ADCF8_PACKED liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus. Je 2 Messwerte werden gemeinsam in einem 32 Bit-Wert zurückgegeben.

Syntax

```
#INCLUDE ADWINPRO_ALL.INC

READ_ADCF8_PACKED (module, array[], index)
```

Parameter

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

Bemerkungen

Es werden immer die Messwerte der F-ADC 1...8 des Moduls gelesen. 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
<code>index</code>	F-ADC 2	F-ADC 1
<code>index+1</code>	F-ADC 4	F-ADC 3
<code>index+2</code>	F-ADC 6	F-ADC 5
<code>index+3</code>	F-ADC 8	F-ADC 7

Die Anweisung ist schneller als das Auslesen mit **READ_ADCF8**.

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

Siehe auch

[ADCF](#), [START_CONV](#), [WAIT_EOCF](#), [READ_ADCF4](#), [READ_ADCF8](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#), [READ_ADCF4_SCONV](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC
DIM value[4] AS LONG      'Feld für Messwerte

INIT:
  START_CONV(1, 0FFh)      'Start AD-Wandlung Kanäle 1...8

EVENT:
  WAIT_EOCF(1, 0FFh)      'Warten auf Wandlungsende
  READ_ADCF8_PACKED(1, value, 1) 'Werte der ADC 1...8 lesen
  START_CONV(1, 0FFh)      'Neue AD-Wandlung starten
```


READADCF_32 liest die Wandlungsergebnisse aus 2 aufeinander folgenden F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = READADCF_32(module, adc_no)
```

Parameter

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

Bemerkungen

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

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

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

Siehe auch

ADCF, READADCF, READ_ADCF4, READ_ADCF8, READ_ADCF4_PACKED, READ_ADCF8_PACKED, READADCF_SCONV, READADCF_SCONV_32, START_CONVF, WAIT_EOCF

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value1 AS LONG 'Deklaration

EVENT:
START_CONVF(1,3) 'Start AD-Wandlung auf ADC1 und ADC2
WAIT_EOCF(1,3) 'Warten auf das Ende der Wandlungen
value1 = READADCF_32(1,1) 'Wert von ADC1 und ADC2 einlesen
```

READADCF_32

READADCF_ SCONV

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = READADCF_SCONV(module, adc_no)
```

Parameter

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

Bemerkungen

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

Siehe auch

[ADCF](#), [READADCF](#), [READ_ADCF4](#), [READ_ADCF8](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#), [READADCF_32](#), [READADCF_SCONV_32](#), [START_CONVF](#), [WAIT_EOCF](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM i AS LONG  
DIM DATA_1[1000] AS LONG 'Deklaration  
  
INIT:  
  i=1  
  START_CONVF(1,1) 'A/D-Wandler starten  
  
EVENT:  
  WAIT_EOCF(1,1)  
  DATA_1[i] = READADCF_SCONV(1,1) 'A/D-Wandler auslesen + starten  
  INC(i) 'Index erhöhen  
  IF (i=1001) THEN END 'Nach 1000 Messwerten Prozess beenden
```

READADCF_SCONV_32 liest die Wandlungsergebnisse aus 2 F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.

Anschließend wird sofort eine neue Konvertierung gestartet.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = READADCF_SCONV_32(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG										
adc_no	Nummer des ersten zu lesenden F-ADC (1...2 oder 1...4).	LONG										
	<table><tr><td>adc_no</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>F-ADC-Nr.</td><td>1, 2</td><td>3, 4</td><td>5, 6</td><td>7, 8</td></tr></table>	adc_no	1	2	3	4	F-ADC-Nr.	1, 2	3, 4	5, 6	7, 8	
adc_no	1	2	3	4								
F-ADC-Nr.	1, 2	3, 4	5, 6	7, 8								
ret_val	Der Rückgabewert (32 Bit) enthält die Messdaten von 2 aufeinanderfolgenden F-ADC (je 16 Bit: 0...65535); je ein Messwert ist im unteren und im oberen Wort.	LONG										

Beispiel

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

Siehe auch

[ADCF](#), [READADCF](#), [READ_ADCF4](#), [READ_ADCF8](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#), [READADCF_32](#), [READADCF_SCONV](#), [START_CONV](#), [WAIT_EOCF](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG 'Deklaration

INIT:
    START_CONV(1,3) 'Start AD-Wandlung

EVENT:
    WAIT_EOCF(1,3) 'Warten auf das Ende der Konvertierung
    value = READADCF_SCONV_32(1,1) 'Wert vom ADC1 und ADC2
    'einlesen und die Wandlung
    'beider ADC neu starten
```

READADCF_SCONV_32

SE_DIFF

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
SE_DIFF(module, choice)
```

Parameter

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

Bemerkungen

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

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

Siehe auch

[ADC](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
  
INIT:  
  SE_DIFF(1,0)           'Modul mit der Adresse 1 wird  
                           'auf SE gesetzt  
  SE_DIFF(2,1)           'Modul mit der Adresse 2 wird  
                           'auf DIFF gesetzt
```



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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

SET_GAIN(module, channel, mode)
```

Parameter

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

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

Siehe auch

ADCF, BURST_CSTART, BURST_START, READADCF, START_CONV, WAIT_EOCF

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
#define ainadr 1 'Moduladresse AIN Modul

INIT:
    SET_GAIN(ainadr, 4, 1)    'Spannungsbereich im Kanal 4 auf
                             'Betriebsart 1 stellen
                             '(Messbereich: +5V...-5V)

EVENT:
    PAR_1 = ADCF(1, 4)       'Misst einen Wert vom analogen Eingang
                             '4
```

SET_GAIN

SET_MUX

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
SET_MUX(module, pattern)
```

Parameter

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

		Bit-Nr.				
31:7	6	5	4	3	2	1 0
ohne Funktion	Verstärkung		Multiplexer-Eingang			
	1 = 00b		Eingang 1: 00000b			
	2 = 01b		Eingang 2: 00001b			
	4 = 10b		Eingang 3: 00010b			
	8 = 11b		...			
			Eingang 32: 11111b			

Bemerkungen

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

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

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

Siehe auch

[ADC](#), [ADC16](#), [START_CONV](#), [WAIT_EOC](#), [READADC](#)

Gültig für Module

(LP)SH-4/8(-FI), Aln-16/14-C Rev. A, Aln-32/12 Rev. A, Aln-32/12 Rev. B, Aln-32/14 Rev. A, Aln-32/16 Rev. B, Aln-32/16 Rev. C, Aln-8/12 Rev. A, Aln-8/12 Rev. B, Aln-8/14 Rev. A, Aln-8/16 Rev. A, Aln-8/16 Rev. B, Aln-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value1 AS LONG      'Deklaration

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

SEQ_MODE

SEQ_MODE initialisiert das angegebene Modul für den Betrieb mit Ablaufsteuerung.

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
SEQ_MODE(module, mode, gain)
```

Parameter

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

Bemerkungen

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

Die Modi unterscheiden sich wie folgt:

Modus	Art der Messung
0 Normal:	Standard: Einzelne Messung an einem Kanal (siehe ADC).
1 „single shot“:	Die Ablaufsteuerung wird beendet, sobald die gewählten Kanäle je einmal gewandelt sind.
3 „continuous“:	Die Ablaufsteuerung wandelt ununterbrochen neue Messwerte an den gewählten Kanälen.

Es können bis zu 32 Messwerte im moduleigenen Speicher abgelegt werden.

Bei 14 Bit-Wandlern wird das Messergebnis linksbündig in einem 16 Bit-Wert zurückgegeben, d.h. die 2 niederwertigsten Bits sind immer Null.

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

Siehe auch

[ADC](#), [SEQ_SELECT](#), [SEQ_SET_DELAY](#), [SEQ_STATUS](#), [SEQ_READ](#)

Gültig für Module

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



Beispiel

```
#DEFINE module 1
#include ADwinPRO_ALL.inc

DIM DATA_1[16] AS LONG AT DM_LOCAL

INIT:
    SE_DIFF(module,0)           'Eingänge auf single ended stellen
    SEQ_MODE(module,3,0)        'Ablaufsteuerung: Continuous Mode,
                                'Verstärkungsfaktor 1
    SEQ_SELECT(module,55555555h) 'Alle ungeradzahligen Kanäle des
                                'Moduls AIN-32/.. messen
    START_CONV(1)               'Messesequenzen starten
                                ' (Continuous Mode)
    WAIT_EOC(1)                 'Warten, bis alle angegebenen Kanäle
                                'einmal gemessen sind

EVENT:
    SEQ_READ(module,16,DATA_1,1) 'Messwerte von dem Modul holen und
                                'in DATA_1 kopieren
```

SEQ_READ

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

In jedes Feldelement wird jeweils 1 Messwert kopiert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

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

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
count	Anzahl der zu lesenden Messwerte (1...32).	LONG
array[]	Ziel-Feld, in das die Messwerte übertragen werden.	ARRAY LONG
array_idx	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1...n).	LONG

Bemerkungen

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

Die Messwerte der Messgruppe (siehe **SEQ_SELECT**) werden von der kleinsten Kanalnummer an in aufsteigender Reihenfolge in das Zielfeld kopiert.

Siehe auch

[SEQ_READ_ONE](#), [SEQ_READ_TWO](#), [SEQ_READ_PACKED](#), [SEQ_READ32](#), [SEQ_MODE](#), [SEQ_READ](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[16] AS LONG AT DM_LOCAL

INIT:
    SE_DIFF(1,0)           'Single-Ended Eingänge
    SEQ_MODE(1,3,0)        'Ablaufsteuerung auf Continuous Mode,
                           'Verstärkungsfaktor 1
    SEQ_SELECT(1,5555555h) 'Alle ungeradzahligten Kanäle einer
                           'AIN-32/.. messen
    START_CONV(1)          'Messsequenzen im Continuous Mode
                           'starten
    WAIT_EOC(1)            'Warten, bis einmal alle angegebenen
                           'Kanäle gemessen wurden

EVENT:
    SEQ_READ(1,16,DATA_1,1) 'Aktuelle Messwerte von dem Modul
                           'in DATA_1 umkopieren
```

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = SEQ_READ_ONE(module,idxno)
```

Parameter

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

Bemerkungen

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

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

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

Siehe auch

[SEQ_MODE](#), [SEQ_SELECT](#), [SEQ_STATUS](#), [SEQ_READ](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[32] AS FLOAT AT DM_LOCAL
DIM i AS LONG

INIT:
    SE_DIFF(1,0)           'Single-Ended Eingänge
    SEQ_MODE(1,1,0)        'Ablaufsteuerung auf Single Shot,
                           'Verstärkungsfaktor 1
    SEQ_SELECT(1,0FFFFFFFh) 'Alle Eingänge 1...32 einer AIN-32/..
                           'messen

EVENT:
    START_CONV(1)          'Messesequenz im Single Shot Modus
                           'starten
    FOR i=1 TO 32
        'Alle 32 Kanäle einer AIN-32/..
        'einlesen ...
        DO
            'sobald die jeweilige ...
            UNTIL(i<=SEQ_STATUS(1)) '... Einzelmessung fertig ist
            DATA_1[i]=(SEQ_READ_ONE(1,i)-32768)*20/65536
            'Digit in Volt umrechnen und speichern
        NEXT i
```

SEQ_READ_ONE

SEQ_READ_TWO

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = SEQ_READ_TWO(module,idxno)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

idxno Indexnummer, die die Position von je 2 Kanälen in der Messgruppe bezeichnet (0 ... 15). LONG

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

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

Bemerkungen

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

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

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

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

Siehe auch

[SEQ_MODE](#), [SEQ_SELECT](#), [SEQ_STATUS](#), [SEQ_READ](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[32] AS LONG AT DM_LOCAL
DIM i, value32 AS LONG

INIT:
  SE_DIFF(1,0)           'Single-Ended Eingänge
  SEQ_MODE(1,1,0)        'Ablaufsteuerung auf Single Shot,
                          'Verstärkungsfaktor 1
  SEQ_SELECT(1,0FFFFFFFh) 'Alle 32 Eing. der AIN-32/.. messen

EVENT:
  START_CONV(1)           'Messsequenz im Single Shot Modus
                          'starten
  FOR i=1 TO 15 STEP 2    'Alle 32 Kanäle einlesen
    DO                    'Sobald jeweils ...
      UNTIL( (i+1)<=SEQ_STATUS(1) ) '2 Messungen fertig sind:
        value32=SEQ_READ_TWO(1,(i-1)/2) 'Langwort abholen und
        'speichern
        DATA_1[i]=value32 AND 0FFFFh 'unteres Wort speichern
        DATA_1[i+1]=SHIFT_RIGHT(value32,16) 'oberes Wort speichern
      NEXT i
    WAIT_EOC(1)
```

SEQ_READ_PACKED

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

In jedes Feldelement werden jeweils 2 Messwerte kopiert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

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

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>count</code>	Anzahl der zu lesenden Messwerte-Paare (1...16).	LONG
<code>array[]</code>	Ziel-Feld, in das die Messwerte-Paare übertragen werden.	ARRAY LONG
<code>array_idx</code>	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden (1...n).	LONG

Bemerkungen

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

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

Siehe auch

[SEQ_READ_ONE](#), [SEQ_READ_TWO](#), [SEQ_READ_PACKED](#), [SEQ_READ32](#), [SEQ_MODE](#), [SEQ_READ](#), [SEQ_SELECT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[32] AS LONG AT DM_LOCAL

INIT:
    SE_DIFF(1,0)           'Single-Ended Eingänge
    SEQ_MODE(1,3,0)        'Ablaufsteuerung auf Continuous Mode,
                           'Verstärkungsfaktor 1
    SEQ_SELECT(1,0AAAAAAAh) 'Alle geradzahligen Kanäle einer
                           'AIN-32/.. messen
    START_CONV(1)          'Messsequenzen im Continuous Mode
                           'starten
    WAIT_EOC(1)            'Warten bis einmal alle angegebenen
                           'Kanäle gemessen wurden

EVENT:
    SEQ_READ_PACKED(1,8,DATA_1,1)
                           '16 Messwerte von dem Modul holen und
                           'in DATA_1 kopieren
```

SEQ_READ32 kopiert alle 32 Messwerte (je 16 Bit) von dem angegebenen Modul in ein Ziel-Feld.

In jedes Feldelement wird jeweils 1 Messwert kopiert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

SEQ_READ32 (module, array[], array_idx)
```

Parameter

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

Bemerkungen

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

Die Messwerte der Messgruppe (siehe **SEQ_SELECT**) werden von der kleinsten Kanalnummer an in aufsteigender Reihenfolge in das Zielfeld kopiert.

Wenn Sie mehr als sechzehn Messwerte benötigen, arbeitet diese Anweisung schneller als **SEQ_READ**, obwohl **SEQ_READ32** immer alle 32 Messwerte überträgt.

Siehe auch

[SEQ_READ_ONE](#), [SEQ_READ_TWO](#), [SEQ_READ_PACKED](#), [SEQ_READ32](#), [SEQ_MODE](#), [SEQ_READ](#), [SEQ_SELECT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[32] AS LONG AT DM_LOCAL

INIT:
    SE_DIFF(1,0)           'Single-Ended Eingänge
    SEQ_MODE(1,3,0)        'Ablaufsteuerung auf Continuous Mode,
                           'Verstärkungsfaktor 1
    SEQ_SELECT(1,0FFFFFFFh) 'Alle Kanäle einer AIN-32/.. messen
    START_CONV(1)          'Messesequenzen im Continuous Mode
                           'starten
    WAIT_EOC(1)            'Warten bis einmal alle angegebenen
                           'Kanäle gemessen wurden

EVENT:
    SEQ_READ32(1,DATA_1,1) 'Messwerte von dem Modul holen
                           'und in DATA_1 kopieren
```

SEQ_READ32

SEQ_SELECT

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
SEQ_SELECT(module, pattern)
```

Parameter

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

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

Bemerkungen

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

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

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

Siehe auch

[SE_DIFF](#), [SEQ_MODE](#), [SEQ_STATUS](#), [SEQ_READ](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[32] AS LONG AT DM_LOCAL

INIT:
    SE_DIFF(1,1)           'Differentielle Eingänge
    SEQ_MODE(1,1,0)        'Ablaufsteuerung auf Single Shot,
                           'Verstärkungsfaktor 1
    SEQ_SELECT(1,0FF00FFh) 'Diff. Eingänge 1-8 und 17-24 einer
                           'AIN-32/.. messen

EVENT:
    START_CONV(1)          'Messsequenz im Single Shot Modus
                           'starten
    ...                   'Hier könnte die Wartezeit sinnvoll
                           'genutzt werden
    WAIT_EOC(1)            'Warten bis einmal alle angegebenen
                           'Kanäle gemessen wurden
    SEQ_READ(1,16,DATA_1,1) 'Messwerte von dem Modul holen
                           'und in DATA_1 kopieren
```



SEQ_SET_DELAY legt die Einschwingzeit (Wartezeit zwischen 2 Messungen) der Ablaufsteuerung auf dem angegebenen Modul fest.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
SEQ_SET_DELAY(module,time)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>time</code>	Anzahl der Zeiteinheiten, aus der sich die Einschwingzeit der Ablaufsteuerung ergibt. 0: Werks-Einstellung für Wartezeit (Default). 1...2047: Wartezeit in Einheiten von 25ns.	LONG

Bemerkungen

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

Die Einschwingzeit berechnet sich nach folgender Formel:

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

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

Siehe auch

[SEQ_MODE](#), [SEQ_READ](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[32] AS LONG AT DM_LOCAL

INIT:
    SE_DIFF(1,1)           'Differentielle Eingänge
    SEQ_MODE(1,1,0)        'Ablaufsteuerung auf Single Shot
    SEQ_SELECT(1,0FF00FFh) 'Diff. Eing. 1-8 und 17-24 einer
                           'AIN-32/.. messen
    SEQ_SET_DELAY(1,200)   'Dem Analogteil tconv + 200 * 25ns
                           'Zeit geben, um einzuschwingen

EVENT:
    START_CONV(1)          'Messesequenz im Single Shot Modus
                           'starten
    ...                   'Hier könnte die Wartezeit sinnvoll
                           'genutzt werden
    WAIT_EOC(1)            'Warten, bis einmal alle angegebenen
                           'Kanäle gemessen wurden
    SEQ_READ(1,16,DATA_1,1) 'Messwerte von dem Modul holen
                           'und in DATA_1 kopieren
```

SEQ_SET_DELAY



SEQ_STATUS

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = SEQ_STATUS(module)
```

Parameter

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

Bemerkungen

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

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

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

Siehe auch

[SEQ_MODE](#), [SEQ_SELECT](#), [SEQ_STATUS](#), [SEQ_READ](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[32] AS FLOAT AT DM_LOCAL
DIM i AS LONG

INIT:
    SE_DIFF(1,0)           'Single-Ended Eingänge
    SEQ_MODE(1,1,0)        'Ablaufsteuerung auf Single Shot,
                           'Verstärkungsfaktor 1
    SEQ_SELECT(1,0FFFFFFFh) 'Alle 32 Eing. einer AIN-32/.. messen

EVENT:
    START_CONV(1)          'Messsequenz im Single Shot Modus
                           'starten
    FOR i=1 TO 32           'Alle 32 Kanäle einlesen ...
    DO                     'sobald die ...
        UNTIL (i<=SEQ_STATUS(1)) 'jeweilige Messung fertig ist ...
        DATA_1[i]=(SEQ_READ_ONE(1,i)-32768)*20/65536
                           'Digit in Volt umrechnen und speichern
    NEXT i
```

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

SH_SETMODE(module, mode)
```

Parameter

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

Bemerkungen

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

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

Siehe auch

-/-

Gültig für Module

(LP)SH-4/8(-FI)

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

EVENT:

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

SH_SETMODE

START_CONV

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
START_CONV (module)
```

Parameter

module Eingestellte Moduladresse (1...255).

LONG

Bemerkungen

Wenn das Modul mit **SYNCENABLE** zur Synchronisation freigeschaltet ist, hat die Anweisung **SYNCALL** die gleiche Funktion wie **START_CONV**.

Siehe auch

[ADC](#), [ADC16](#), [READADC](#), [SE_DIFF](#), [SET_MUX](#), [SYNCALL](#), [WAIT_EOC](#)

Gültig für Module

(LP)SH-4/8(-FI), Aln-16/14-C Rev. A, Aln-32/12 Rev. A, Aln-32/12 Rev. B, Aln-32/14 Rev. A, Aln-32/16 Rev. B, Aln-32/16 Rev. C, Aln-8/12 Rev. A, Aln-8/12 Rev. B, Aln-8/14 Rev. A, Aln-8/16 Rev. A, Aln-8/16 Rev. B, Aln-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM value1 AS LONG      'Deklaration
```

EVENT:

```
SET_MUX(1,0)      'Multiplexer auf Eingang 1 setzen  
3µs (12-Bit ADC) bzw. 14µs (Aln-8/16 Rev. B, Aln-32/16 Rev. B) auf das Ein-  
schwingen des Multiplexers warten*  
START_CONV(1)      'Start AD-Wandlung  
WAIT_EOC(1)      'Warten auf Wandlung-Ende  
value1 = READADC(1)      'Wert vom ADC einlesen
```

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

Eine weitere Möglichkeit zur Überbrückung bietet die folgende Warteschleife, welche jedoch nur in hoch priorisierten Prozessen angewendet werden darf:

```
time = READ_TIMER()      'Aktuellen Zählerstand ermitteln  
DO  
UNTIL (READ_TIMER()-time>120) '25ns*120=3µs warten
```

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

```
UNTIL (READ_TIMER()-time>560) '25ns*560=14µs warten
```

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
START_CONV(module, adc_no)
```

Parameter

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

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

Bemerkungen

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

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

Sie können mehrere Wandlungen ebenfalls synchron ausführen, wenn Sie die entsprechenden Module mit **SYNC_MODE** zur Synchronisation frei geben.

Sobald Sie auf dem Master-Modul eine Wandlung starten, starten zeitgleich auch Wandlungen auf allen Kanälen der Slave-Module. Bei Event-gesteuerten Modulen geschieht das Gleiche, sobald am gewünschten Event-Eingang ein Signal eingeht.

Siehe auch

[ADCF](#), [READADCF](#), [READ_ADCF4](#), [READ_ADCF8](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#), [WAIT_EOCF](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG 'Deklaration

EVENT:
  START_CONV(1,1) 'Start AD-Wandlung auf Kanal 1
  WAIT_EOCF(1,1) 'Warten auf Wandlung-Ende
  value = READADCF(1,1) 'Wert vom ADC einlesen
```

START_CONV

SYNC_MODE

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
SYNC_MODE (module, mode)
```

Parameter

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

Bemerkungen

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

Modus „Master / Slave“

Sie dürfen nur ein einziges Master-Modul einstellen. Wenn das Master-Modul eine Wandlung startet – entweder bei einer einzelnen Wandlung (**START_CONV**) oder als Teil einer Burst-Messreihe – sendet es gleichzeitig ein Synchron-Signal.

Sobald Slave-Module das Signal des Master-Moduls empfangen, starten sie auf allen Kanälen eine Wandlung.

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

Modus „Event“

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

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

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

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



Siehe auch

BURST_INIT, BURST_CSTART, BURST_READ, BURST_READ_PACKED, BURST_START, BURST_STATUS, SET_GAIN

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE count 10000
#DEFINE module 1
#DEFINE sampleperiod 40    '1 MHz Messrate [=1/(25ns*40)]

DIM i AS LONG
DIM DATA_1[count], DATA_2[count], DATA_3[count] AS LONG
DIM DATA_4[count], DATA_5[count], DATA_6[count] AS LONG
DIM DATA_7[count], DATA_8[count], DATA_9[count] AS LONG
DIM DATA_10[count], DATA_11[count], DATA_12[count] AS LONG

INIT:
    SYNC_MODE(module,1)      'Master-Modul
    SYNC_MODE(module+1,2)    'Slave-Modul
    SYNC_MODE(module+2,2)    'Slave-Modul
    REM Burst-Messungen für je 4 Kanäle vorbereiten und starten
    BURST_INIT(module,3,sampleperiod,count)
    BURST_INIT(module+1,3,sampleperiod,count)
    BURST_INIT(module+2,3,sampleperiod,count)
    BURST_START(module+1)    'Burst-Messung Slave starten
    BURST_START(module+2)    'Burst-Messung Slave starten
    BURST_START(module)      'Burst-Messung Master starten
    PROCESSDELAY=800         'Mit 50 kHz den Trigger-Punkt finden

EVENT:
    PAR_1=BURST_STATUS(module) 'Anzahl der noch durchzuführenden
                                'Messungen
    IF (PAR_1=0) THEN END      'Burst-Messung beendet - dann FINISH
                                'ausführen

FINISH:
    REM Die jeweils letzten gesammelten Daten aller 4 Kanäle
    REM kopieren
    BURST_READ(module,1,1,count,DATA_1,1)
    BURST_READ(module,2,1,count,DATA_2,1)
    BURST_READ(module,3,1,count,DATA_3,1)
    BURST_READ(module,4,1,count,DATA_4,1)
    BURST_READ(module+1,1,1,count,DATA_5,1)
    BURST_READ(module+1,2,1,count,DATA_6,1)
    BURST_READ(module+1,3,1,count,DATA_7,1)
    BURST_READ(module+1,4,1,count,DATA_8,1)
    BURST_READ(module+2,1,1,count,DATA_9,1)
    BURST_READ(module+2,2,1,count,DATA_10,1)
    BURST_READ(module+2,3,1,count,DATA_11,1)
    BURST_READ(module+2,4,1,count,DATA_12,1)
```

WAIT_EOC

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
WAIT_EOC (module)
```

Parameter

module Eingestellte Moduladresse (1...255).

LONG

Siehe auch

[ADC](#), [ADC16](#), [SET_MUX](#), [START_CONV](#), [READADC](#)

Gültig für Module

(LP)SH-4/8(-FI), AIn-16/14-C Rev. A, AIn-32/12 Rev. A, AIn-32/12 Rev. B, AIn-32/14 Rev. A, AIn-32/16 Rev. B, AIn-32/16 Rev. C, AIn-8/12 Rev. A, AIn-8/12 Rev. B, AIn-8/14 Rev. A, AIn-8/16 Rev. A, AIn-8/16 Rev. B, AIn-8/16 Rev. C, AOut-16/8-12

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value1 AS LONG      'Deklaration

INIT:
  SET_MUX(1,0)      'Multiplexer auf Eingang 1 setzen
  REM An dieser Stelle das Einschwingen des Multiplexers abwarten1
  REM Für 12-Bit ADC: 3µs;
  REM Für AIn-8/16 Rev. B, AIn-32/16 Rev. B: 14µs

EVENT:
  START_CONV(1)      'Start AD-Wandlung
  WAIT_EOC(1)      'Warten auf Wandlung-Ende
  value1 = READADC(1)      'Wert vom ADC einlesen
```

Eine weitere Möglichkeit zur Überbrückung der Einschwingzeit bietet die folgende Warteschleife, welche jedoch nur in hoch priorisierten Prozessen angewendet werden darf:

```
time = READ_TIMER()
DO
  UNTIL (READ_TIMER()-time>120) '25ns*120=3µs warten
```

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

```
UNTIL (READ_TIMER()-time>560) '25ns*560=14µs warten
```

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

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
WAIT_EOCF(module, adc_no)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

adc_no Bitmuster, das die ADC festlegt, auf deren Konvertierungsende gewartet werden soll (siehe Tabelle). LONG

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

Bemerkungen

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

Siehe auch

[ADCF](#), [START_CONV](#), [READADCF](#), [READ_ADCF4](#), [READ_ADCF8](#), [READ_ADCF4_PACKED](#), [READ_ADCF8_PACKED](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
DIM value AS LONG      'Deklaration
```

EVENT:

```
START_CONV(1,1)      'Start AD-Wandlung
WAIT_EOCF(1,1)      'Warten auf das Ende der Konvertierung
value = READADCF(1,1)      'Wert vom ADC einlesen
```

WAIT_EOCF

3.3 Pro I: Ausgangsmodule

Die Include-Datei `<ADWPDA.INC>` beinhaltet alle Funktionen und Prozeduren, die zum Ansprechen der *ADwin-Pro I* D/A-Wandler-Module benötigt werden. Wenn Sie diese Datei mit der *ADbasic*-Anweisung

```
#INCLUDE ADwinPRO_ALL.inc
```

in Ihr *ADbasic*-Programm einbinden, können Sie sämtliche Funktionen und Prozeduren daraus verwenden.

Auf den folgenden Seiten werden alle Funktionen aus der Datei `<ADWPDA.INC>` ausführlich erklärt. Zusätzlich ist zu jeder Funktion ein kleines Anwendungsbeispiel aufgeführt.

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

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



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

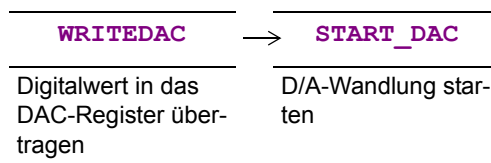
Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DAC(module, dac_no, value)
```

Parameter

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

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



Siehe auch

[START_DAC](#), [WRITEDAC](#)

Gültig für Module

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

Beispiel

```
REM Digitaler P-Regler
#include ADwinPRO_ALL.inc
DIM sp, dev AS LONG 'Deklaration
DIM g, actuate AS LONG 'Deklaration

EVENT:
  sp = PAR_1 'Sollwert
  g = PAR_2 'Verstärkung
  dev = sp - ADC(1,1) 'Regelabweichung berechnen
  actuate = dev * g 'Stellgröße berechnen
  DAC(1,1,actuate) 'Ausgabe der Stellgröße
```

DAC

FG_CONTROL

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

FG_CONTROL(module,output,run_mode)
```

Parameter

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

Bitnr.	31...4	3	2	1	0
DAC-Nr.	–	4	3	2	1

Bemerkungen

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

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

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

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

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

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

Siehe auch

[DAC](#), [FG_DEF](#), [FG_DELAY](#), [FG_MODE](#), [FG_READ_INDEX](#), [FG_STATUS](#), [FG_WRITE](#)

Gültig für Module

AOut-4/16 Rev. C

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM values[100] AS LONG 'Feld mit Ausgabedaten
DIM i AS LONG 'Schleifenvariable

LOWINIT:
  FG_MODE(1, 1) 'Modus Funktionsgenerator aktivieren
  FG_DEF(1, 1, 200, 100) 'DacNo=1, startadr=200, memsize=100
  FOR i=1 TO 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  NEXT i
  FG_WRITE(1, 200, 100, values, 1) 'Startadr=200, length=100,
    'array=values, array_idx=1
  FG_DELAY(1, 1, 80000) 'DacNo = 1, scale=80000 (= 1kHz
    'Ausgaberate); bei 100 Werten beträgt
    'die Periodendauer 100ms

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

EVENT:
  PAR_1=FG_READ_INDEX(1, 1) 'Ausgabeposition des
    'Funktionsgenerators (DAC1) in PAR_1
    'übergeben

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

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

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

FG_DEF

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
FG_DEF(module, dac_no, startadr, memsize)
```

Parameter

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

Bemerkungen

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

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

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

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

Siehe auch

[FG_CONTROL](#), [FG_DELAY](#), [FG_MODE](#), [FG_READ_INDEX](#), [FG_STATUS](#), [FG_WRITE](#)

Gültig für Module

AOut-4/16 Rev. C

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM values[100] AS LONG 'Feld mit Ausgabedaten
DIM i AS LONG           'Schleifenvariable

LOWINIT:
  FG_MODE(1, 1)          'Modus Funktionsgenerator aktivieren
  FG_DEF(1, 1, 200, 100) 'DacNo=1, startadr=200, memsize=100
  FOR i=1 TO 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  NEXT i
  FG_WRITE(1, 200, 100, values, 1) 'Startadr=200, length=100,
                                   'array=values, array_idx=1
  FG_DELAY(1, 1, 80000) 'DacNo = 1, scale=80000 (= 1kHz
                        'Ausgaberate); bei 100 Werten beträgt
                        'die Periodendauer 100ms
```

FG_DELAY

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
FG_DELAY (module, DACno, scale)
```

Parameter

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

Bemerkungen

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

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

Die Anweisung ändert die Ausgaberate sofort.

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

Siehe auch

[FG_CONTROL](#), [FG_DEF](#), [FG_MODE](#), [FG_READ_INDEX](#), [FG_STATUS](#), [FG_WRITE](#)

Gültig für Module

AOut-4/16 Rev. C

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM values[100] AS LONG 'Feld mit Ausgabedaten
DIM i AS LONG 'Schleifenvariable

LOWINIT:
  FG_MODE(1, 1) 'Modus Funktionsgenerator aktivieren
  FG_DEF(1, 1, 200, 100) 'DacNo=1, startadr=200, memsize=100
  FOR i=1 TO 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  NEXT i
  FG_WRITE(1, 200, 100, values, 1) 'Startadr=200, length=100,
    'array=values, array_idx=1
  FG_DELAY(1, 1, 80000) 'DacNo = 1, scale=80000 (= 1kHz
    'Ausgaberate); bei 100 Werten beträgt
    'die Periodendauer 100ms
```



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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
FG_MODE (module, mode)
```

Parameter

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

Bemerkungen

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

- Funktionsgenerator-Modus aktivieren: **FG_MODE** **LOWINIT:**
- Parameter einstellen: **FG_DEF, FG_DELAY, FG_WRITE** **LOWINIT:**
- Funktionsgenerator starten: **FG_CONTROL** **INIT:**
- Abfrage nach Bedarf: **FG_READ_INDEX, FG_STATUS** **INIT:**
EVENT:
FINISH:
- Funktionsgenerator stoppen: **FG_CONTROL** **FINISH:**
- Funktionsgenerator-Modus deaktivieren: **FG_MODE** **FINISH:**

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

Beachten Sie, dass nach jedem Aktivieren (auch ohne vorheriges Deaktivieren) die Parameter mit **FG_DEF**, **FG_DELAY** und **FG_WRITE** erneut einzustellen sind. Die Initialisierung sollte nur ein einziger Prozess durchführen.

Bei inaktivem Funktionsgenerator-Modus können alle Kanäle mit den Anweisungen **DAC**, **WRITEDAC** und **START_DAC** angesprochen werden.

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

Siehe auch

FG_CONTROL, **FG_DEF**, **FG_DELAY**, **FG_READ_INDEX**, **FG_STATUS**, **FG_WRITE**

FG_MODE



Gültig für Module

AOut-4/16 Rev. C

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
DIM values[100] AS LONG 'Feld mit Ausgabedaten
```

```
DIM i AS LONG 'Schleifenvariable
```

LOWINIT:

```
FG_MODE(1, 1)
```

'Modus Funktionsgenerator aktivieren

```
FG_DEF(1, 1, 200, 100)
```

'DacNo=1, startadr=200, memsize=100

```
FOR i=1 TO 100
```

```
values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
```

```
NEXT i
```

```
FG_WRITE(1, 200, 100, values, 1) 'Startadr=200, length=100,
```

'array=values, array_idx=1

```
FG_DELAY(1, 1, 80000)
```

'DacNo = 1, scale=80000 (= 1kHz

'Ausgaberate); bei 100 Werten beträgt

'die Periodendauer 100ms

INIT:

```
FG_CONTROL(1,01b, 0)
```

'nur DAC-Nr. 1 sofort starten

EVENT:

```
PAR_1=FG_READ_INDEX(1, 1) 'Ausgabeposition des
```

'Funktionsgenerators (DAC1) in PAR_1

'übergeben

FINISH:

```
FG_CONTROL(1, 01111b,2)
```

'softstop für alle Kanäle =

'Funktionsgenerator stoppen,

'sobald der letzte Wert des

'Zwischenspeichers ausgegeben wurde.

```
DAC(1, 1, 49152)
```

'Ausgang 1 auf 5V setzen, sobald der

'Funktionsgenerator gestoppt ist

REM Warten, bis alle Funktionsgeneratoren gestoppt sind. Durch
REM diese Warteschleife ist sicher gestellt, dass alle Werte aus
REM den Zwischenspeichern ausgegeben werden, bevor der FG-Modus
REM ausgeschaltet wird.

```
DO
```

```
UNTIL (FG_STATUS(1)=0)
```

```
FG_MODE(1,0)
```

'Modus Funktionsgenerator aus

'= Standard-Modus ein.

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

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = FG_READ_INDEX(module, DACno)
```

Parameter

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

Bemerkungen

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

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

Siehe auch

FG_CONTROL, **FG_DEF**, **FG_DELAY**, **FG_MODE**, **FG_STATUS**, **FG_WRITE**

Gültig für Module

AOut-4/16 Rev. C

FG_READ_INDEX

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM values[100] AS LONG    'Feld mit Ausgabedaten
DIM i AS LONG              'Schleifenvariable

LOWINIT:
  FG_MODE(1, 1)             'Modus Funktionsgenerator aktivieren
  FG_DEF(1, 1, 200, 100)    'DacNo=1, startadr=200, memsize=100
  FOR i=1 TO 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  NEXT i
  FG_WRITE(1, 200, 100, values, 1) 'Startadr=200, length=100,
                                   'array=values, array_idx=1
  FG_DELAY(1, 1, 80000)     'DacNo = 1, scale=80000 (= 1kHz
                           'Ausgaberate); Periodendauer 100ms

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

EVENT:
  PAR_1=FG_READ_INDEX(1, 1) 'Ausgabeposition des
                           'Funktionsgenerators (DAC1) in PAR_1
                           'übergeben
```

FG_STATUS gibt den Status aller Funktionsgeneratoren auf dem angegebenen Modul zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = FG_STATUS(module)
```

Parameter

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

Bitnr.	31...4	3	2	1	0
DAC-Nr.	–	4	3	2	1

Bemerkungen

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

Wenn ein Funktionsgenerator mit **FG_CONTROL** (... , 2) (soft stop) gehalten wird, wird dessen Bit in **ret_val** erst gelöscht, wenn der letzte Wert des Bereichs ausgegeben ist.

Siehe auch

[FG_CONTROL](#), [FG_DEF](#), [FG_DELAY](#), [FG_MODE](#), [FG_READ_INDEX](#), [FG_WRITE](#)

Gültig für Module

AOut-4/16 Rev. C

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

LOWINIT:
    FG_MODE(1, 1)           'Modus Funktionsgenerator aktivieren
    FG_DEF(1, 1, 200, 100)  'DacNo=1, startadr=200, memsize=100
    ...

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

EVENT:
    ...

FINISH:
    ...
    DO
        UNTIL (FG_STATUS(1)=0) 'Warten, bis alle
                                'Funktionsgeneratoren gestoppt sind.
    FG_MODE(1, 0)           'Modus Funktionsgenerator aus
                                '= Standard-Modus ein.
```

FG_STATUS

FG_WRITE

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
FG_WRITE(module, startadr, length, array[], array_idx)
```

Parameter

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

Bemerkungen

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

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

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

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

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

Siehe auch

[FG_CONTROL](#), [FG_DEF](#), [FG_DELAY](#), [FG_MODE](#), [FG_READ_INDEX](#), [FG_STATUS](#)

Gültig für Module

AOut-4/16 Rev. C



Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM values[100] AS LONG    'Feld mit Ausgabedaten
DIM i AS LONG              'Schleifenvariable

LOWINIT:
  FG_MODE(1, 1)            'Modus Funktionsgenerator aktivieren
  FG_DEF(1, 1, 200, 100)   'DacNo=1, startadr=200, memsize=100

  FOR i=1 TO 100
    values[i]=32768+i*327.67 'Sägezahnfunktion berechnen 0-10V
  NEXT i
  FG_WRITE(1, 200, 100, values, 1) 'Startadr=200, length=100,
                                   'array=values, array_idx=1

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

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

EVENT:
  ...
```

START_DAC

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
START_DAC (module)
```

Parameter

module Eingestellte Moduladresse (1...255).

LONG

Siehe auch

WRITEDAC, DAC

Gültig für Module

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

Beispiel

REM Simultane Ausgabe von zwei verschiedenen Signalverläufen
REM auf den Ausgängen 1 und 2 eines D/A-Moduls

```
#INCLUDE ADwinPRO_ALL.inc  
DIM i AS LONG                    'Deklaration  
INIT:  
  i=0  
  
EVENT:  
  WRITEDAC (1,1,i)                'Ausgaberegister DAC1 setzen  
  WRITEDAC (1,2,65535-i)        'Ausgaberegister DAC2 setzen  
  START_DAC (1)                  'Ausgabe auf allen DAC starten  
  INC (i)  
  IF (i=65535) THEN i=0
```


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

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
WRITEDAC (module, dac_no, value)
```

Parameter

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

Siehe auch

[START_DAC](#), [DAC](#)

Gültig für Module

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

Beispiel

REM Simultane Ausgabe von vier verschiedenen Signalverläufen
REM auf den Ausgängen 1, 2, 3 und 4 eines D/A-Moduls
REM Die Signalverläufe sind in vier DATA-Feldern abgelegt und
REM können vor dem Programmstart vom PC übergeben werden

```
#INCLUDE ADwinPRO_ALL.inc
DIM i AS LONG 'Deklaration
DIM DATA_1[1000], DATA_2[1000], DATA_3[1000] AS LONG
DIM DATA_4[1000] AS LONG

INIT:
  i=1

EVENT:
  WRITEDAC (1,1,DATA_1[i]) 'Ausgaberegister DAC1 setzen
  WRITEDAC (1,2,DATA_2[i]) 'Ausgaberegister DAC2 setzen
  WRITEDAC (1,3,DATA_3[i]) 'Ausgaberegister DAC3 setzen
  WRITEDAC (1,4,DATA_4[i]) 'Ausgaberegister DAC4 setzen
  START_DAC (1) 'Ausgabe auf allen DAC starten
  INC (i)
  IF (i>1000) THEN i=1
```

WRITEDAC

3.4 Pro I: Digitalmodule

Die Include-Datei `<ADWPDIO.INC>` beinhaltet alle Funktionen und Prozeduren, die zum Ansprechen der *ADwin-Pro* Digital-I/O-Module benötigt werden. Wenn Sie mit der *ADbasic*-Anweisung

```
#INCLUDE ADwinPRO_ALL.inc
```

die Datei in Ihr *ADbasic*-Programm einbinden, können Sie sämtliche Funktionen und Prozeduren daraus verwenden.

Auf den folgenden Seiten werden die Befehle ausführlich erklärt. Zusätzlich ist zu jeder Funktion ein kleines Anwendungsbeispiel aufgeführt.

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

In den Anwendungsbeispielen wird davon ausgegangen, dass auf dem Digital-Modul die Adresse 1 eingestellt ist



CNT_CLEAR setzt den Zählerstand eines oder mehrerer Zähler auf dem angegebenen Modul auf den Wert 0 (Null).

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
CNT_CLEAR(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

pattern Bitmuster, Zuordnung zu Zählern siehe Tabelle. LONG
 Bit = 0: keine Funktion.
 Bit = 1: Zähler zurücksetzen.

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

Gültig für Module

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

Beispiel; in dieser Form nur für das Modul Pro-CNT-VR4 gültig!

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
CNT_SETMODE(1,01111b)      'Zähler 1...4 auf
                             'Takt/Richtungsauswertung setzen
CNT_CLEAR(1,01111b)        'Zählerstand Zähler 1...4 auf 0 setzen
CNT_ENABLE(1,01111b)      'Zähler 1...4 aktivieren
```

CNT_CLEAR

CNT_ENABLE

CNT_ENABLE aktiviert oder deaktiviert einen oder mehrere Zähler auf dem angegebenen Modul.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
CNT_ENABLE(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

pattern Zähleraktivierung nach Bitmuster, Zuordnung zu Zählern siehe Tabelle. LONG
Bit = 0: Zähler deaktivieren / sperren.
Bit = 1: Zähler aktivieren / freigeben.

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

Gültig für Module

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

Beispiel

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

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
CNT_SETMODE(1,01000b)      'Zähler 4 auf Takt/Richtungs-  
                             'auswertung, alle anderen Zähler auf  
                             'Vierfachflankenauswertung setzen  
CNT_CLEAR(1,01000b)        'Zählerstand Zähler 4 auf 0 setzen  
CNT_ENABLE(1,01000b)       'Zähler 4 aktivieren, alle anderen  
                             'deaktivieren
```



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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
CNT_LATCH(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

pattern Zählerstand "latchen" gemäß Bitmuster, Zuordnung zu Zählern siehe Tabelle. LONG

Bit = 0: keine Funktion.
Bit = 1: Zählerstand ins Zwischenregister übernehmen.

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

Siehe auch

[CNT_READLATCH16](#), [CNT_READLATCH32](#)

Gültig für Module

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

Beispiel

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

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
value = ADC(1,1)           'Messwert aufnehmen
IF (value>49151) THEN
CNT_LATCH(1,00011b)       'Zähler 1 u. 2 latchen
ENDIF
REM Differenz bilden
PAR_1 = CNT_READLATCH32(1,00001b) - CNT_READLATCH32(1,00010b)
```



CNT_LATCH

CNT_READ16

CNT_READ16 gibt den aktuellen Zählerstand eines 16 Bit-Zählers auf dem angegebenen Modul zurück.

Syntax

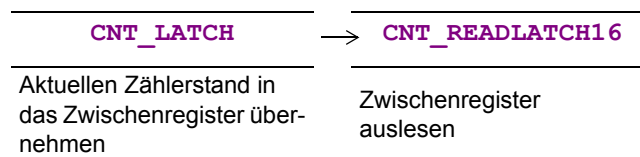
```
#INCLUDE ADwinPRO_ALL.inc
ret_val = CNT_READ16(module, cnt_no)
```

Parameter

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

Beispiel

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



Für spezielle Anwendungen können Sie auch diese Befehle anstelle von **CNT_READ16** verwenden.

Siehe auch

[CNT_READ32](#), [CNT_LATCH](#), [CNT_READLATCH16](#)

Gültig für Module

CNT-16/16(-I)

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
EVENT:
  PAR_1 = CNT_READ16(1,1) 'Aktuellen Zählerstand von
                          'Zähler 1 ermitteln
  PAR_2 = CNT_READ16(1,2) 'Aktuellen Zählerstand von
                          'Zähler 2 ermitteln
```

CNT_READ32 gibt den aktuellen Zählerstand eines 32 Bit-Zählers auf dem angegebenen Modul zurück.

Syntax

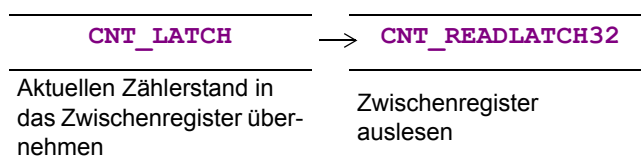
```
#INCLUDE ADwinPRO_ALL.inc
ret_val = CNT_READ32(module, cnt_no)
```

Parameter

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

Bemerkungen

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



Für spezielle Anwendungen können Sie auch diese Befehle anstelle von **CNT_READ32** verwenden.

Siehe auch

[CNT_READ16](#), [CNT_LATCH](#), [CNT_READLATCH32](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
EVENT:
  PAR_1 = CNT_READ32(1,3) 'Aktuellen Zählerstand von
                          'Zähler 3 ermitteln
  PAR_2 = CNT_READ32(1,4) 'Aktuellen Zählerstand von
                          'Zähler 4 ermitteln
```

CNT_READ32

CNT_READLATCH16

CNT_READLATCH16 gibt den Wert aus dem Zwischenregister eines 16 Bit-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = CNT_READLATCH16(module, cnt_no)
```

Parameter

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

Bemerkungen

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

Siehe auch

[CNT_LATCH](#), [CNT_READ16](#)

Gültig für Module

CNT-16/16(-I)

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM value AS LONG  
  
INIT:  
value = ADC(1,1) 'Messwert aufnehmen  
IF (value>49151) THEN CNT_LATCH(1,3) 'Zähler 1 u. 2 latchen  
PAR_1 = CNT_READLATCH16(1,1) - CNT_READLATCH16(1,2) 'Differenz Zähler 1 u. 2
```


CNT_READLATCH32 gibt den Wert aus dem Zwischenregister eines 32 Bit-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = CNT_READLATCH32(module, cnt_no)
```

Parameter

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

Bemerkungen

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

Siehe auch

[CNT_LATCH](#), [CNT_READ32](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
value = ADC(1,1)           'Messwert aufnehmen
IF (value>49151) THEN CNT_LATCH(1,3)
                           'Zähler 1 u. 2 latchen
PAR_1 = CNT_READLATCH32(1,1) - CNT_READLATCH32(1,2)
                           'Differenz Zähler 1 u. 2
```

CNT_READLATCH32

CNT_SETMODE

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

CNT_SETMODE(module,pattern)
```

Parameter

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

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

Bemerkungen

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

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

CNT_SETMODE (1,1100b)	'Zähler 3 und 4 auf Takt-/Richtungs- 'auswertung, alle anderen Zähler auf 'Vierfach-Flankenauswertung
CNT_CLEAR (1,1100b)	'Zählerstand der Zähler 3 und 4 auf '0 (Null) setzen
CNT_ENABLE (1,1100b)	'Zähler 3 und 4 aktivieren, 'alle anderen deaktivieren

CO4_CLEARENABLE schaltet den externen Eingang CLR eines oder mehrerer Zähler auf dem angegebenen Modul frei.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
CO4_CLEARENABLE(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

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

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

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

Bemerkungen

Der externe Eingang darf entweder mit der Anweisung **CO4_CLEAR-ENABLE** oder mit **CO4_LATCHENABLE** freigeschaltet sein, nicht aber mit beiden Anweisungen gleichzeitig!

Siehe auch

[CO4_LATCHENABLE](#), [CO4_SETMODE](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
CO4_SETMODE(1,1,1)      'Zähler 1: CLK/DIR-Modus
CO4_LATCHENABLE(1,0)    'Eingang Latch für alle Zähler sperren
CO4_CLEARENABLE(1,1)    'Eingang Clear für Zähler 1 freigeben
CNT_CLEAR(1,1)          'Zähler 1 löschen
CNT_ENABLE(1,1)         'Zähler 1 starten

EVENT:
PAR_1 = CO4_READ(1,1)    'Zähler 1 auslesen
```

CO4_CLEARENABLE

CO4_GETSTATUS

CO4_GETSTATUS gibt den Status der Eingangssignale eines Zählers auf dem angegebenen Modul in einem Bitmuster zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = CO4_GETSTATUS(module, cnt_no)
```

Parameter

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

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

Fehler (mehrmals lesbare Statusmeldung): Verwenden Sie zum Löschen der Meldung den Befehl **CO4_RESETSTATUS**.

Bit 0 = 1: Kabelfehler; die differentiellen Signale (A & /A oder B & /B oder C & /C; C = CLR-/LATCH) weisen einen der folgenden Fehler auf:

Offene Leitung (Kabelbruch), Kurzschluss, zu kleine Signalspannung, zu hohe Gleichtaktspannung oder zu kleine Slew-Rate (< 0,33 V/μs).

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

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

Bit 2 = 1: CLR-Signal liegt an (wenn es durch **CO4_CLEARENABLE** freigeschaltet wurde).

Bit 3 = 1: LATCH-Signal liegt an (wenn es durch **CO4_LATCHENABLE** freigeschaltet wurde).

Bemerkungen

Ein Kabelfehler (Bit 0) kann nur bei differentiellen Eingängen erkannt werden! Bei TTL-Eingängen sind diese Bits stets 0 (Null) und der Befehl **CO4_RESETSTATUS** hat keine Wirkung.

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

Siehe auch

[CO4_RESETSTATUS](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM error AS LONG

INIT:
  CO4_SETMODE(1,1,0)      'Zähler 1 Vierflanken-Modus
  CNT_CLEAR(1,1)          'Zähler 1 löschen
  CNT_ENABLE(1,1)         'Zähler 1 starten
  CO4_RESETSTATUS(1,1)    'Status-Register löschen
  error = 0               'Fehlerindikator zurücksetzen

EVENT:
  PAR_1 = CO4_READ(1,1)   'Zähler 1 auslesen
  PAR_2 = CO4_GETSTATUS(1,1) 'Zähler 1 auslesen
  IF (PAR_2 AND 1 = 1) THEN 'Kabelfehler?
    INC PAR_3              'Anzahl Kabelfehler bisher
    error = 1              'Fehlerindikator setzen
  ENDIF
  IF (PAR_2 AND 2 = 2) THEN 'Korrelationsfehler?
    INC PAR_4              'Anzahl Korrelationsfehler bisher
    error = 1              'Fehlerindikator setzen
  ENDIF
  PAR_5 = SHIFT_RIGHT(PAR_2 AND 16,4) 'akt. Zustand CLR-Eingang
  PAR_6 = SHIFT_RIGHT(PAR_2 AND 32,5) 'akt. Zustand A-Eingang
  PAR_7 = SHIFT_RIGHT(PAR_2 AND 64,6) 'akt. Zustand B-Eingang
  IF (error = 1) THEN      'Fehlerindikator gesetzt?
    CO4_RESETSTATUS(1,1)   'Status-Register zurücksetzen
    error = 0              'Fehlerindikator zurücksetzen
  ENDIF
```

CO4_LATCHENABLE

CO4_LATCHENABLE schaltet den externen Eingang LATCH eines oder mehrerer Zähler auf dem angegebenen Modul frei.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

CO4_LATCHENABLE(module,pattern)
```

Parameter

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

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

Bemerkungen

Der externe Eingang darf entweder mit der Anweisung **CO4_CLEAR-ENABLE** oder mit **CO4_LATCHENABLE** freigeschaltet sein, nicht aber mit beiden Anweisungen gleichzeitig!

Siehe auch

[CO4_CLEARENABLE](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
CO4_SETMODE(1,1,1)      'Zähler 1: CLK/DIR-Modus
CO4_CLEARENABLE(1,0)    'Eingang Clear für alle Zähler sperren
CO4_LATCHENABLE(1,1)    'Eingang Latch (Zähler 1) freigeben
CNT_CLEAR(1,1)          'Zähler 1 löschen
CNT_ENABLE(1,1)         'Zähler 1 starten

EVENT:
PAR_1 = CO4_READLATCH(1,1) 'Zähler 1 auslesen
```

CO4_READ gibt den aktuellen Zählerstand eines Zählers des angegebenen Moduls zurück.

Syntax

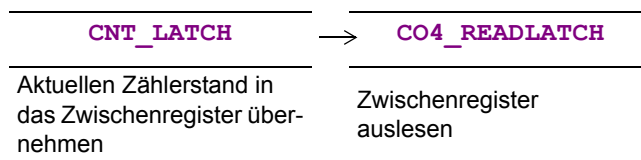
```
#INCLUDE ADwinPRO_ALL.inc
ret_val = CO4_READ(module, cnt_no)
```

Parameter

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

Bemerkungen

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



Für spezielle Anwendungen können Sie diese Befehle auch anstelle von **CO4_READ** verwenden.

Siehe auch

[CNT_LATCH](#), [CO4_READLATCH](#)

Gültig für Module

CNT-16/32(-I), CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
CO4_SETMODE(1,1,1)      'Zähler 1: CLK/DIR-Modus
CNT_CLEAR(1,1)          'Zähler 1 löschen
CNT_ENABLE(1,1)         'Zähler 1 starten
```

EVENT:

```
PAR_1 = CO4_READ(1,1)   'Aktuellen Zählerstand von Zähler 1
                        'ermitteln
```

CO4_READ

CO4_READ- LATCH

CO4_READLATCH gibt den Wert aus dem Zwischenregister (Latch) eines Zählers des angegebenen Moduls zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = CO4_READLATCH(module, cnt_no)
```

Parameter

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

Bemerkungen

Um den aktuellen Zählerstand zu erhalten, muss dieser zuerst mit dem Befehl **CNT_LATCH** in das Zwischenregister übernommen werden und kann dann mit **CO4_READLATCH** gelesen werden.

Siehe auch

[CNT_LATCH](#), [CO4_READ](#)

Gültig für Module

CNT-16/32(-I), CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM old, new AS LONG      'Variablen dimensionieren

INIT:
  old = 0                  'Variable initialisieren
  CO4_SETMODE(1,1,1)      'Zähler 1: CLK/DIR-Modus
  CNT_CLEAR(1,1)          'Zähler 1 zurücksetzen
  CNT_ENABLE(1,1)         'Zähler 1 starten

EVENT:
  CNT_LATCH(1,1)           'Zähler 1 latches und..
  new = CO4_READLATCH(1,1) 'Latch auslesen
  PAR_1 = new - old        'Differenz bilden (f = Impulse / Zeit)
  old = new                'Neuen Zählerstand als alten
                           'speichern
```


CO4_RESETSTATUS löscht das Statusregister eines oder mehrerer Zähler auf dem angegebenen Modul.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
CO4_RESETSTATUS (module, pattern)
```

Parameter

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

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

Bemerkungen

Das Statusregister enthält den Status der Eingangssignale eines Zählers (Auslesen mit **CO4_GETSTATUS**).

Siehe auch

[CO4_GETSTATUS](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM error AS LONG

INIT:
CO4_SETMODE (1,1,0)      'Zähler 1: Vierflanken-Modus
CNT_CLEAR (1,1)          'Zähler 1 löschen
CNT_ENABLE (1,1)         'Zähler 1 starten
CO4_RESETSTATUS (1,1)    'Status-Register löschen
error = 0                'Fehlerindikator zurücksetzen

EVENT:
PAR_1 = CO4_READ (1,1)   'Zähler 1 auslesen
PAR_2 = CO4_GETSTATUS (1,1) 'Eingangs-Status Zähler 1 auslesen
IF (PAR_2 AND 1 = 1) THEN 'Kabelfehler?
    INC PAR_3             'Anzahl Kabelfehler bisher
    error = 1             'Fehlerindikator setzen
ENDIF
IF (PAR_2 AND 2 = 2) THEN 'Korrelationsfehler?
    INC PAR_4             'Anzahl Korrelationsfehler bisher
    error = 1             'Fehlerindikator setzen
ENDIF
PAR_5 = SHIFT_RIGHT (PAR_2 AND 16,4) 'akt. Zustand CLR-Eingang
PAR_6 = SHIFT_RIGHT (PAR_2 AND 32,5) 'akt. Zustand A-Eingang
PAR_7 = SHIFT_RIGHT (PAR_2 AND 64,6) 'akt. Zustand B-Eingang
IF (error = 1) THEN      'Fehlerindikator gesetzt?
    CO4_RESETSTATUS (1,1) 'Status-Register zurücksetzen
    error = 0            'Fehlerindikator zurücksetzen
ENDIF
```

CO4_RESETSTATUS

CO4_SET_LATCHMODE

CO4_SET_LATCHMODE bestimmt den Modus der Latch-Eingänge für alle Zähler des angegebenen Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
CO4_SET_LATCHMODE(module, pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

pattern Bitmuster für den Latch-Modus der Zähler (siehe LONG Tabellen); voreingestellt ist 00000000b.

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

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

Bemerkungen

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

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

Der Befehl **CO4_SET_LATCHMODE** legt fest,

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

Siehe auch

[CNT_LATCH](#)

Gültig für Module

CO4



CO4_SETMODE stellt den Zählmodus eines Zählers auf dem angegebenen Modul ein.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
CO4_SETMODE(module, cnt_no, mode)
```

Parameter

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

Bemerkungen

Die Zähler können in 3 verschiedenen Modi betrieben werden:

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

Siehe auch

[CNT_CLEAR](#), [CNT_ENABLE](#)

Gültig für Module

CO4

CO4_SETMODE

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE refCLK 40E6
DIM rise, rise_old, fall, fall_old, T AS LONG

INIT:
    rise_old = 0
    fall_old = 0
    CO4_SETMODE(1,1,2)      'Zähler 1: PWM-Analyse
    CNT_CLEAR(1,1)          'Zähler 1 löschen
    CNT_ENABLE(1,1)         'Zähler 1 starten

EVENT:
    rise = CO4_READLATCH(1,1) 'Latch 1 auslesen
                                '(pos. Flanke, Zähler 1)
    fall = CO4_READLATCH(1,5) 'Latch 5 auslesen
                                '(neg. Flanke, Zähler 1)
    IF (rise <> rise_old) THEN 'Pos. Flanke detektiert?
        IF (fall <> fall_old) THEN 'Neg. Flanke detektiert,
                                    'd.h. PWM-Signal = low?
            PAR_1 = fall - rise    'Impulsdauer in Anzahl Perioden des
                                    'Ref.-Taktes
            PAR_2 = rise - fall_old 'Pausendauer in Anzahl Perioden des
                                    'Ref.-Taktes
        ELSE
            'Keine neg. Flanke detektiert,
            'd.h. PWM-Signal = HIGH?
            PAR_1 = fall - rise_old 'Impulsdauer in Anzahl Perioden des
                                    'Ref.-Taktes
            PAR_2 = rise - fall    'Pausendauer in Anzahl Perioden des
                                    'Ref.-Taktes
        ENDIF
    ENDIF
    T = PAR_1 + PAR_2            'Periodendauer in Anzahl Perioden des
                                    'Ref.-Taktes
    FPAR_1 = refCLK / T          'Frequenz des PWM-Signals
    FPAR_2 = PAR_1 * 100 / T    'Tastverhältnis in Prozent
    rise_old = rise             'Latch-Inhalt speichern
    fall_old = fall             'Latch-Inhalt speichern
```

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
DIG_LATCH(module)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

Bemerkungen

Bemerkungen

Wir empfehlen bei den Modulen Pro-DIO-32 und Pro-DIO-32 Rev. B, die Leitungen zunächst mit den Anweisungen **DIGPROG1** und **DIGPROG2** als Eingänge oder Ausgänge zu programmieren.

Abhängig vom Modul überträgt die Anweisung folgende Informationen:

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

Wenn das angegebene Modul durch **SYNCENABLE** zur Synchronisation freigeschaltet ist, hat der Befehl **SYNCALL** die gleiche Funktion wie der Befehl **DIG_LATCH**.

Siehe auch

[DIG_READLATCH1](#), [DIG_READLATCH2](#), [DIG_WRITELATCH1](#), [DIG_WRITELATCH2](#), [DIG_WRITELATCH32](#), [EXTLCH_ENABLE](#)
[DIGPROG1](#), [DIGPROG2](#), [DIGIN_WORD1](#), [DIGIN_WORD2](#), [DIGOUT](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#)
[DIGIN_LONG_F](#), [DIGOUT_BITS_F](#), [DIGOUT_F](#), [DIGOUT_LONG_F](#)
[GET_DIGOUT_LONG](#),
[GET_DIGOUT_WORD1](#), [GET_DIGOUT_WORD2](#)
[SYNCALL](#), [SYNCENABLE](#)

Gültig für Module

DIO-32, DIO-32 Rev. B, OPT-16 Rev. A, REL-16, TRA-16 Rev. A, TRA-16 Rev. B

DIG_LATCH

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
DIGPROG1(1,0FFFFh)      'DIO15:00 (Low-Word) des DIO-32-  
                          'Moduls als Ausgang  
DIGPROG2(1,0)           'DIO31:16 (High-Word) des DIO-32-  
                          'Moduls als Eingang  
DIG_WRITELATCH1(1,0)    'Alle Ausgangs-Bits auf 0 setzen
```

EVENT:

```
DIG_LATCH(1)            'Eingänge latchen, Inhalt des  
                          'Ausgangs-Latches ausgeben  
...                     'weitere Programmschritte  
PAR_1 = DIG_READLATCH2(1) 'High-Word einlesen und beim...  
DIG_WRITELATCH1(1,PAR_1) 'nächsten Event im Low-Word ausgeben
```

DIG_READLATCH1 liefert die unteren 16 Bit (Bit 0...Bit 15) aus dem Zwischenregister für die digitalen Eingänge des angegebenen Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIG_READLATCH1 (module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
---------------------	--------------------------------------	------

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **DIGPROG1** als Eingänge zu programmieren.

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

- **DIGIN_WORD1**
- **DIGIN_WORD2**
- **SYNCALL** (falls für das Modul aktiviert).

Siehe auch

[DIG_LATCH](#), [DIG_READLATCH2](#), [DIG_WRITELATCH1](#), [DIG_WRITELATCH2](#), [DIG_WRITELATCH32](#), [EXTLCH_ENABLE](#)
[DIGPROG1](#), [DIGPROG2](#), [DIGIN_WORD1](#), [DIGIN_WORD2](#), [DIGOUT](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#)
[DIGIN_LONG_F](#)
[SYNCALL](#), [SYNCENABLE](#)

Gültig für Module

DIO-32, DIO-32 Rev. B, OPT-16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
    REM DIO15:00 der Module 1+2 als Eingänge setzen
    DIGPROG1 (1,0)
    DIGPROG1 (2,0)
    SYNCENABLE (1,dio,1)      'Synchron. Dig.-Modul 1 freigeben
    SYNCENABLE (2,dio,1)      'Synchron. Dig.-Modul 2 freigeben

EVENT:
    SYNCALL ()                'Pegel an den digitalen Eingängen von
                              'beiden Modulen synchron in die
                              'Zwischenregister übernehmen

    PAR_1 = DIG_READLATCH1 (1) 'Zwischenregister Modul 1 auslesen
                              '(Bits 0...15)
    PAR_2 = DIG_READLATCH1 (2) 'Zwischenregister Modul 2 auslesen
                              '(Bits 0...15)
```

DIG_READLATCH1

DIG_READLATCH2

DIG_READLATCH2 liefert die oberen 16 Bit (Bit 16...Bit 31) aus dem Zwischenregister für die digitalen Eingänge des angegebenen Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIG_READLATCH2(module)
```

Parameter

module Eingestellte Moduladresse (1...255).

LONG

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **DIGPROG2** als Eingänge zu programmieren.

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

- **DIGIN_WORD1**
- **DIGIN_WORD2**
- **SYNCALL** (falls für das Modul aktiviert).

Siehe auch

DIG_LATCH, DIG_READLATCH1, DIG_WRITELATCH1, DIG_WRITELATCH2, DIG_WRITELATCH32, EXTLCH_ENABLE

DIGPROG1, DIGPROG2, DIGIN_LONG_F, DIGIN_WORD1, DIGIN_WORD2

SYNCALL, SYNCENABLE

Gültig für Module

DIO-32, DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG
```

INIT:

```
REM DIO31:16 der Module 1+2 als Eingänge setzen
DIGPROG2(1,0)
DIGPROG2(2,0)
SYNCENABLE(1,dio,1)      'Synchr. Digital-Modul 1 freigeben
SYNCENABLE(2,dio,1)      'Synchr. Digital-Modul 2 freigeben
```

EVENT:

```
SYNCALL()                'Pegel an den digitalen Eingängen von
                          'beiden Modulen synchron in die
                          'Zwischenregister übernehmen
PAR_1 = DIG_READLATCH2(1) 'Zwischenregister Modul 1 auslesen
                          '(Bits 16...31)
PAR_2 = DIG_READLATCH2(2) 'Zwischenregister Modul 2 auslesen
                          '(Bits 16...31)
```


DIG_Writelatch1 schreibt einen Wert in die unteren 16 Bit (Bit 0...15) des Zwischenregisters für die digitalen Ausgänge des angegebenen Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIG_Writelatch1(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

pattern Bitmuster. Jedes Bit entspricht einem digitalen Ausgang (siehe Tabelle). LONG

Bitnr.	31:16	15	14	13	...	2	1	0
Ausgang	–	15	14	13	...	2	1	0

Bemerkungen

Bei den Modulen Pro-DIO-32 und Pro-DIO-32 Rev. B müssen die angesprochenen Leitungen zunächst mit den Anweisungen **DIGPROG1** als Ausgänge programmiert werden.

Sie können den Wert des Zwischenregisters für die digitalen Ausgänge mit folgenden Anweisungen setzen:

- **DIGOUT**
- **DIGOUT_WORD1**
- **DIGOUT_WORD2**
- **DIG_Writelatch1**
- **DIG_Writelatch2**
- **DIG_Writelatch32**

Diese Anweisung darf nicht in Kombination mit **DIG_Writelatch2** verwendet werden. Falls Sie Bits sowohl im unteren als auch im oberen Wort des Zwischenregisters verändern wollen, benutzen Sie bitte den Befehl **DIG_Writelatch32**.

Siehe auch

DIG_Latch, **DIG_Readlatch1**, **DIG_Readlatch2**, **DIG_Writelatch2**, **DIG_Writelatch32**, **EXTLCH_Enable**
DIGPROG1, **DIGPROG2**, **DIGOUT**, **DIGOUT_WORD1**, **DIGOUT_WORD2**
GET_DIGOUT_WORD1, **GET_DIGOUT_WORD2**

Gültig für Module

DIO-32, DIO-32 Rev. B, REL-16, TRA-16 Rev. A, TRA-16 Rev. B

DIG_Writelatch1



Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
    REM Nur DIO-32: DIO15:00 der Module als Ausgänge setzen
    DIGPROG1(1,0FFFFh)
    DIGPROG1(2,0FFFFh)

    SYNCENABLE(1,dio,1)      'Synchr. Digital-Modul 1 freigeben
    SYNCENABLE(2,dio,1)      'Synchr. Digital-Modul 2 freigeben
    DIG_WRITELATCH1(1,1)     'Unterstes Bit im Ausgangs-
                             'Zwischenregister setzen
    DIG_WRITELATCH1(2,1)     'Unterstes Bit im Ausgangs-
                             'Zwischenregister setzen

EVENT:
    value = ADC(1,1)          'Messwerterfassung
    IF (value > 3000) THEN    'Grenzwert überschritten?
        SYNCALL()            'Werte aus den Zwischenregistern
                              'ausgeben
    ENDIF
```

DIG_Writelatch2 schreibt einen Wert in die oberen 16 Bit (Bit 16...31) des Zwischenregisters für die digitalen Ausgänge des angegebenen Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIG_Writelatch2(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

pattern Bitmuster. Jedes Bit entspricht einem digitalen Ausgang (siehe Tabelle). LONG

Bitnr.	31:16	15	14	13	...	2	1	0
Ausgang	–	31	30	29	...	18	17	16

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **DIGPROG2** als Ausgänge programmiert werden.

Sie können den Wert des Zwischenregisters für die digitalen Ausgänge mit folgenden Anweisungen setzen:

- **DIGOUT**
- **DIGOUT_WORD1**
- **DIGOUT_WORD2**
- **DIG_Writelatch1**
- **DIG_Writelatch2**
- **DIG_Writelatch32**

Diese Anweisung darf nicht in Kombination mit **DIG_Writelatch1** verwendet werden. Falls Sie Bits sowohl im unteren als auch im oberen Wort des Zwischenregisters verändern wollen, benutzen Sie bitte den Befehl **DIG_Writelatch32**.

Siehe auch

DIG_Latch, **DIG_Readlatch1**, **DIG_Readlatch2**, **DIG_Writelatch1**, **DIG_Writelatch32**, **EXTLCH_ENABLE**

DIGPROG1, **DIGPROG2**, **DIGOUT**, **DIGOUT_WORD1**, **DIGOUT_WORD2**

GET_DIGOUT_WORD1, **GET_DIGOUT_WORD2**

Gültig für Module

DIO-32, DIO-32 Rev. B

DIG_Writelatch2



Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
    REM Nur DIO-32: DIO31:16 der Module als Ausgänge setzen
    DIGPROG2(1,0FFFFh)
    DIGPROG2(2,0FFFFh)

    SYNCENABLE(1,dio,1)      'Synchr. Digital-Modul 1 freigeben
    SYNCENABLE(2,dio,1)      'Synchr. Digital-Modul 2 freigeben
    DIG_WRITELATCH2(1,1)     'Bit 16 im Ausgangs-Zwischenregister
                              'setzen
    DIG_WRITELATCH2(2,16)    'Bit 20 im Ausgangs-Zwischenregister
                              'setzen

EVENT:
    value = ADC(1,1)          'Messwerterfassung
    IF (value > 3000) THEN    'Grenzwert überschritten?
        SYNCALL()           'Werte aus den Zwischenregistern
                              'ausgeben
    ENDIF
```

DIG_Writelatch32 schreibt einen 32 Bit-Wert in das Langwort (Bits 31...00) des Latches auf dem angegebenen Modul.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIG_Writelatch32 (module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
pattern	Bitmuster. Jedes Bit entspricht einem digitalen Ausgang (siehe Tabelle).	LONG

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

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit den Anweisungen **DIGPROG1** und **DIGPROG2** als Ausgänge programmiert werden.

Sie können den Wert des Zwischenregisters für die digitalen Ausgänge mit folgenden Anweisungen setzen:

- **DIGOUT**
- **DIGOUT_WORD1**
- **DIGOUT_WORD2**
- **DIG_Writelatch1**
- **DIG_Writelatch2**
- **DIG_Writelatch32**

Siehe auch

DIG_LATCH, **DIG_READLATCH1**, **DIG_READLATCH2**, **DIG_Writelatch1**, **DIG_Writelatch2**, **EXTLCH_ENABLE**

DIGPROG1, **DIGPROG2**, **DIGOUT**, **DIGOUT_WORD1**, **DIGOUT_WORD2**

GET_DIGOUT_WORD1, **GET_DIGOUT_WORD2**

Gültig für Module

DIO-32, DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    DIGPROG1 (1, 0FFFFh)      'DIO15:00 (Low-Word) des DIO-32-
                              'Moduls als Ausgang
    DIGPROG2 (1, 0FFFFh)      'DIO31:16 (High-Word) des DIO-32-
                              'Moduls als Ausgang

EVENT:
    DIG_LATCH (1)             'Informationen des Ausgangs-Latches
                              'auf einem DIO-32-Modul ausgeben
    DIG_Writelatch32 (1, PAR_1) 'Long-Word ins Ausgangs-Latch
                              'schreiben
```

DIG_Writelatch32

DIGIN_LONG_F

DIGIN_LONG_F gibt den Zustand der Eingänge (Bits 31...00) des angegebenen Moduls als Bitmuster zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = DIGIN_LONG_F(module)
```

Parameter

module	Eingestellte Moduladresse (1...255). LONG
ret_val	Bitmuster. Jedes Bit (31...0) entspricht dem Eingangszustand eines digitalen Eingangs (siehe Tabelle). LONG Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an.

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

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit den Anweisungen **DIGPROG1** und **DIGPROG2** als Eingänge zu programmieren.

Wenn ein oder mehrere der Kanäle 0...31 als Ausgang programmiert sind, sind die zugehörigen Bits nicht definiert.

Siehe auch

[DIG_LATCH](#), [DIGIN_WORD1](#), [DIGIN_WORD2](#), [DIGPROG1](#), [DIGPROG2](#), [DIGOUT_LONG_F](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#)

Gültig für Module

DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    DIGPROG1(1,0)          'DIO15:00 (Low-Word) als Eingang
    DIGPROG2(1,0)          'DIO31:16 (High-Word) als Eingang

EVENT:
    PAR_1 = DIGIN_LONG_F(1) 'Alle Eingänge einlesen
```

DIGIN_WORD1 gibt den Zustand der Eingänge 0...15 des angegebenen Moduls als Bitmuster zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = DIGIN_WORD1(module)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
ret_val	Bitmuster. Jedes der Bits 15...0 entspricht dem Eingangszustand eines digitalen Eingangs (siehe Tabelle). Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an.	LONG

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

Bemerkungen

Wir empfehlen, bei den Modulen Pro-DIO-32 und Pro-DIO-32 Rev. B die angesprochenen Leitungen zunächst mit der Anweisung **DIGPROG1** als Eingänge zu programmieren.

Wenn ein oder mehrere der Kanäle 0...15 als Ausgang programmiert sind, sind die zugehörigen Bits nicht definiert.

Siehe auch

[DIGIN_WORD2](#), [DIGOUT](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#), [DIGPROG1](#), [DIGPROG2](#)

Gültig für Module

DIO-32, DIO-32 Rev. B, OPT-16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM DATA_1[10000] AS LONG AS FIFO

INIT:
  REM nur für DIO32: DIO15:00 (Low-Word) als Eingang setzen
  DIGPROG1(4,0)

EVENT:
  REM Abfrage, ob Eingänge 1, 2 und 3 des Moduls 4 gesetzt sind
  IF (DIGIN_WORD1(4) AND 14 = 14) THEN
    DATA_1 = ADC(1,1)      'Messwerterfassung
  ENDIF
```

Am Bitmuster des Dezimalwerts 14 (nämlich ...01110b) können Sie erkennen, welche digitalen Eingänge gesetzt sind, hier die Eingänge 1, 2 und 3.

Sie können in *ADbasic* Zahlen auch in binärer Schreibweise eingeben. Fügen Sie hierzu an das Bitmuster den Buchstaben b an. Die Zeile im obigen Beispiel würde dann lauten:

```
IF (DIGIN_WORD1(4) AND 1110b = 1110b) THEN
```

DIGIN_WORD1

DIGIN_WORD2

DIGIN_WORD2 gibt den Zustand der Eingänge 16...31 des angegebenen Moduls als Bitmuster zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = DIGIN_WORD2(module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>ret_val</code>	Bitmuster. Jedes der Bits 15...0 entspricht dem Eingangszustand eines digitalen Eingangs (siehe Tabelle). Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an.	LONG

Bitnr.	31:16	15	14	...	2	1	0
Eingang	–	31	30	...	18	17	16

Bemerkungen

Wir empfehlen, die angesprochenen Leitungen zunächst mit der Anweisung **DIGPROG2** als Eingänge zu programmieren.

Wenn ein oder mehrere der Kanäle 16...31 als Ausgang programmiert sind, sind die zugehörigen Bits nicht definiert.

Siehe auch

[DIGIN_WORD1](#), [DIGOUT](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#), [DIGPROG1](#), [DIGPROG2](#)

Gültig für Module

DIO-32, DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM DATA_1[10000] AS LONG AS FIFO

INIT:
    DIGPROG2(4,0)                'DIO31:16 (High-Word) als Eingang

EVENT:
    REM Abfrage, ob Eingänge 16, 18 und 21 auf Modul 4 gesetzt sind
    IF (DIGIN_WORD2(4) AND 39 = 39) THEN
        DATA_1 = ADC(1,1)        'Messwerterfassung
    ENDIF
```

Am Bitmuster des Dezimalwerts 39 (nämlich ...0100101b) können Sie erkennen, welche digitalen Eingänge gesetzt sind, hier die Eingänge 16, 18 und 21.

Sie können in *ADbasic* Zahlen auch in binärer Schreibweise eingeben. Fügen Sie hierzu an das Bitmuster den Buchstaben b an. Die Zeile im obigen Beispiel würde dann lauten:

```
IF (DIGIN_WORD2(4) AND 0100101b = 0100101b) THEN
```


DIGOUT setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIGOUT (module, output, value)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
output	Nummer des Ausgangs, der angesprochen werden soll (0...31).	LONG
value	Neuer Zustand für den gewählten Ausgang: 0: Pegel Low. 1: Pegel High.	LONG

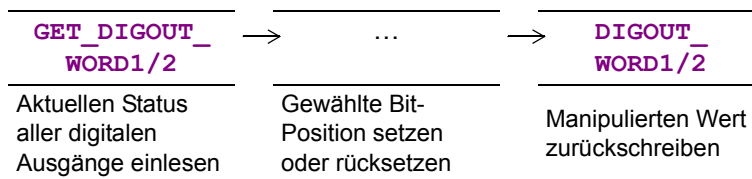
Bemerkungen

Für das Modul Pro-DIO-32 Rev. B ist **DIGOUT** anwendbar, wir empfehlen aber die Verwendung der Anweisung **DIGOUT_F**, die schneller ist und erheblich weniger Programmspeicher benötigt.

Der angesprochene Kanal muss zuerst mit der Anweisung **DIGPROG1** oder **DIGPROG2** als Ausgang programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Die Anweisung **DIGOUT** besteht aus einer Sequenz von Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Die Anweisung darf in zwei parallel laufenden Prozessen mit unterschiedlicher Priorität nicht auf dasselbe Modul angewendet werden:

Ein niedrig priorisierter Prozess kann inmitten der **DIGOUT** Sequenz von einem höher priorisierten Prozess unterbrochen werden. Wenn der höher priorisierte Prozess während dieser Unterbrechung die Stellung der digitalen Ausgänge ändert, geht diese Änderung verloren, wenn der niederpriorisierte Prozess den manipulierten Wert mit **DIGOUT_WORD** zurückschreibt.

Mehrere parallel laufende hoch priorisierte Prozesse können die Anweisung **DIGOUT** problemlos verwenden, da sich hoch priorisierte Prozesse nicht gegenseitig unterbrechen.

Siehe auch

DIGPROG1, **DIGPROG2**

DIGOUT_F, **DIGOUT_BITS_F**, **DIGOUT_WORD1**, **DIGOUT_WORD2**, **GET_DIGOUT_WORD1**, **GET_DIGOUT_WORD2**

Gültig für Module

DIO-32, DIO-32 Rev. B, REL-16, TRA-16 Rev. A, TRA-16 Rev. B

DIGOUT



Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
    REM nur für DIO32: Kanäle als Ausgang setzen
    DIGPROG1(1,0FFFFh)      'DIO15:00 (Low-Word) als Ausgang
    DIGPROG2(1,0FFFFh)      'DIO31:16 (High-Word) als Ausgang

EVENT:
    value = ADC(1,1)         'Messwerterfassung
    IF (value < 100) THEN    'Grenzwert unterschritten
        DIGOUT(1,2,0)       'Dig. Ausgang 2 zurücksetzen
    ENDIF
```

DIGOUT_BITS_F setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel High oder den Pegel Low.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIGOUT_BITS_F(module, set, clear)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>set</code>	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
<code>clear</code>	Bitmuster, mit dem einzelne digitale Ausgänge auf den Pegel Low gesetzt werden. Bit = 0: Ausgang unverändert lassen. Bit = 1: Ausgang auf Pegel Low setzen.	LONG

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

Bemerkungen

Die angesprochenen Kanäle müssen zunächst mit den Anweisungen **DIGPROG1** und **DIGPROG2** als Ausgänge programmiert werden. **DIGOUT_BITS_F** beeinflusst keine Kanäle, die als Eingang programmiert sind.

DIGOUT_BITS_F kann beliebige Ausgänge löschen oder setzen, ohne den Zustand der anderen Ausgänge zu ändern. Die logische Verknüpfung der entsprechenden Register erfolgt bei diesem Befehl auf Hardware-Ebene und läuft damit wesentlich schneller ab als beim Befehl **DIGOUT**, der auf Software-Ebene arbeitet.

Zur Klarheit weisen wir darauf hin, dass Sie die im Bitmuster `set` gesetzten Bits nicht gleichzeitig im Bitmuster `clear` setzen dürfen und umgekehrt.

Siehe auch

[DIGOUT](#), [DIGOUT_F](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#), [DIGPROG1](#), [DIGPROG2](#)

Gültig für Module

DIO-32 Rev. B, TRA-16 Rev. B

DIGOUT_BITS_F

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    REM nur für DIO32: Kanäle als Ausgang setzen
    DIGPROG1(1,0FFFFh)      'DIO15:00 (Low-Word) als Ausgang
    DIGPROG2(1,0FFFFh)      'DIO31:16 (High-Word) als Ausgang

EVENT:
    IF (PAR_1 = 1) THEN      'Bedingung abfragen
        REM MSB jedes einzelnen Bytes setzen, alle anderen Bits löschen
        DIGOUT_BITS_F(1,8888h,7777h)
    ELSE
        REM ungeradzahlige Bits setzen, geradzahlige Bits löschen
        DIGOUT_BITS_F(1,5555h,0AAAAh)
    ENDIF
```

DIGOUT_F setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIGOUT_F(module,output,value)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
output	Nummer des Ausgangs, der angesprochen werden soll (0...31).	LONG
value	Neuer Zustand für den gewählten Ausgang: 0: Pegel Low. 1: Pegel High.	LONG

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit den Anweisungen **DIGPROG1** und **DIGPROG2** als Ausgänge programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Die logische Verknüpfung der entsprechenden Register erfolgt bei diesem Befehl auf Hardware-Ebene und läuft damit wesentlich schneller ab als beim Befehl **DIGOUT**, der auf Software-Ebene arbeitet.

Siehe auch

[DIGOUT](#), [DIGOUT_BITS_F](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#), [DIGPROG1](#), [DIGPROG2](#)

Gültig für Module

DIO-32 Rev. B, TRA-16 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
  REM nur für DIO32: Kanäle als Ausgang setzen
  DIGPROG1(1,0FFFFh)      'DIO15:00 (Low-Word) als Ausgang
  DIGPROG2(1,0FFFFh)      'DIO31:16 (High-Word) als Ausgang

EVENT:
  REM Low-Word (Bits 0...15) einlesen und prüfen, ob das MSB
  REM gesetzt ist
  IF (DIGIN_WORD1(1) AND 8000h = 1) THEN
    DIGOUT_F(2,31,0)      'Falls MSB gesetzt, Bit 31 löschen
  ELSE
    DIGOUT_F(2,31,1)      'Falls MSB gelöscht, Bit 31 setzen
  ENDIF
```

DIGOUT_F

DIGOUT_LONG_F

DIGOUT_LONG_F setzt oder löscht durch den übergebenen 32 Bit-Wert alle Ausgänge des angegebenen Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIGOUT_LONG_F(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1...255). LONG
pattern	Bitmuster, nach dem die digitalen Ausgänge gesetzt werden: LONG Bit = 0: Ausgang auf Pegel Low setzen. Bit = 1: Ausgang auf Pegel High setzen.

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

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit den Anweisungen **DIGPROG1** und **DIGPROG2** als Ausgänge programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Siehe auch

[DIGOUT](#), [DIGOUT_F](#), [DIGOUT_BITS_F](#), [DIGOUT_WORD1](#),
[DIGOUT_WORD2](#), [DIGPROG1](#), [DIGPROG2](#)

Gültig für Module

DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
  DIGPROG1(1,0FFFFh)      'DIO15:00 (Low-Word) als Ausgang
  DIGPROG2(1,0FFFFh)      'DIO31:16 (High-Word) als Ausgang

EVENT:
  DIGOUT_LONG_F(1,1000000) 'Den Wert 1 Mio. als Binärwert
                           'auf die DI0s ausgeben
```

DIGOUT_WORD1 setzt gleichzeitig die digitalen Ausgänge 0...15 des angegebenen Moduls auf einen bestimmten Pegel.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIGOUT_WORD1(module, pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

pattern Bitmuster, nach dem die digitalen Ausgänge LONG gesetzt werden:
 Bit = 0: Ausgang auf Pegel Low setzen.
 Bit = 1: Ausgang auf Pegel High setzen.

Bitnr.	31...16	15	...	1	0
Ausgang	–	15	...	1	0

Bemerkungen

Bei den Modulen Pro-DIO-32 und Pro-DIO-32 Rev. B müssen die angesprochenen Leitungen zunächst mit der Anweisung **DIGPROG1** als Ausgänge programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Siehe auch

DIG_LATCH, DIG_READLATCH1, DIG_READLATCH2, DIG_WRITELATCH1, DIG_WRITELATCH2, DIG_WRITELATCH32
 DIGPROG1, DIGPROG2, DIGIN_WORD1, DIGIN_WORD2, DIGOUT, DIGOUT_WORD2
 DIGIN_LONG_F, DIGOUT_BITS_F, DIGOUT_F, DIGOUT_LONG_F
 GET_DIGOUT_LONG,
 GET_DIGOUT_WORD1, GET_DIGOUT_WORD2

Gültig für Module

DIO-32, DIO-32 Rev. B, REL-16, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
  REM nur für DIO32: Kanäle als Ausgang setzen
  DIGPROG1(4,0FFFFh) 'DIO15:00 (Low-Word) als Ausgang

EVENT:
  value = ADC(1,1) 'Messwerterfassung
  IF (value > 3000) THEN 'Grenzwert überschritten?
    DIGOUT_WORD1(4,111b) 'Ausgänge 0, 1 u. 2 des Moduls 4
    'setzen, alle anderen Ausgänge werden
    'zurückgesetzt.
  ENDIF
```

DIGOUT_WORD1

DIGOUT_WORD2

DIGOUT_WORD2 setzt gleichzeitig die digitalen Ausgänge 16...31 des angegebenen Moduls auf einen bestimmten Pegel.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIGOUT_WORD2 (module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...255). LONG
pattern	Bitmuster, nach dem die digitalen Ausgänge gesetzt werden: Bit = 0: Ausgang auf Pegel Low setzen. Bit = 1: Ausgang auf Pegel High setzen.

Bitnr.	31...16	15	...	1	0
Ausgang	–	31	...	17	16

Bemerkungen

Die angesprochenen Leitungen müssen zunächst mit der Anweisung **DIGPROG2** als Ausgänge programmiert werden.

Kanäle, die als Eingang programmiert sind, werden nicht beeinflusst.

Siehe auch

[DIG_LATCH](#), [DIG_READLATCH1](#), [DIG_READLATCH2](#), [DIG_WRITELATCH1](#), [DIG_WRITELATCH2](#), [DIG_WRITELATCH32](#)
[DIGIN_WORD1](#), [DIGIN_WORD2](#), [DIGOUT](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#)
[DIGPROG1](#), [DIGPROG2](#), [DIGIN_LONG_F](#), [DIGOUT_BITS_F](#), [DIGOUT_F](#), [DIGOUT_LONG_F](#)
[GET_DIGOUT_LONG](#),
[GET_DIGOUT_WORD1](#), [GET_DIGOUT_WORD2](#)

Siehe auch

[DIGIN_WORD2](#)

Gültig für Module

DIO-32, DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
    DIGPROG2 (1, 0FFFFh)      'DIO31:16 (High-Word) als Ausgang

EVENT:
    value = ADC (1, 1)        'Messwarterfassung
    IF (value > 3000) THEN    'Grenzwert überschritten?
        DIGOUT_WORD2 (4, 1011b) 'Ausgänge 16, 17 u. 19 setzen, alle
                                'anderen Ausgänge werden
                                'zurückgesetzt!
    ENDIF
```


DIGPROG1 programmiert die digitalen Kanäle 0...15 des angegebenen Moduls als Ein- oder Ausgang.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
DIGPROG1(module,pattern)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

pattern Bitmuster, nach dem die Kanäle als Ein- oder Ausgang gesetzt werden: LONG
 Bit = 0: Kanal als Eingang setzen.
 Bit = 1: Kanal als Ausgang setzen.

Bitnr.	Modul	31...16	15	...	8	7	...	0
Kanalnr.	DIO-32 Rev. A	–	15	...	08	07	...	00
	DIO-32 Rev. B	–	–	–	15:08	–	–	07:00

Bemerkungen

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

Beachten Sie die unterschiedliche Programmierung der Module:

- Pro-DIO-32 Rev. A: Jeder einzelne Kanal kann als Eingang oder Ausgang gesetzt werden.
- Pro-DIO-32 Rev. B: Die Kanäle können nur in Gruppen zu je 8 als Ein- oder Ausgang gesetzt werden (nur 2 relevante Bits, die anderen Bits werden ignoriert).

Siehe auch

DIG_LATCH, DIG_READLATCH1, DIG_READLATCH2, DIG_WRITELATCH1, DIG_WRITELATCH2, DIG_WRITELATCH32

DIGIN_WORD1, DIGIN_WORD2, DIGOUT, DIGOUT_WORD1, DIGOUT_WORD2

DIGPROG2, DIGIN_LONG_F, DIGOUT_BITS_F, DIGOUT_F, DIGOUT_LONG_F

GET_DIGOUT_LONG,
GET_DIGOUT_WORD1, GET_DIGOUT_WORD2

Gültig für Module

DIO-32, DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
INIT:
```

```
    DIGPROG1(1, 11111111b)  'Konfiguriert Kanal 0...7 des DIO-
                             'Moduls Nr. 1 als Ausgänge und Kanal
                             '8...15 als Eingänge
```

DIGPROG1

DIGPROG2

DIGPROG2 programmiert die Kanäle 16...31 des angegebenen Moduls als Ein- oder Ausgang.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
DIGPROG2 (module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
pattern	Bitmuster, nach dem die Kanäle als Ein- oder Ausgang gesetzt werden: Bit = 0: Kanal als Eingang setzen. Bit = 1: Kanal als Ausgang setzen.	LONG

Bitnr.	Modul	31...16	15	...	8	7	...	0
Kanalnr.	DIO-32 Rev. A	–	31	...	24	23	...	16
	DIO-32 Rev. B	–	–	–	31:24	–	–	23:16

Bemerkungen

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

Beachten Sie die unterschiedliche Programmierung der Module:

- Pro-DIO-32 Rev. A: Jeder einzelne Kanal kann als Eingang oder Ausgang gesetzt werden.
- Pro-DIO-32 Rev. B: Die Kanäle können nur in Gruppen zu je 8 als Ein- oder Ausgang gesetzt werden (nur 2 relevante Bits, die anderen Bits werden ignoriert).

Siehe auch

[DIG_LATCH](#), [DIG_READLATCH1](#), [DIG_READLATCH2](#), [DIG_WRITELATCH1](#), [DIG_WRITELATCH2](#), [DIG_WRITELATCH32](#)

[DIGIN_WORD1](#), [DIGIN_WORD2](#), [DIGOUT](#), [DIGOUT_WORD1](#), [DIGOUT_WORD2](#)

[DIGPROG1](#), [DIGIN_LONG_F](#), [DIGOUT_BITS_F](#), [DIGOUT_F](#), [DIGOUT_LONG_F](#)

[GET_DIGOUT_LONG](#),
[GET_DIGOUT_WORD1](#), [GET_DIGOUT_WORD2](#)

Gültig für Module

DIO-32, DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
DIGPROG2 (1, 11111111b) 'Konfiguriert Kanal 16...23 des  
                          'DIO-Moduls Nr. 1 als Ausgänge und  
                          'Kanal 24...31 als Eingänge
```

EXTLCH_ENABLE gibt alle Latch-Eingänge auf dem angegebenen Modul entweder frei oder sperrt sie. Die Latch-Eingänge werden über die Nummer des zugehörigen Zählers ausgewählt.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
EXTLCH_ENABLE(module, cnt_select)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

cnt_select Bitmuster zur Auswahl der Zähler und zur Einstellung des Latch-Zustands: LONG

Bit = 0: Latch-Eingang sperren.
Bit = 1: Latch-Eingang freigeben.

Bitnr.	31:5	3	2	1	0
Zähler-Nr.	–	4	3	2	1

Bemerkungen

Im Ordner <C:\ADwin\ADbasic\samples_ADwin_PRO> finden Sie das Beispielprogramm <Pro-CNT-VR4-L-I_CLK-DIR.BAS>.

Siehe auch

[CNT_CLEAR](#), [CNT_ENABLE](#), [CNT_LATCH](#), [CNT_READ32](#), [CNT_READLATCH32](#), [CNT_SETMODE](#)

Gültig für Module

CNT-VR4L(-I)

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
REM Latch-Eingänge der Zähler 1+2 freigeben,
REM Latch-Eingänge der Zähler 3+4 sperren
EXTLCH_ENABLE(1, 0011b)
```

EXTLCH_ENABLE

GET_DIGOUT_LONG

GET_DIGOUT_LONG gibt den Inhalt des Ausgangs-Latches (Register für digitale Ausgänge) auf dem angegebenen Modul zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = GET_DIGOUT_LONG(module)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
ret_val	Inhalt des Ausgangs-Latches (Bits 31:00).	LONG

Bemerkungen

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

Siehe auch

DIG_LATCH, DIG_READLATCH1, DIG_READLATCH2, DIG_WRITELATCH1, DIG_WRITELATCH2, DIG_WRITELATCH32, EXTLCH_ENABLE
DIGPROG1, DIGPROG2, DIGIN_WORD1, DIGIN_WORD2, DIGOUT, DIGOUT_WORD1, DIGOUT_WORD2
DIGIN_LONG_F, DIGOUT_BITS_F, DIGOUT_F, DIGOUT_LONG_F
GET_DIGOUT_WORD1, GET_DIGOUT_WORD2

Gültig für Module

DIO-32

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
  
EVENT:  
PAR_1 = GET_DIGOUT_LONG(1) 'Bits 31:00 aus dem Latch zurücklesen'
```

GET_DIGOUT_WORD1 gibt das untere Wort (Bit 0...15) des Ausgangs-Latches (Register für digitale Ausgänge) des angegebenen Moduls zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = GET_DIGOUT_WORD1(module)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
ret_val	Inhalt des Ausgangs-Latches (Bits 15:00).	LONG

Bemerkungen

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

Siehe auch

DIG_LATCH, DIG_READLATCH1, DIG_READLATCH2, DIG_WRITELATCH1, DIG_WRITELATCH2, DIG_WRITELATCH32, EXTLCH_ENABLE
DIGPROG1, DIGPROG2, DIGIN_WORD1, DIGIN_WORD2, DIGOUT, DIGOUT_WORD1, DIGOUT_WORD2
DIGIN_LONG_F, DIGOUT_BITS_F, DIGOUT_F, DIGOUT_LONG_F
GET_DIGOUT_LONG, GET_DIGOUT_WORD2

Gültig für Module

DIO-32, DIO-32 Rev. B, REL-16, TRA-16 Rev. A, TRA-16 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value AS LONG

INIT:
value = GET_DIGOUT_WORD1(1) 'aktuellen Wert des Ausgangs-
                             'registers lesen
value = value AND 0FFFFh 'letztes Bit (Bit 0) auf 0 setzen
DIGOUT_WORD1(value) 'geänderten Wert zurückschreiben
```

GET_DIGOUT_WORD1

GET_DIGOUT_WORD2

GET_DIGOUT_WORD2 gibt das obere Wort (Bit 16...31) des Ausgangs-Latches (Register für digitale Ausgänge) des angegebenen Moduls zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = GET_DIGOUT_WORD2(module)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
ret_val	Inhalt des Ausgangs-Latches (Bits 31:16).	LONG

Bemerkungen

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

Siehe auch

DIG_LATCH, DIG_READLATCH1, DIG_READLATCH2, DIG_WRITELATCH1, DIG_WRITELATCH2, DIG_WRITELATCH32, EXTLCH_ENABLE
DIGPROG1, DIGPROG2, DIGIN_WORD1, DIGIN_WORD2, DIGOUT, DIGOUT_WORD1, DIGOUT_WORD2
DIGIN_LONG_F, DIGOUT_BITS_F, DIGOUT_F, DIGOUT_LONG_F
GET_DIGOUT_LONG, GET_DIGOUT_WORD1

Gültig für Module

DIO-32, DIO-32 Rev. B

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM value AS LONG  
  
INIT:  
value = GET_DIGOUT_WORD2(1) 'aktuellen Wert der  
                             'Ausgangsregister lesen  
value = value AND 0FFFDh 'vorletztes Bit (Bit 17) auf 0 setzen  
DIGOUT_WORD2(value)      'geänderten Wert zurückschreiben
```

PWM_ENABLE gibt alle internen Zähler frei oder stoppt sie. Die Zähler werden über die Nummer des zugehörigen PWM-Ausgangs gewählt.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

PWM_ENABLE(module, output)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

output Bitmuster zur Auswahl der PWM-Ausgänge und zur Einstellung des Zähler-Zustands: LONG

Bit = 0: Zähler sperren.
Bit = 1: Zähler freigeben.

Bitnr.	31:5	3	2	1	0
PWM-Kanal	–	4	3	2	1

Bemerkungen

Diese Anweisung ändert nicht den Pegel der PWM-Ausgänge. Wenn Sie den Pegel der PWM-Ausgänge ändern möchten, tun Sie dies mit dem Befehl **PWM_OUT** und stoppen anschließend mit **PWM_ENABLE** die zugehörigen internen Zähler.

Sobald und solange die Zähler gestoppt sind, können die Pegel der PWM-Ausgänge nicht geändert werden.

Siehe auch

[PWM_OUT](#), [PWM_SET](#)

Gültig für Module

PWM-4(-I)

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    PWM_ENABLE(1,1)           'PWM-Ausgang 1 für die Ausgabe
                              'aktivieren
    PWM_SET(1,1,0,2500,2500) 'PWM-Signal für Ausgang 1 einstellen
```

PWM_ENABLE



PWM_OUT

PWM_OUT setzt einen bestimmten PWM-Ausgabekanal des angegebenen Moduls auf den Pegel High oder Low.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
PWM_OUT(module, channel, level)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer des PWM-Ausgabekanals (1...4).	LONG
level	Pegel, auf den der Kanal gesetzt werden soll: 0: auf Pegel Low setzen. 1: auf Pegel High setzen.	LONG

Bemerkungen

Dieser Befehl hat nur dann eine Funktion, wenn der zugehörige Zähler des gewählten Kanals freigegeben ist.

Siehe auch

[PWM_ENABLE](#), [PWM_SET](#)

Gültig für Module

PWM-4(-I)

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
  
INIT:  
  PWM_ENABLE(1,3)           'Zähler der PWM-Ausgänge 1 und 2  
                             'freigeben  
  PWM_SET(1,1,0,2500,2500)  'PWM-Signal für Ausgang 1 einstellen  
  PWM_OUT(1,2,1)           'PWM-Ausgang 2 auf den logischen Wert  
                             '"1" setzen
```


PWM_SET setzt die Voreinstellungen eines bestimmten PWM-Ausgabekanals auf dem angegebenen Modul.

Die Voreinstellungen sind:

- Faktor des Vorteilers (Prescaler)
- Zählwert der Low-Zeit
- Zählwert der High-Zeit

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

PWM_SET(module, channel, prescale, low, high)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer des PWM-Ausgabekanals (1...4).	LONG
prescale	Exponent (0...7) für den Faktor des Vorteilers, siehe Formel.	LONG
low	Zählwert der Low-Zeit (1...32768), siehe Formel.	LONG
high	Zählwert der High-Zeit (1...32768), siehe Formel.	LONG

Bemerkungen

Die Ausgangsfrequenz (nach dem Vorteiler) wird nach folgender Formel berechnet:

$$f_{\text{out}} = \frac{5\text{MHz}}{2^{\text{prescale}} \cdot (\text{low} + \text{high})}$$

Die Zählwerte für Low- und High-Zeit stehen für die Anzahl der Impulse nach dem Vorteiler, die der interne Zähler erreichen muss, um den Logik-Pegel zu wechseln.

Die niedrigste Ausgangsfrequenz – bei noch einstellbarem Tastverhältnis von annähernd 0...100% – beträgt ca. 0,6 Hz.

Die höchste Ausgangsfrequenz, bei der das Tastverhältnis noch in 1%-Schritten einstellbar ist, beträgt ca. 50kHz.

Siehe auch

[PWM_ENABLE](#), [PWM_OUT](#)

Gültig für Module

PWM-4(-I)

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    PWM_ENABLE(1,3)      'PWM-Ausgang 1 und 2 für die Ausgabe
                          'aktivieren
    PWM_SET(1,1,0,2500,2500) 'PWM-Signal für Ausgang 1 einstellen
    PWM_OUT(1,2,1)        'PWM-Ausgang 2 auf den logischen Wert
                          '"1" setzen
```

PWM_SET

SSI_MODE

SSI_MODE stellt auf dem angegebenen Modul den Modus aller SSI-Decoder ein, entweder „single shot“ (einzelne lesen) und „continuous“ (kontinuierlich lesen).

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

SSI_MODE(module, pattern)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
pattern	Betriebsmodus der SSI-Decoder, angegeben als Bitmuster. Jedem Decoder ist ein Bit zugeordnet (siehe Tabelle). Bit = 0: Modus „single shot“, der Encoder wird einmal ausgelesen. Bit = 1: Modus „continuous“, der Encoder wird kontinuierlich ausgelesen.	LONG

Bitnr.	31:2	1	0
SSI-Decoder	–	2	1

Bemerkungen

Wenn Sie den Modus „continuous“ wählen, startet das Auslesen des entsprechenden Encoders sofort. Die Anweisung **SSI_START** ist hierzu nicht erforderlich.

Manche Encoder-Typen liefern im Modus „continuous“ gelegentlich den falschen Messwert 0 (Null) anstelle des korrekten Messwerts zurück. Im Modus „single shot“ tritt dieser Fehler nicht auf.

Siehe auch

[SSI_READ](#), [SSI_SET_BITS](#), [SSI_SET_CLOCK](#), [SSI_START](#), [SSI_STATUS](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
  SSI_SET_CLOCK(1,200)      'CLK (Taktrate) = 50 kHz
  SSI_MODE(1,3)             'Continuous-Mode einstellen
                             '(für beide Encoder)
  SSI_SET_BITS(1,1,23)      'Anzahl Encoder-Bits = 23 (Encoder 1)
  SSI_SET_BITS(1,2,23)      'Anzahl Encoder-Bits = 23 (Encoder 2)

EVENT:
  PAR_1 = SSI_READ(1,1)     'Positionswert (Encoder 1) auslesen
                             'und anzeigen
  PAR_2 = SSI_READ(1,2)     'Positionswert (Encoder 2) auslesen
                             'und anzeigen
```

SSI_READ gibt den zuletzt gespeicherten Zählerstand eines bestimmten SSI-Zählers auf dem angegebenen Modul zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = SSI_READ(module, dcd_r_no)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
dcd_r_no	Nummer (1, 2) des SSI-Decoders, dessen Zählerstand auszulesen ist.	LONG
ret_val	Letzter Zählerstand des SSI-Zählers (= Absolutwert-Position des Encoders).	LONG

Bemerkungen

Ein Encoder-Wert wird dann gespeichert, wenn die durch **SSI_SET_BITS** angegebene Anzahl von Bits eingelesen wurde.

Es wird immer diejenige Anzahl an Bits zurückgegeben, die mit der Anweisung **SSI_SET_BITS** eingestellt wurde, auch wenn dies nicht mit der Auflösung des Encoders übereinstimmt.

In diesem Fall ist der zurückgegebene Zählerstand abhängig vom Encoder (siehe Dokumentation des Herstellers). In der Regel gilt:

- Wenn der Encoder eine größere Auflösung besitzt, werden dessen überzählige niederwertigste Bits nicht genutzt.
- Besitzt der Encoder eine kleinere als die eingestellte Auflösung, wird für jedes fehlende höchstwertige Bit eine 0 (Null) gelesen.

Siehe auch

[SSI_MODE](#), [SSI_SET_BITS](#), [SSI_SET_CLOCK](#), [SSI_START](#), [SSI_STATUS](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM m, n, y AS LONG

INIT:
  SSI_SET_CLOCK(1,50)      'CLK (Taktrate) = 200 kHz
  SSI_MODE(1,1)            'Continuous-Mode setzen (Encoder 1)
  SSI_SET_BITS(1,1,23)     'Anzahl Encoder-Bits = 23 (Encoder 1)

EVENT:
  PAR_1 = SSI_READ(1,1)    'Positionswert (Encoder 1) auslesen
                           'und anzeigen.
  REM Falls es sich um einen Encoder mit Gray-Code handelt:
  m = 0                    'Werte der letzten Wandlung löschen
  y = 0                    '  _" _
  FOR n = 1 TO 32          'Alle 32 mögl. Bits durchgehen
    m = (SHIFT_RIGHT(PAR_1, (32 - n)) AND 1) XOR m
    y = (SHIFT_LEFT(m, (32 - n))) OR y
  NEXT n
  PAR_9 = y                'Das Ergebnis der Gray-/Binär-
                           'Wandlung in PAR_9
```

SSI_READ



SSI_SET_BITS

SSI_SET_BITS stellt für einen bestimmten SSI-Zähler auf dem angegebenen Modul die Anzahl der zu Bits ein, die einen vollständigen Encoder-Wert bilden. Die Zahl der Bits sollte mit der Auflösung des Encoders identisch sein.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

SSI_SET_BITS(module, dcd_r_no, bit_no)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>dcd_r_no</code>	Nummer (1, 2) des SSI-Decoders, dessen Auflösung einzustellen ist.	LONG
<code>bit_no</code>	Anzahl der Bits (1...32) der zu lesenden Bits für einen Encoder-Wert (entspricht der Encoder-Auflösung).	LONG

Bemerkungen

Die Auflösung (Anzahl der Bits) des SSI-Encoders sollte mit der Anzahl der zu übertragenden Bits übereinstimmen.

Es wird immer diejenige Anzahl an Bits für einen Encoder-Wert erwartet, die mit **SSI_SET_BITS** eingestellt wurde, auch wenn dies nicht mit der Auflösung des Encoders übereinstimmt.

In diesem Fall ist der zurückgegebene Zählerstand abhängig vom Encoder (siehe Dokumentation des Herstellers). In der Regel gilt:

- Wenn der Encoder eine größere Auflösung besitzt, werden dessen überzählige niederwertigste Bits nicht genutzt.
- Besitzt der Encoder eine kleinere als die eingestellte Auflösung, wird für jedes fehlende höchstwertige Bit eine 0 (Null) gelesen.

Siehe auch

[SSI_MODE](#), [SSI_READ](#), [SSI_SET_CLOCK](#), [SSI_START](#), [SSI_STATUS](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
SSI_SET_CLOCK(1, 50)      'CLK (Taktrate) = 200 kHz
SSI_MODE(1, 3)            'Continuous-Mode einstellen (für
                           'beide Encoder)
SSI_SET_BITS(1, 1, 10)    '10 Encoder-Bits für Encoder 1
SSI_SET_BITS(1, 2, 25)    '25 Encoder-Bits für Encoder 2
```

EVENT:

```
PAR_1 = SSI_READ(1, 1)    'Positionswert (Encoder 1) auslesen
                           'und anzeigen
PAR_2 = SSI_READ(1, 2)    'Positionswert (Encoder 2) auslesen
                           'und anzeigen
```



SSI_SET_CLOCK stellt die Taktrate (ca. 40kHz bis 1 MHz) auf dem angegebenen Modul ein, mit der der Encoder getaktet wird.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

SSI_SET_CLOCK(module,prescale)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
prescale	Teilerfaktor (10...255) zur Einstellung der Taktrate nach der Formel: Taktrate = 10MHz / 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.

Teilerfaktoren kleiner 10 werden automatisch auf den Wert 10 korrigiert; von Werten über 255 werden die niederwertigsten 8 Bits als Teilerfaktor verwendet.

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

Siehe auch

[SSI_MODE](#), [SSI_READ](#), [SSI_SET_BITS](#), [SSI_START](#), [SSI_STATUS](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
SSI_SET_CLOCK(1,10)    'CLK (Taktrate) = 1 MHz
SSI_MODE(1,3)          'Continuous-Mode setzen
                        '(für beide Encoder)
SSI_SET_BITS(1,1,10)   'Anzahl Encoder-Bits = 10 (Encoder 1)
SSI_SET_BITS(1,2,25)   'Anzahl Encoder-Bits = 25 (Encoder 2)
```

EVENT:

```
PAR_1 = SSI_READ(1,1)  'Positionswert (Encoder 1) auslesen
                        'und anzeigen
PAR_2 = SSI_READ(1,2)  'Positionswert (Encoder 2) auslesen
                        'und anzeigen
```

SSI_SET_CLOCK



SSI_START

SSI_START startet auf dem angegebenen Modul das Auslesen eines oder beider SSI-Encoder (nur im Modus single shot).

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
SSI_START(module, dcd_r_no)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
dcd_r_no	Bitmuster zur Auswahl der SSI-Decoder, die gestartet werden sollen: Bit = 0: keine Funktion. Bit = 1: Auslesen des SSI-Decoders starten.	LONG

Bitnr.	31:2	1	0
SSI-Decoder	–	2	1

Bemerkungen

Im Modus „continuous“ ist die Anweisung ohne Funktion, weil dann die Encoder-Werte ohnehin kontinuierlich ausgelesen werden.

Ein Encoder-Wert wird dann gespeichert, wenn die durch **SSI_SET_BITS** angegebene Anzahl von Bits eingelesen wurde.

Es wird immer ein vollständiger Encoder-Wert übertragen, auch wenn währenddessen der Modus umgestellt wird.

Siehe auch

[SSI_MODE](#), [SSI_READ](#), [SSI_SET_BITS](#), [SSI_SET_CLOCK](#), [SSI_STATUS](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
  SSI_SET_CLOCK(1,250)      'CLK (Taktrate) = ca. 40 kHz
  SSI_MODE(1,0)             'Single shot-Mode einstellen
                             '(beide Zähler)
  SSI_SET_BITS(1,1,23)      'Anzahl Encoder-Bits = 23 (Encoder 1)
  SSI_SET_BITS(1,2,23)      'Anzahl Encoder-Bits = 23 (Encoder 2)

EVENT:
  SSI_START(1,3)            'Positionswert von Encoder 1 & 2 lesen
DO
  'Für Encoder 1:
UNTIL (SSI_STATUS(1,1) = 0)
REM Wenn Positionswert komplett gelesen ist, dann ...
PAR_1 = SSI_READ(1,1)       'Positionswert auslesen und anzeigen
DO
  'Für Encoder 2:
UNTIL (SSI_STATUS(1,2) = 0)
REM Wenn Positionswert komplett gelesen ist, dann ...
PAR_1 = SSI_READ(1,2)       'Positionswert auslesen und anzeigen
```



SSI_STATUS liefert für einen bestimmten Decoder den aktuellen Lese-Status auf dem angegebenen Modul zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = SSI_STATUS(module, dcd_r_no)
```

Parameter

module	Eingestellte Moduladresse (1...255).	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

[SSI_MODE](#), [SSI_READ](#), [SSI_SET_BITS](#), [SSI_SET_CLOCK](#), [SSI_START](#)

Gültig für Module

CO4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
  SSI_SET_CLOCK(1,250)      'CLK (Taktrate) = ca. 40 kHz
  SSI_MODE(1,0)             'Single shot-Mode einstellen
                             '(beide Zähler)
  SSI_SET_BITS(1,1,23)      'Anzahl Encoder-Bits = 23 (Encoder 1)
  SSI_SET_BITS(1,2,23)      'Anzahl Encoder-Bits = 23 (Encoder 2)

EVENT:
  SSI_START(1,3)            'Positionswert von Encoder 1 & 2 lesen
  DO                        'Für Encoder 1:
  UNTIL (SSI_STATUS(1,1) = 0)
  REM Wenn Positionswert komplett gelesen ist, dann ...
  PAR_1 = SSI_READ(1,1)     'Positionswert auslesen und anzeigen
  DO                        'Für Encoder 2:
  UNTIL (SSI_STATUS(1,2) = 0)
  REM Wenn Positionswert komplett gelesen ist, dann ...
  PAR_1 = SSI_READ(1,2)     'Positionswert auslesen und anzeigen
```

SSI_STATUS



COMP_DIGIN_WORD gibt den aktuellen Status des Schwellenwert-Vergleichs für alle Kanäle auf dem angegebenen Modul zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = COMP_DIGIN_WORD(module)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
ret_val	Bitmuster, das den Status des Schwellenwert-Vergleichs der Kanäle wiedergibt (Zuordnung zu den Kanälen siehe Tabelle): Bit = 0: Messwert < unterer Schwellenwert. Bit = 1: Messwert > oberer Schwellenwert.	LONG

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

Bemerkungen

Die Bewertung der Messwerte erfolgt über Schaltschwellen (Hysteresis), die mit **COMP_SET** festgelegt werden. Der Status ist umso stabiler, je weiter die Schaltschwellen auseinander liegen.

Diese Anweisung kann z.B. verwendet werden, wenn Analogsignale im Modus single-ended an den Eingängen angelegt sind.

Siehe auch

[COMP_READ](#), [COMP_RESET](#), [COMP_FIFO_SELECT](#), [COMP_SET](#), [COMP_DIGIN_WORD_DIFF](#), [COMP_FIFO_READ](#)

Gültig für Module

COMP16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    COMP_SET(1,3,500,300)    'Schwellenwerte des Kanals 3
                             'auf +3V und +1V setzen
                             '(bei Spannungsbereich -2V ... +8,23V)
    COMP_SET(1,4,600,350)    'Schwellenwerte des Kanals 4
                             'auf +4V und +1,5V setzen
                             '(bei Spannungsbereich -2V ... +8,23V)

EVENT:
    REM Vergleichs-Status des Kanals 3 abfragen
    PAR_1=(COMP_DIGIN_WORD(1) AND 00100b)
    REM Vergleichs-Status des Kanals 4 abfragen
    PAR_2=(COMP_DIGIN_WORD(1) AND 01000b)
```

COMP_DIGIN_WORD

COMP_DIGIN_WORD_DIFF

COMP_DIGIN_WORD_DIFF gibt den aktuellen Status des Schwellenwert-Vergleichs auf dem angegebenen Modul zurück.

Der Vergleich wird auf den Differenzwert von je 2 Kanälen (differentielles Signal) angewendet.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = COMP_DIGIN_WORD_DIFF(module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>ret_val</code>	Bitmuster, das den Status des Schwellenwert-Vergleichs der Differenzwerte wiedergibt (Zuordnung zu den Kanalpaaren siehe Tabelle): Bit = 0: Differenz < unterer Schwellenwert. Bit = 1: Differenz > oberer Schwellenwert.	LONG

Bitnr.	31...8	7	6	...	1	0
Kanalpaar	–	15,16	13,14	...	3,4	1,2

Bemerkungen

Die Bewertung der Differenzwerte erfolgt über Schaltschwellen (Hysteresis), die mit **COMP_SET** festgelegt werden. Der Status ist umso stabiler, je weiter die Schaltschwellen auseinander liegen.

Die Kanalpaare sind in aufsteigender Reihenfolge (1/2, 3/4, ..., 15/16) festgelegt, die Differenz wird berechnet als Wert1 - Wert2, Wert3 - Wert4, ..., Wert15 - Wert16. Für den Vergleich werden jeweils die Schwellenwerte des kleineren Kanals verwendet (= ungerade Kanalnummer).

Siehe auch

[COMP_READ](#), [COMP_RESET](#), [COMP_FIFO_SELECT](#), [COMP_SET](#), [COMP_DIGIN_WORD](#), [COMP_FIFO_READ](#)

Gültig für Module

COMP16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    COMP_SET(1,3,500,300)    'Schwellenwerte des Kanals 3
                              'auf +3V und +1V setzen
                              '(bei Spannungsbereich -2V ... +8,23V)
    COMP_SET(1,13,450,50)   'Schwellenwerte des Kanals 13
                              'auf +2,5V und -1,5V setzen
                              '(bei Spannungsbereich -2V ... +8,23V)

EVENT:
    REM Different. Vergleichs-Status des Kanalpaars 3/4 abfragen
    PAR_1=(COMP_DIGIN_WORD_DIFF(1) AND 00000010b)
    REM Different. Vergleichs-Status des Kanalpaars 13/14 abfragen
    PAR_2=(COMP_DIGIN_WORD_DIFF(1) AND 01000000b)
```

COMP_FIFO_READ liest die letzten 2 x 1024 Messwerte eines Kanalpaars aus dem internen FIFO-Speicher des angegebenen Moduls und überträgt sie in 2 Felder.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

COMP_FIFO_READ(module, array1[], array2[])
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
array1[]	Zielfeld für 1024 Messwerte des Kanals mit der kleineren der beiden Kanalnummern.	ARRAY LONG
array2[]	Zielfeld für 1024 Messwerte des Kanals mit der größeren der beiden Kanalnummern.	ARRAY LONG

Bemerkungen

Im internen FIFO-Speicher des Moduls sind die jeweils letzten 1024 Messwerte von 2 Kanälen abgelegt. Sie wählen das Kanalpaar mit **COMP_FIFO_SELECT** aus.

Die Anweisung beschreibt in beiden Zielfeldern die Feldelemente `arrayx[1]...arrayx[1024]`. Beachten Sie bitte, dass die Zielfelder entsprechend groß dimensioniert sein müssen. Die übertragenen Messwerte sind Werte aus dem Bereich 0...1023.

Während diese Anweisung die Messdaten in die Zielfelder überträgt, werden keine neuen Messwerte in den Speicher geschrieben. Damit ist sichergestellt, dass ein zusammenhängendes Messwert-Paket übertragen wird.

Nach der Datenübertragung wird der Speicher automatisch wieder mit aktuellen Messwerten beschrieben.



Siehe auch

[COMP_READ](#), [COMP_RESET](#), [COMP_FIFO_SELECT](#), [COMP_SET](#), [COMP_DIGIN_WORD](#), [COMP_DIGIN_WORD_DIFF](#)

Gültig für Module

COMP16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM array1[1024] AS LONG
DIM array2[1024] AS LONG

INIT:
    COMP_FIFO_SELECT(1,2)    'Werte der Kanäle 5,6 im FIFO-Speicher
                              'ablegen

EVENT:
    COMP_FIFO_READ(1,array1,array2) '1024 Messwerte für beide
                                    'Kanäle abholen:
                                    'Kanal 5 in array1, Kanal 6 in array2
```

COMP_FIFO_SELECT

COMP_FIFO_SELECT legt das Kanalpaar fest, dessen Daten im internen FIFO-Speicher des angegebenen Moduls gespeichert werden.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
COMP_FIFO_SELECT(module, ch_pair)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
ch_pair	Zahl (0...7) zur Festlegung eines Kanalpaars; die Zuordnung zu den Kanälen siehe Tabelle.	LONG

ch_pair	7	...	1	0
Kanalnr.	15, 16	...	3, 4	1, 2

Bemerkungen

Es kann immer nur ein Kanalpaar gewählt werden, mehr oder weniger Kanäle können nicht festgelegt werden.

Im FIFO-Speicher werden die jeweils letzten 1024 Messwerte der beiden gewählten Kanäle abgelegt (= 2×1024 Werte). Sie können den FIFO-Speicher mit **COMP_FIFO_READ** auslesen.

Siehe auch

[COMP_READ](#), [COMP_RESET](#), [COMP_SET](#), [COMP_DIGIN_WORD](#),
[COMP_DIGIN_WORD_DIFF](#), [COMP_FIFO_READ](#)

Gültig für Module

COMP16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM array1[1024] AS LONG  
DIM array2[1024] AS LONG  
  
INIT:  
    COMP_FIFO_SELECT(1,2)    'Werte der Kanäle 5,6 im FIFO-Speicher  
                              'ablegen  
  
EVENT:  
    COMP_FIFO_READ(1,array1,array2) '1024 Messwerte für beide  
                                    'Kanäle abholen:  
                                    'Kanal 5 in array1, Kanal 6 in array2
```

COMP_READ gibt den aktuellen, minimalen oder maximalen Messwert eines bestimmten Kanals auf dem angegebenen Modul zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = COMP_READ(module, channel, value)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Kanalnummer (1...16).	LONG
value	Neuer Zustand für den gewählten Ausgang: 0: Aktueller Messwert. 1: Minimaler Messwert. 2: Maximaler Messwert.	LONG
ret_val	Messwert (0...1023) am gewählten Kanal in Digits.	LONG

Bemerkungen

Der Rückgabewert 0 Digits entspricht dem minimalen Analogsignal U_{\min} , der Wert 1023 Digits dem maximalen Analogsignal U_{\max} . Zur Umrechnung von Digits in Volt gilt die Formel:

$$\text{Volt} = \text{Digits} \cdot \frac{U_{\max} - U_{\min}}{1023} + U_{\min}$$

Das Messen des Analogsignals wird durch diese Funktion nicht verzögert oder unterbrochen.

Der minimale und maximale Messwert bezieht sich auf die Messwerte im Zeitraum seit dem letzten Rücksetzen mit Comp_Reset (oder seit dem Einschalten des Systems).

Siehe auch

[COMP_RESET](#), [COMP_FIFO_SELECT](#), [COMP_SET](#), [COMP_DIGIN_WORD](#), [COMP_DIGIN_WORD_DIFF](#), [COMP_FIFO_READ](#)

Gültig für Module

COMP16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
    COMP_SET(1,3,500,300)    'Schwellenwerte des Kanals 3
                              'auf +3V und +1V setzen
                              '(bei Spannungsbereich -2V ... +8,23V)
    COMP_RESET(1,100b)      'Minimal- und Maximalwerte
                              'des Kanals 3 rücksetzen

EVENT:
    PAR_1=COMP_READ(1,3,1)  'Minimum des Kanals 3 lesen
    PAR_2=COMP_READ(1,3,2)  'Maximum des Kanals 3 lesen
```

COMP_READ

Umrechnung Digits in Volt

COMP_RESET

COMP_RESET setzt auf dem angegebenen Modul die Messung des Minimal- und Maximalwerts für die ausgewählten Kanäle gleichzeitig zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
COMP_RESET(module,pattern)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
pattern	Bitmuster zum Rücksetzen der Minimal- und Maximalwerte (Zuordnung zu Kanälen siehe Tabelle). Bit = 0: Minimal- und Maximalwerte beibehalten. Bit = 1: Minimal- und Maximalwerte zurücksetzen.	LONG

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

Bemerkungen

Beim Rücksetzen wird der Maximalwert auf den Wert 0, der Minimalwert auf den Wert 1023 gesetzt. Nach dem Rücksetzen wird die Messung der beiden Werte sofort weitergeführt.

Siehe auch

[COMP_READ](#), [COMP_FIFO_SELECT](#), [COMP_SET](#), [COMP_DIGIN_WORD](#), [COMP_DIGIN_WORD_DIFF](#), [COMP_FIFO_READ](#)

Gültig für Module

COMP16 Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
REM Schwellenwerte der Kanäle 1, 3 und 4 auf +3V und +1V setzen  
REM (bei Spannungsbereich -2V ... +8,23V)  
COMP_SET(1,1,500,300)  
COMP_SET(1,3,500,300)  
COMP_SET(1,4,500,300)  
COMP_RESET(1,1101b)      'Minimal- und Maximalwerte  
                           'der Kanäle 1,3,4 zurücksetzen
```

EVENT:

```
PAR_1 = COMP_READ(1,1,1) 'Minimum des Kanals 1 lesen  
PAR_2 = COMP_READ(1,1,2) 'Maximum des Kanals 1 lesen  
PAR_3 = COMP_READ(1,3,1) 'Minimum des Kanals 3 lesen  
PAR_4 = COMP_READ(1,4,2) 'Maximum des Kanals 4 lesen
```

COMP_SET setzt den unteren und den oberen Schwellenwert für einen bestimmten Kanal des angegebenen Moduls fest.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

COMP_SET(module, channel, ValHigh, ValLow)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Kanalnummer (1...16).	LONG
ValHigh	oberer Schwellenwert (1...1023) des Kanals.	LONG
ValLow	unterer Schwellenwert (0...1022) des Kanals.	LONG

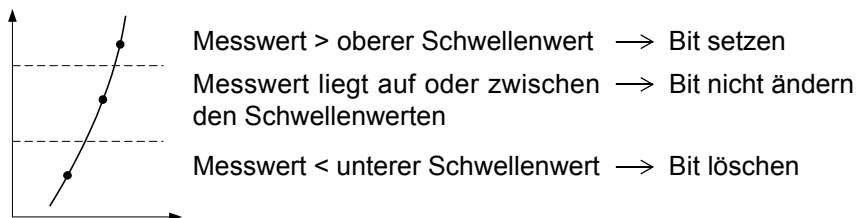
Bemerkungen

Mit der Festlegung der Schwellenwerte definieren Sie eine Hysterese bei der Bewertung des anliegenden Analogsignals.

Beispiel: Solange ein verrauschtes Analogsignal in der Nähe eines Schaltpunktes liegt, wechselt der Schaltzustand (ohne Hysterese) mehr oder weniger zufällig. Durch geschickte Wahl der Hysterese-Schwellenwerte kann dagegen der Schaltzustand stabilisiert werden.

Der obere Schwellenwert muss immer größer sein als der untere Schwellenwert, sonst funktioniert die Bewertung des Analogsignals nicht korrekt.

Die anliegenden Analogsignale (auf allen Kanälen parallel) werden mit der festgelegten Messrate (20MHz je Kanal) gemessen. Jeder einzelne Messwert wird nach folgendem Muster mit dem oberen und unteren Schwellenwert des zugehörigen Kanals verglichen:



Die Vergleichs-Ergebnisse aller Kanäle werden in einem 16 Bit-Wort gespeichert (je Kanal 1 Bit); das aktuelle Vergleichs-Ergebnis kann mit **COMP_DIGIN_WORD** ausgelesen werden.

In gleicher Weise wird der Differenzwert von je 2 Kanälen mit den Schwellenwerten verglichen und gespeichert (je Kanalpaar 1 Bit). Der Differenzwert wird berechnet als Wert₁ - Wert₂, Wert₃ - Wert₄, ..., Wert₁₅ - Wert₁₆; es werden die Schwellenwerte des Kanals mit der ungeraden Nummer verwendet.

Das aktuelle Vergleichs-Ergebnis kann mit **COMP_DIGIN_WORD_DIFF** ausgelesen werden.

Siehe auch

[COMP_READ](#), [COMP_RESET](#), [COMP_FIFO_SELECT](#), [COMP_DIGIN_WORD](#), [COMP_DIGIN_WORD_DIFF](#), [COMP_FIFO_READ](#)

Gültig für Module

COMP16 Rev. A

COMP_SET

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
INIT:
```

```
  COMP_SET(1,3,500,300)      'Schwellenwerte des Kanals 3  
                             'auf +3V und +1V setzen  
                             '(bei Spannungsbereich -2V ... +8,23V)
```


RTC_SET setzt das Datum und die Zeit auf der Echtzeituhr des angegebenen Moduls. Ungültige Werte werden nicht akzeptiert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
RTC_SET(module, year, month, day, hour, minute, second)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
year	Jahreszahl (0...63), entspricht 2000...2063.	LONG
month	Monat (1...12).	LONG
day	Tag (1...31); gültiger Wertebereich je nach Monat und Schaltjahr. .	LONG
hour	Stunde (0...23).	LONG
minute	Minute (0...59).	LONG
second	Sekunde (0...59).	LONG

Bemerkungen

Die Jahresangabe bezieht sich auf den Zeitraum 2000 - 2063.

Siehe auch

[RTC_GET](#)

Gültig für Module

Storage Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
INIT:
```

```
REM Echtzeituhr auf 4.7.2003 9:17:30 setzen
```

```
RTC_SET(1, 3, 7, 4, 9, 17, 30) 'auf Modul 1
```

RTC_SET

RTC_GET

RTC_GET gibt das Datum und die Zeit von der Echtzeituhr des angegebenen Moduls zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
RTC_GET(module, year, month, day, hour, minute, second)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
year	Jahreszahl (0...63), entspricht 2000...2063.	LONG
month	Monat (1...12).	LONG
day	Tag (1...31); gültiger Wertebereich je nach Monat und Schaltjahr. .	LONG
hour	Stunde (0...23).	LONG
minute	Minute (0...59).	LONG
second	Sekunde (0...59).	LONG

Bemerkungen

Alle Parameter (bis auf module) sind Rückgabeparameter.

Die Anweisung ersetzt die folgenden Befehle, die zeitweise verfügbar waren: **RTC_GET_YEAR**, **RTC_GET_MONTH**, **RTC_GET_DAY**, **RTC_GET_HOUR**, **RTC_GET_MINUTE**, **RTC_GET_SECOND**.

Siehe auch

[RTC_SET](#)

Gültig für Module

Storage Rev. A

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
DIM year, mon, day, h, m, s AS LONG
```

```
INIT:
```

```
REM Echtzeituhr des Moduls 1 lesen
```

```
RTC_GET(1, year, mon, day, h, m, s)
```

MEDIA_WR_BLK_L kopiert eine Anzahl an Long-Datenblöcken aus einem Feld in eine der zehn Dateien auf dem Datenträger des angegebenen Moduls. Der erste zu beschreibende Sektor der Datei ist frei wählbar.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = MEDIA_WR_BLK_L(module, f_info[], file,
                        sec_idx, sec_cnt, array[], array_idx)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>f_info[]</code>	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
<code>file</code>	Nummer (1...10) der Datei.	LONG
<code>sec_idx</code>	Index (1...m) des ersten zu beschreibenden Sektors, bezogen auf den Startsektor der Datei. Der Max.wert ist abhängig von der Dateigröße.	LONG
<code>sec_cnt</code>	Anzahl (1...12) der zu übertragenden Datenblöcke (=Sektoren) zu 128 Werten.	LONG
<code>array[]</code>	Feld, dessen Daten übertragen werden.	ARRAY LONG
<code>array_idx</code>	Index (1...n) des ersten zu übertragenden Feld-elements.	LONG
<code>ret_val</code>	Status der Medienverwaltung als Bitmuster (siehe Tabelle).	LONG

Bitnr.	31:08	07	06	05	04	03	02	01	00
	–	A8	A7	A6	A5	A4	A3	A2	A1

Falls Bit = 1 (Bit = 0: ohne Bedeutung):

- A1 Ein Medium ist im Schacht.
- A2 Unbekannter Fehler aufgetreten.
- A3 Medienverwaltung (Glue-Logik) ist beschäftigt.
- A4 Daten können gelesen werden.
- A5 Daten sind geschrieben.
- A6 Reset wird gerade ausgeführt.
- A7 Dateiende überschritten, es werden keine Daten übertragen.
- A8 Time out, das Medium hat nicht reagiert.
- Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Bevor diese Anweisung genutzt werden kann, muss das Feld `f_info[]` mit **MEDIA_RD_FILEINFO** initialisiert werden.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LOWINIT**: oder **FINISH**: eines hoch-prioren Prozesses.

Der Start-Sektor ist frei wählbar. Dadurch können Daten an bereits gespeicherte Daten angehängt werden. Achten Sie darauf, nicht mehr Datenblöcke anzugeben als in die Datei passen. Anderenfalls werden die Datenblöcke nicht gespeichert!

MEDIA_WR_BLK_L

Jeder Datenblock enthält 128 Long-Werte und wird in einem Sektor gespeichert. Wenn beispielsweise in einem ersten Schritt n Datenblöcke ab dem Start-Sektor x gespeichert wurden, werden weitere Datenblöcke angehängt, indem sie ab Sektor $x+n+1$ geschrieben werden.

Das Feld `array[]` muss mindestens `array_idx+s_cnt`×128 Werte enthalten.

Siehe auch

`MEDIA_RD_BLK_L`, `MEDIA_RD_BLK_F`, `MEDIA_WR_BLK_F`,
`MEDIA_RD_FILEINFO`

Gültig für Module

Storage Rev. A

Beispiel

```

REM #####
REM Sinus-Tabelle mit 3600 Punkten in Daten-Datei speichern
REM #####
#INCLUDE ADwinPRO_ALL.inc

#DEFINE pstom 1           'Adresse des Pro-STORAGE-Modules
#DEFINE f_info DATA_197 'Feld mit Datei-Informationen
#DEFINE LUT DATA_1      'Look-Up-Tabelle für den Sinus
#DEFINE file 1           'Datei-Nummer zum Abspeichern
                           'des Sinus

#DEFINE nds 3600          'Anzahl der Stützstellen
REM Die Länge des Felds mit dem Sinus muss ein Vielfaches
REM von 128 und damit größer oder gleich "nds" sein:
#DEFINE lng 3712          'hier: 29*128
#DEFINE pi2 6.2831853     'Zahlenwert für 2*pi

DIM f_info[22] AS LONG AT DM_LOCAL 'Dimensionierung des Felds
DIM LUT[lng] AS LONG              'Look-Up-Tabelle mit LONGs
DIM sec_p[128] AS LONG            'Sektor für Schreib-Pointer
DIM idx, n, ret AS LONG           'Lokale Variablen
DIM sec_s, sec_e, sec_c, sec_n AS LONG 'Sektor-Variablen
DIM sptr_a, sptr_b AS LONG        'Pointer-Variablen (Sektor-#)

INIT:
PAR_52 = MEDIA_RD_FILEINFO(pstom,f_info) 'Datei-Info holen
REM Überprüfen, ob Medium gesteckt und bereit zum Lesen.
REM Sonst -> EXIT
IF ((PAR_52 AND 1001b) <> 1001b) THEN EXIT

sec_s = f_info[file*2 + 1] 'Erster und ...
sec_e = f_info[file*2 + 2] 'letzter Sektor der Daten-Datei
sec_c = 1                  'Aktueller (rel.) Sektor ist
                           '1. Sektor der Datei
sec_n = SHIFT_RIGHT(nds,7) 'Anz. kompl. Sektoren der
IF ((sec_n*128) < nds) THEN 'Tabelle, ... plus ein Sektor,
    INC sec_n                'falls angefangen
ENDIF
IF ((sec_s+sec_n-1) > sec_e) THEN 'Tab. größer als Datei?
    EXIT                      ' dann EXIT
ENDIF

FOR idx = 1 TO lng          'Werte der Feld-Elemente
                           'berechnen
    IF (idx <= nds) THEN     'Sinus-Tabellen berechnen:
        LUT[idx] = 32767.5 * SIN((idx-1) * pi2 / nds)
    ELSE                    'restl. Elemente ...
        LUT[idx] = 0         'mit 0 auffüllen
    ENDIF
NEXT idx

'Alle Daten-Sektoren schreiben:
FOR n=1 TO sec_n
    REM Alle notwend. Sekt. einzeln schreiben:
    ret = MEDIA_WR_BLK_L(pstom,f_info,file,sec_c,1,LUT,
        (128*n - 127))
    INC sec_c
NEXT n
REM 1. Sektor, in dem WR-Ptr gespeichert wird
sptr_a = f_info[1] + file - 1
REM 10. Sektor, in dem WR-Ptr gespeichert wird
sptr_b = sptr_a + 10
sec_p[1] = nds          '1. LONG im Sektor = Pointer
sec_p[2] = NOT(nds)     '2. LONG im Sektor = /Pointer
FOR idx = 3 TO 128    'Die restlichen 126 LONGs ...

```

```
sec_p[idx] = 0           'mit 0 füllen
NEXT idx

REM Beide Pointer-Sektoren schreiben:
ret = MEDIA_WR_BLK_L(pstom,f_info,0,file,1,sec_p,1)
ret = MEDIA_WR_BLK_L(pstom,f_info,0,file+10,1,sec_p,1)
```

MEDIA_WR_BLK_F kopiert eine Anzahl an Float-Datenblöcken aus einem Feld in eine der zehn Dateien auf dem Datenträger des angegebenen Moduls. Der erste zu beschreibende Sektor der Datei ist frei wählbar.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = MEDIA_WR_BLK_F(module, f_info[], file,
                        sec_idx, sec_cnt, array[], array_idx)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>f_info[]</code>	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
<code>file</code>	Nummer (1...10) der Datei.	LONG
<code>sec_idx</code>	Index (1...m) des ersten zu beschreibenden Sektors, bezogen auf den Startsektor der Datei. Der Max.wert ist abhängig von der Dateigröße.	LONG
<code>sec_cnt</code>	Anzahl (1...12) der zu übertragenden Datenblöcke (=Sektoren) zu 128 Werten.	LONG
<code>array[]</code>	Feld, dessen Daten übertragen werden.	ARRAY FLOAT
<code>array_idx</code>	Index (1...n) des ersten zu übertragenden Feld-elements.	LONG
<code>ret_val</code>	Status der Medienverwaltung als Bitmuster (siehe Tabelle).	LONG

Bitnr.	31:08	07	06	05	04	03	02	01	00
	–	A8	A7	A6	A5	A4	A3	A2	A1

Falls Bit = 1 (Bit = 0: ohne Bedeutung):

- A1 Ein Medium ist im Schacht.
 - A2 Unbekannter Fehler aufgetreten.
 - A3 Medienverwaltung (Glue-Logik) ist beschäftigt.
 - A4 Daten können gelesen werden.
 - A5 Daten sind geschrieben.
 - A6 Reset wird gerade ausgeführt.
 - A7 Dateiende überschritten, es werden keine Daten übertragen.
 - A8 Time out, das Medium hat nicht reagiert.
- Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Bevor diese Anweisung genutzt werden kann, muss das Feld `f_info[]` mit **MEDIA_RD_FILEINFO** initialisiert werden.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LOWINIT**: oder **FINISH**: eines hoch-prioren Prozesses.

Der Start-Sektor ist frei wählbar. Dadurch können Daten an bereits gespeicherte Daten angehängt werden. Achten Sie darauf, nicht mehr Datenblöcke anzugeben als in die Datei passen. Anderenfalls werden die Datenblöcke nicht gespeichert!

MEDIA_WR_BLK_F



Jeder Datenblock enthält 128 Float-Werte und wird in einem Sektor gespeichert. Wenn beispielsweise in einem ersten Schritt n Datenblöcke ab dem Start-Sektor x gespeichert wurden, werden weitere Datenblöcke angehängt, indem sie ab Sektor $x+n+1$ geschrieben werden.

Das Feld `array[]` muss mindestens `array_idx+s_cnt`×128 Werte enthalten.

Siehe auch

[MEDIA_RD_BLK_L](#), [MEDIA_RD_BLK_F](#), [MEDIA_WR_BLK_L](#),
[MEDIA_RD_FILEINFO](#)

Gültig für Module

Storage Rev. A

MEDIA_RD_BLK_L kopiert eine Anzahl an Datenblöcken mit Long-Werten aus einer der zehn Dateien auf dem Datenträger des angegebenen Moduls in ein Feld.

Der erste zu lesende Sektor der Datei ist frei wählbar.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = MEDIA_RD_BLK_L(module,f_info[],file,
                        sec_idx,sec_cnt,array[],array_idx)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>f_info[]</code>	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
<code>file</code>	Nummer (1...10) der Datei.	LONG
<code>sec_idx</code>	Index (1...m) des ersten zu lesenden Sektors, bezogen auf den Startsektor der Datei. Der Max.wert ist abhängig von der Dateigröße.	LONG
<code>sec_cnt</code>	Anzahl (1...12) der zu lesenden Datenblöcke (=Sektoren) zu 128 Long-Werten.	LONG
<code>array[]</code>	Zielfeld, in das die Daten übertragen werden.	ARRAY LONG
<code>array_idx</code>	Index (1...n) des ersten zu beschreibenden Feldelements.	LONG
<code>ret_val</code>	Status der Medienverwaltung als Bitmuster (s.u.).	LONG

Bitnr.	31:08	07	06	05	04	03	02	01	00
	–	A8	A7	A6	A5	A4	A3	A2	A1

Falls Bit = 1 (Bit = 0: ohne Bedeutung):

- A1 Ein Medium ist im Schacht.
 - A2 Unbekannter Fehler aufgetreten.
 - A3 Medienverwaltung (Glue-Logik) ist beschäftigt.
 - A4 Daten können gelesen werden.
 - A5 Daten sind geschrieben.
 - A6 Reset wird gerade ausgeführt.
 - A7 Dateiende überschritten, es werden keine Daten übertragen.
 - A8 Time out, das Medium hat nicht reagiert.
- Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Bevor diese Anweisung genutzt werden kann, muss das Feld `f_info[]` mit **MEDIA_RD_FILEINFO** initialisiert werden.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LOWINIT**: oder **FINISH**: eines hoch-prioren Prozesses.

Der Start-Sektor ist frei wählbar. Achten Sie darauf, nicht mehr Datenblöcke zu lesen als in der Datei vorhanden sind. Anderenfalls werden keine Datenblöcke gelesen!

MEDIA_RD_BLK_L



Jeder Datenblock enthält 128 Long-Werte und ist in einem Sektor gespeichert. Das Zielfeld `array[]` muss daher mindestens `array_idx+s_cnt×128` Feldelemente enthalten.

Siehe auch

[MEDIA_WR_BLK_L](#), [MEDIA_WR_BLK_F](#), [MEDIA_RD_BLK_F](#),
[MEDIA_RD_FILEINFO](#)

Gültig für Module

Storage Rev. A

Beispiel

```

REM #####
REM Sinus-Tabelle mit 3600 Punkten aus Daten-Datei lesen
REM #####
#INCLUDE ADwinPRO_ALL.inc

#DEFINE pstom 1           'Adresse des Pro-STORAGE-Modules
#DEFINE f_info DATA_197  'Feld mit Datei-Informationen
#DEFINE LUT DATA_1       'Look-Up-Tabelle für den Sinus
#DEFINE file 1            'Datei-Nummer zum Lesen
                           'des Sinus

#DEFINE nds 3600          'Anzahl der Stützstellen
REM Die Länge des Felds mit dem Sinus muss ein Vielfaches
REM von 128 und damit größer oder gleich "nds" sein:
#DEFINE lng 3712          'hier: 29*128
#DEFINE pi2 6.2831853     'Zahlenwert für 2*pi

DIM f_info[22] AS LONG AT DM_LOCAL 'Dimensionierung des Felds
DIM LUT[lng] AS LONG              'Look-Up-Tabelle mit LONGs
DIM sec_p[128] AS LONG            'Sektor für Schreib-Pointer
DIM idx, n, ret AS LONG           'Lokale Variablen
DIM sec_c, sec_n AS LONG          'Sektor-Variablen
DIM dzn, rmn AS LONG              'Blöcke zu 12 Sek., restl. Werte

INIT:
  PAR_52 = MEDIA_RD_FILEINFO(pstom,f_info) 'Datei-Info holen
  REM Überprüfen, ob Medium gesteckt und bereit zum Lesen.
  REM Sonst -> EXIT
  IF ((PAR_52 AND 1001b) <> 1001b) THEN EXIT

  REM Länge der gesp. Tabelle ermitteln - dazu 1. Pointer
                           'Sektor laden:
  ret = MEDIA_RD_BLK_L(pstom,f_info,(file-1),1,1,sec_p,1)
  IF ((ret AND 22h) > 0) THEN EXIT 'Exit bei Reset o. Error

  REM 1. Daten-Pointer auf Gültigkeit überprüfen
  IF ((sec_p[1] XOR sec_p[2]) <> -1) THEN

    REM 1. Daten-Ptr. ist ungültig -> 2. Daten-Pointer auswerten
    ret = MEDIA_RD_BLK_L(pstom,f_info,(file-1),11,1,sec_p,1)
    IF ((ret AND 22h) > 0) THEN EXIT 'Exit bei RESET o. ERROR

    REM 2. Daten-Pointer auf Gültigkeit überprüfen
    IF ((sec_p[1] XOR sec_p[2]) <> -1) THEN
      REM 2. Daten-Ptr. ebenfalls ungültig: Exit
      EXIT
    ELSE
      nds = sec_p[1]           '2. Daten-Pointer übernehmen
    ENDIF
  ELSE
    nds = sec_p[1]           '1. Daten-Pointer übernehmen
  ENDIF

  REM Anzahl zu ladender Sektoren und Pakete (á 12 Sektoren)
                           'ermitteln:
  dzn = nds/1536            'Anz. von Paketen á 12 Sektoren
  rmn = nds - dzn*1536      'Anz. restl. LONGs errechnen
  sec_n = SHIFT_RIGHT(rmn,7) 'Anz. weitere zu ladende
                           'Sektoren ...

  IF ((128*sec_n) < rmn) THEN
    REM Sektor war angefangen: plus eins
    INC sec_n
  ENDIF

  REM Sektoren der Daten-Datei ins Feld kopieren

```

```
IF (dzn>0) THEN                                'Pakete á 12 Sektoren vorhanden?
FOR n=1 TO dzn                                  'Alle Pakete kopieren ...
    sec_c = 12*n - 11                            'aktuellen Sektor errechnen
    idx = 1536*n - 1535                          'Pointer im Ziel-Array errechnen
    REM 12 Sektoren lesen
    ret = MEDIA_RD_BLK_L(pstom,f_info,file,sec_c,12,LUT,idx)
    IF ((ret AND 22h) > 0) THEN EXIT 'Exit bei Reset o. Error
NEXT n
ENDIF

REM restliche Sektoren kopieren
ret = MEDIA_RD_BLK_L(pstom,f_info,file,(12*dzn+1),sec
n,LUT,(1536*dzn+1))
IF ((ret AND 22h) > 0) THEN EXIT 'Exit bei Reset o. Error
```

MEDIA_RD_BLK_F kopiert eine Anzahl an Datenblöcken mit Float-Werten aus einer der zehn Dateien auf dem Datenträger des angegebenen Moduls in ein Feld.

Der erste zu lesende Sektor der Datei ist frei wählbar.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = MEDIA_RD_BLK_F(module, f_info[], file,
                        sec_idx, sec_cnt, array[], array_idx)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>f_info[]</code>	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
<code>file</code>	Nummer (1...10) der Datei.	LONG
<code>sec_idx</code>	Index (1...m) des ersten zu lesenden Sektors, bezogen auf den Startsektor der Datei. Der Max.wert ist abhängig von der Dateigröße.	LONG
<code>sec_cnt</code>	Anzahl (1...12) der zu lesenden Datenblöcke (=Sektoren) zu 128 Long-Werten.	LONG
<code>array[]</code>	Zielfeld, in das die Daten übertragen werden.	ARRAY FLOAT
<code>array_idx</code>	Index (1...n) des ersten zu beschreibenden Feldelements.	LONG
<code>ret_val</code>	Status der Medienverwaltung als Bitmuster (s. u.).	LONG

Bitnr.	31:08	07	06	05	04	03	02	01	00
	–	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁

Falls Bit = 1 (Bit = 0: ohne Bedeutung):

- A₁ Ein Medium ist im Schacht.
 - A₂ Unbekannter Fehler aufgetreten.
 - A₃ Medienverwaltung (Glue-Logik) ist beschäftigt.
 - A₄ Daten können gelesen werden.
 - A₅ Daten sind geschrieben.
 - A₆ Reset wird gerade ausgeführt.
 - A₇ Dateiende überschritten, es werden keine Daten übertragen.
 - A₈ Time out, das Medium hat nicht reagiert.
- Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Bevor diese Anweisung genutzt werden kann, muss das Feld `f_info[]` mit **MEDIA_RD_FILEINFO** initialisiert werden.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LOWINIT**: oder **FINISH**: eines hochprioren Prozesses.

Der Start-Sektor ist frei wählbar. Achten Sie darauf, nicht mehr Datenblöcke zu lesen als in der Datei vorhanden sind. Anderenfalls werden keine Datenblöcke gelesen!

MEDIA_RD_BLK_F



Jeder Datenblock enthält 128 Float-Werte und ist in einem Sektor gespeichert. Das Zielfeld `array[]` muss daher mindestens `array_idx+s_cnt×128` Feldelemente enthalten.

Siehe auch

[MEDIA_WR_BLK_L](#), [MEDIA_WR_BLK_F](#), [MEDIA_RD_BLK_L](#),
[MEDIA_RD_FILEINFO](#)

Gültig für Module

Storage Rev. A

MEDIA_RD_FILEINFO initialisiert die Mediumverwaltung (Glue-Logik) auf dem angegebenen Modul und gibt die Datei-Informationen (Start- und Endsektor) in einem Feld zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = MEDIA_RD_FILEINFO(module, f_info[])
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
f_info[]	Feld, in dem die Nummern der Start- und End-Sektoren aller Dateien enthalten sind.	ARRAY LONG
ret_val	Status der Medienverwaltung als Bitmuster (siehe Tabelle).	LONG

Bitnr.	31:06	05	04	03	02	01	00
	–	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁

Falls Bit = 1 (Bit = 0: ohne Bedeutung):

- A₁ Ein Medium ist im Schacht.
 - A₂ Unbekannter Fehler aufgetreten.
 - A₃ Medienverwaltung (Glue-Logik) ist beschäftigt.
 - A₄ Daten können gelesen werden.
 - A₅ Daten sind geschrieben.
 - A₆ Reset wird gerade ausgeführt.
- Andere Bits können gesetzt sein, sind aber hier nicht verwertbar.

Bemerkungen

Das Feld `f_info[]` muss mit **MEDIA_RD_FILEINFO** initialisiert werden, bevor Daten auf das Medium geschrieben oder von dort gelesen werden können. Das Feld muss mindestens 22 Elemente groß sein.

Die Anweisung darf nur in einem niedrig-prioren Prozessabschnitt aufgerufen werden:

- an beliebiger Stelle in einem niedrig-prioren Prozess.
- in den Abschnitten **LOWINIT**: oder **FINISH**: eines hochprioren Prozesses.

Die Feldelemente mit ungeradem Index enthalten jeweils den Startsektor einer Datei, diejenigen mit geradem Index die Endsektoren. Die folgende Tabelle zeigt die Zuordnung zwischen den Indexnummern des Felds und den Dateien.

Indexnr.	Datei
1, 2	Fileinfo.dat
3, 4	ADWIN1.dat
...	...
21, 22	ADWIN10.dat

Siehe auch

[MEDIA_WR_BLK_L](#), [MEDIA_WR_BLK_F](#), [MEDIA_RD_BLK_L](#), [MEDIA_RD_BLK_F](#)

Gültig für Module

Storage Rev. A

MEDIA_RD_FILE- INFO



Beispiel

```

REM #####
REM Auslesen und Interpretation der Datei <FILEINFO.DAT>
REM #####
REM PAR_1 ... PAR_10: Erster Sektor der Datei 1 ... 10
REM PAR_11 ... PAR_20: Letzter Sektor der Datei 1 ... 10
REM PAR_21 ... PAR_30: Anzahl Sektoren der Datei 1 ... 10
REM PAR_31 ... PAR_40: Anzahl LONGs/FLOATs der Datei 1 ... 10
REM #####
#INCLUDE ADwinPRO_ALL.inc

#DEFINE pstom 1                'Adresse des Pro-STORAGE-Modules
#DEFINE f_info DATA_197      'Feld mit Datei-Informationen

DIM f_info[22] AS LONG AT DM_LOCAL 'Dimensionierung des Felds
DIM n AS LONG                 'lokale Variable

INIT:
  PAR_52 = MEDIA_RD_FILEINFO(pstom,f_info) 'Datei-Info holen
  REM Überprüfen, ob Medium gesteckt und bereit zum Lesen.
  REM Sonst -> EXIT
  IF ((PAR_52 AND 1001b) <> 1001b) THEN EXIT

  REM alle 10 Dateien lesen
  FOR n = 1 TO 10
    REM Nummer des ersten und letzten Dateisektors berechnen
    PAR[n] = f_info[n*2 + 1]
    PAR[n+10] = f_info[n*2 + 2]
    REM Dateilänge > 1 Sektor?
    IF ((PAR[n+10] - PAR[n]) <> 0) THEN
      PAR[n+20] = PAR[n+10] - PAR[n] + 1 'Anz. Sektoren
      PAR[n+30] = PAR[n+20] * 128 'Anz. Datenwerte
    ENDIF
  NEXT n

EXIT

```


3.5 Pro I: Spezielle Module

Dieser Abschnitt beschreibt Befehle zum Ansprechen spezieller *ADwin-Pro*-Module. Um die Befehle zu verwenden, fügen Sie diese Zeile in Ihr *ADbasic*-Programm ein:

```
#INCLUDE ADwinPRO_ALL.inc
```

Auf den folgenden Seiten werden die Befehle ausführlich erklärt. Zusätzlich ist zu jeder Funktion ein kleines Anwendungsbeispiel aufgeführt. Die Befehle sind nach Schnittstellentyp sortiert:

- [Thermo-Module, Seite 165](#)
- [CAN-Bus, Seite 181](#)
- [Feldbus, Seite 197](#)
- [RSxxx, Seite 208](#)
- [LS-Bus + Pro I, Seite 218](#)

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

3.5.1 Thermo-Module

PT100_DIG_TO_TEMP

PT100_DIG_TO_TEMP berechnet aus dem Digitalwert eines PT100-Fühlers die zugehörige Temperatur in Celsius oder Fahrenheit.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = PT100_DIG_TO_TEMP(dig_val, t_unit)
```

Parameter

dig_val	Digitalwert (0...65535).	LONG
t_unit	Temperatur-Einheit: 1: Grad Celsius. 2: Grad Fahrenheit.	LONG
ret_val	Temperatur in Grad Celsius oder in Grad Fahrenheit.	FLOAT

Bemerkungen

Für die Ermittlung der Temperaturwerte in °C und °F aus der Thermospannung wird die Grundwertreihe der IEC 751 (= EN 60751: 1990) verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Alternativ kann die Temperatur von Hand mit einer Konvertierungstabelle berechnet werden: Thermoelement-Verstärker PT100-x.

Sie finden die Konvertierungstabellen in der Online-Hilfe von *ADbasic*, Menüpunkt „Hardware-Informationen“ (unter Contents) oder auch im Datenblatt des Herstellers: Analog Devices.

Siehe auch

[TC_SELECT](#), [PT100_DIG_TO_R](#)

Gültig für Module

PT100-4, PT100-8

Beispiel

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des PT100-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

```
#INCLUDE ADWINPRO_ALL.INC  
DIM temp[8] AS FLOAT  
DIM channel AS LONG  
  
INIT:  
    PROCESSDELAY=100000  
  
EVENT:  
    FOR channel = 1 TO 8  
        TC_SELECT(1,channel)    'nächsten Kanal wählen  
        P1_SLEEP(500)           'Einschwingzeit abwarten  
        REM Thermospannung lesen und in °C umwandeln  
        temp[channel] = PT100_DIG_TO_TEMP(ADC(1,1),1)  
    NEXT
```

PT100_DIG_TO_R berechnet aus dem Digitalwert eines PT100-Fühlers den zugehörigen Widerstand in Ohm.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = PT100_DIG_TO_R(dig_val)
```

Parameter

dig_val	Digitalwert (0...65535).	LONG
ret_val	Widerstand in Ohm.	FLOAT

Bemerkungen

Für die Ermittlung der Widerstandswerte aus der Thermospannung wird folgende Formel verwendet:

$$\text{ret_val} = ((\text{dig_val} - 32768) \cdot 200 / 65536) + 100\Omega$$

Siehe auch

[TC_SELECT](#), [PT100_DIG_TO_TEMP](#)

Gültig für Module

PT100-4, PT100-8

Beispiel

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des Thermoelement-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

```
#INCLUDE ADWINPRO_ALL.INC
DIM temp[8] AS FLOAT
DIM channel AS LONG

INIT:
    PROCESSDELAY=100000

EVENT:
    FOR channel = 1 TO 8
        TC_SELECT(1,channel)    'nächsten Kanal wählen
        P1_SLEEP(500)           'Einschwingzeit abwarten
        REM Thermospannung lesen und in °C umwandeln
        temp[channel] = PT100_DIG_TO_R(ADC(1,1),1)
    NEXT
```

PT100_DIG_TO_R

TC_SELECT

TC_SELECT schaltet den angegebenen Thermoelement-Kanal über Multiplexer auf den analogen Ausgang des Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
TC_SELECT(module, channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Kanalnummer: TC-4, PT100-4: 1...4. TC-8, PT100-8: 1...8. TC-16: 1...16.	LONG

Bemerkungen



Die Einschwingzeit des Multiplexers muss durch entsprechende Programmierung überbrückt werden, damit ein korrekter Messwert erreicht wird. Die Einschwingzeit ist im Handbuch Pro-Hardware angegeben.

Zur Umrechnung der gemessenen Spannung in die Einheit [°C] wird die Konvertierungstabelle des Thermoelement-Verstärkers benötigt:

- TC-x Typ J (AD594)
- TC-x Typ K (AD595) oder
- PT100-x.

Sie finden die Konvertierungstabellen in der Online-Hilfe von *ADbasic*, Menüpunkt Hardware-Informationen (unter Contents) oder auch im Datenblatt des Herstellers: Analog Devices.

Siehe auch

[PT100_DIG_TO_TEMP](#), [PT100_DIG_TO_R](#), [TCJ_DIG_TO_TEMP](#),
[TCK_DIG_TO_TEMP](#)

Gültig für Module

PT100-4, PT100-8, TC-16, TC-4, TC-8

Beispiel

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des Thermoelement-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

```
#INCLUDE ADWINPRO_ALL.INC
DIM value, i AS LONG

INIT:
    value = 0          'Variablen...
    i = 1              ' initialisieren

EVENT:
    TC_SELECT(1,3)      'Thermoelement-Kanal 3 wählen
    REM Fügen Sie Programmcode ein, damit die Temperaturmessung
    REM erst nach der Einschwingzeit des Multiplexers erfolgt.
    REM Der nachfolgende Befehl ADC benötigt bereits einen Teil
    REM dieser Zeit.
    value = value + ADC(1,1) 'Wert messen
    IF (i = 10) THEN
        PAR_1 = value / 10    'Mittelwert aus 10 Messungen
        value = 0
        i = 0
    ENDIF
    INC i
```

TCJ_DIG_TO_TEMP

TCJ_DIG_TO_TEMP berechnet aus einem Digitalwert eines Thermoelements vom Typ J die zugehörige Temperatur in Celsius oder Fahrenheit.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = TCJ_DIG_TO_TEMP(dig_val, t_unit)
```

Parameter

dig_val	Digitalwert (0...65535).	LONG
t_unit	Temperatur-Einheit: 1: Grad Celsius. 2: Grad Fahrenheit.	LONG
ret_val	Temperatur in Grad Celsius oder in Grad Fahrenheit.	FLOAT

Bemerkungen

Für die Ermittlung der Temperaturwerte in °C und °F aus der Thermo-spannung wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Alternativ kann die Temperatur von Hand mit einer Konvertierungstabelle berechnet werden: Thermoelement-Verstärker TC-x Typ J (AD594).

Sie finden die Konvertierungstabellen in der Online-Hilfe von *ADbasic*, Menüpunkt „Hardware-Informationen“ (unter Contents) oder auch im Datenblatt des Herstellers: Analog Devices.

Siehe auch

[TC_SELECT](#), [TCK_DIG_TO_TEMP](#)

Gültig für Module

TC-16, TC-4, TC-8

Beispiel

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des Thermoelement-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

```
#INCLUDE ADWINPRO_ALL.INC  
DIM temp[8] AS FLOAT  
DIM channel AS LONG  
  
INIT:  
    PROCESSDELAY=100000  
  
EVENT:  
    FOR channel = 1 TO 8  
        TC_SELECT(1,channel)    'nächsten Kanal wählen  
        P1_SLEEP(500)           'Einschwingzeit abwarten  
        REM Thermospannung lesen und in °C umwandeln  
        temp[channel] = TCJ_DIG_TO_TEMP(ADC(1,1),1)  
    NEXT
```

TCK_DIG_TO_TEMP berechnet aus einem Digitalwert eines Thermoelements vom Typ K die zugehörige Temperatur in Celsius oder Fahrenheit.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = TCK_DIG_TO_TEMP(dig_val, t_unit)
```

Parameter

dig_val	Digitalwert (0...65535).	LONG
t_unit	Temperatur-Einheit: 1: Grad Celsius. 2: Grad Fahrenheit.	LONG
ret_val	Temperatur in Grad Celsius oder in Grad Fahrenheit.	FLOAT

Bemerkungen

Für die Ermittlung der Temperaturwerte in °C und °F aus der Thermo-spannung wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Alternativ kann die Temperatur von Hand mit einer Konvertierungstabelle berechnet werden: Thermoelement-Verstärker TC-x Typ K (AD595).

Sie finden die Konvertierungstabellen in der Online-Hilfe von *ADbasic*, Menüpunkt „Hardware-Informationen“ (unter Contents) oder auch im Datenblatt des Herstellers: Analog Devices.

Siehe auch

[TC_SELECT](#), [TCJ_DIG_TO_TEMP](#)

Gültig für Module

TC-16, TC-4, TC-8

Beispiel

Im Beispiel wird davon ausgegangen, dass der analoge Ausgang des Thermoelement-Moduls mit dem analogen Eingang 1 eines A/D-Moduls mit Moduladresse 1 verbunden ist.

```
#INCLUDE ADWINPRO_ALL.INC
DIM temp[8] AS FLOAT
DIM channel AS LONG

INIT:
    PROCESSDELAY=100000

EVENT:
    FOR channel = 1 TO 8
        TC_SELECT(1,channel)    'nächsten Kanal wählen
        P1_SLEEP(500)           'Einschwingzeit abwarten
        REM Thermospannung lesen und in °C umwandeln
        temp[channel] = TCK_DIG_TO_TEMP(ADC(1,1),1)
    NEXT
```

TCK_DIG_TO_TEMP

TC_READ_B

TC_READ_B gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ B.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
ret_val = TC_READ_B(module, channel, ret_type)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1...8) des Kanals.	LONG
<code>ret_type</code>	Typ des Rückgabewerts <code>ret_val</code> : 0: Thermospannung in μV . 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
<code>ret_val</code>	Messwert des Kanals, abhängig von <code>ret_type</code> : Thermospannung: 291 μV ... 13820 μV . Temperatur in $^{\circ}\text{C}$: 250 $^{\circ}\text{C}$... 1820 $^{\circ}\text{C}$. Temperatur in $^{\circ}\text{F}$: 482 $^{\circ}\text{F}$... 3329,6 $^{\circ}\text{F}$.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtast-rate siehe **TC_SET_RATE**). Die Anweisung **TC_READ_B** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

[TC_READ_E](#), [TC_READ_J](#), [TC_READ_K](#), [TC_READ_N](#), [TC_READ_R](#), [TC_READ_S](#), [TC_READ_T](#), [TC_SET_RATE](#)

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC  
  
EVENT:  
REM Read temperature in  $^{\circ}\text{C}$  from channel 5  
FPAR_1 = TC_READ_B(1,5,1)
```


TC_READ_E gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ E.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = TC_READ_E(module, channel, ret_type)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer (1...8) des Kanals.	LONG
ret_type	Typ des Rückgabewerts <code>ret_val</code> : 0: Thermospannung in μV . 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
ret_val	Messwert des Kanals, abhängig von <code>ret_type</code> : Thermospannung: -8825 μV ... 76373 μV . Temperatur in $^{\circ}\text{C}$: -200 $^{\circ}\text{C}$... 1000 $^{\circ}\text{C}$. Temperatur in $^{\circ}\text{F}$: -328 $^{\circ}\text{F}$... 1832 $^{\circ}\text{F}$.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtast-rate siehe **TC_SET_RATE**). Die Anweisung **TC_READ_E** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

[TC_READ_B](#), [TC_READ_J](#), [TC_READ_K](#), [TC_READ_N](#), [TC_READ_R](#), [TC_READ_S](#), [TC_READ_T](#), [TC_SET_RATE](#)

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC
```

EVENT:

```
REM Read temperature in  $^{\circ}\text{C}$  from channel 5
FPAR_1 = TC_READ_E(1,5,1)
```

TC_READ_E

TC_READ_J

TC_READ_J gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ J.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
ret_val = TC_READ_J(module, channel, ret_type)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1...8) des Kanals.	LONG
<code>ret_type</code>	Typ des Rückgabewerts <code>ret_val</code> : 0: Thermospannung in μV . 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
<code>ret_val</code>	Messwert des Kanals, abhängig von <code>ret_type</code> : Thermospannung: $-8095\mu\text{V}$... $69553\mu\text{V}$. Temperatur in $^{\circ}\text{C}$: -210°C ... 1200°C . Temperatur in $^{\circ}\text{F}$: -346°F ... 2192°F .	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtast-rate siehe **TC_SET_RATE**). Die Anweisung **TC_READ_J** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

[TC_READ_B](#), [TC_READ_E](#), [TC_READ_K](#), [TC_READ_N](#), [TC_READ_R](#), [TC_READ_S](#), [TC_READ_T](#), [TC_SET_RATE](#)

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC  
  
EVENT:  
REM Read temperature in  $^{\circ}\text{C}$  from channel 5  
FPAR_1 = TC_READ_J(1,5,1)
```

TC_READ_K gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ K.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = TC_READ_K(module, channel, ret_type)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1...8) des Kanals.	LONG
<code>ret_type</code>	Typ des Rückgabewerts <code>ret_val</code> : 0: Thermospannung in μV . 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
<code>ret_val</code>	Messwert des Kanals, abhängig von <code>ret_type</code> : Thermospannung: $-5891\mu\text{V}$... $54886\mu\text{V}$. Temperatur in $^{\circ}\text{C}$: -200°C ... 1372°C . Temperatur in $^{\circ}\text{F}$: -328°F ... $2501,6^{\circ}\text{F}$.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtast-rate siehe **TC_SET_RATE**). Die Anweisung **TC_READ_K** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

[TC_READ_B](#), [TC_READ_E](#), [TC_READ_J](#), [TC_READ_N](#), [TC_READ_R](#), [TC_READ_S](#), [TC_READ_T](#), [TC_SET_RATE](#)

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

EVENT:
REM Read temperature in  $^{\circ}\text{C}$  from channel 5
FPAR_1 = TC_READ_K(1,5,1)
```

TC_READ_K

TC_READ_N

TC_READ_N gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ N.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
ret_val = TC_READ_N(module, channel, ret_type)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer (1...8) des Kanals.	LONG
ret_type	Typ des Rückgabewerts <code>ret_val</code> : 0: Thermospannung in μV . 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
ret_val	Messwert des Kanals, abhängig von <code>ret_type</code> : Thermospannung: $-3990\mu\text{V}$... $47513\mu\text{V}$. Temperatur in $^{\circ}\text{C}$: -200°C ... 1300°C . Temperatur in $^{\circ}\text{F}$: -328°F ... 2372°F .	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtast-rate siehe **TC_SET_RATE**). Die Anweisung **TC_READ_N** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

[TC_READ_B](#), [TC_READ_E](#), [TC_READ_J](#), [TC_READ_K](#), [TC_READ_R](#), [TC_READ_S](#), [TC_READ_T](#), [TC_SET_RATE](#)

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
  
EVENT:  
REM Read temperature in  $^{\circ}\text{C}$  from channel 5  
FPAR_1 = TC_READ_N(1,5,1)
```

TC_READ_R gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ R.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = TC_READ_R(module, channel, ret_type)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1...8) des Kanals.	LONG
<code>ret_type</code>	Typ des Rückgabewerts <code>ret_val</code> : 0: Thermospannung in μV . 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
<code>ret_val</code>	Messwert des Kanals, abhängig von <code>ret_type</code> : Thermospannung: -226 μV ... 21101 μV . Temperatur in $^{\circ}\text{C}$: -50 $^{\circ}\text{C}$... 1768 $^{\circ}\text{C}$. Temperatur in $^{\circ}\text{F}$: -58 $^{\circ}\text{F}$... 3214,4 $^{\circ}\text{F}$.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtast-rate siehe **TC_SET_RATE**). Die Anweisung **TC_READ_R** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

[TC_READ_B](#), [TC_READ_E](#), [TC_READ_J](#), [TC_READ_K](#), [TC_READ_N](#), [TC_READ_S](#), [TC_READ_T](#), [TC_SET_RATE](#)

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

EVENT:
REM Read temperature in  $^{\circ}\text{C}$  from channel 5
FPAR_1 = TC_READ_R(1,5,1)
```

TC_READ_R

TC_READ_S

TC_READ_S gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ S.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
ret_val = TC_READ_S(module, channel, ret_type)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1...8) des Kanals.	LONG
<code>ret_type</code>	Typ des Rückgabewerts <code>ret_val</code> : 0: Thermospannung in μV . 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
<code>ret_val</code>	Messwert des Kanals, abhängig von <code>ret_type</code> : Thermospannung: $-236\mu\text{V}$... $18693\mu\text{V}$. Temperatur in $^{\circ}\text{C}$: -50°C ... 1768°C . Temperatur in $^{\circ}\text{F}$: -58°F ... $3214,4^{\circ}\text{F}$.	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtast-rate siehe **TC_SET_RATE**). Die Anweisung **TC_READ_S** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

[TC_READ_B](#), [TC_READ_E](#), [TC_READ_J](#), [TC_READ_K](#), [TC_READ_N](#), [TC_READ_R](#), [TC_READ_T](#), [TC_SET_RATE](#)

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
  
EVENT:  
REM Read temperature in  $^{\circ}\text{C}$  from channel 5  
FPAR_1 = TC_READ_S(1,5,1)
```

TC_READ_T gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.

Der Temperaturwert gilt nur für Temperaturfühler vom Typ T.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = TC_READ_T(module, channel, ret_type)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1...8) des Kanals.	LONG
<code>ret_type</code>	Typ des Rückgabewerts <code>ret_val</code> : 0: Thermospannung in μV . 1: Temperatur in $^{\circ}\text{C}$. 2: Temperatur in $^{\circ}\text{F}$.	LONG
<code>ret_val</code>	Messwert des Kanals, abhängig von <code>ret_type</code> : Thermospannung: $-5603\mu\text{V}$... $20872\mu\text{V}$. Temperatur in $^{\circ}\text{C}$: -200°C ... 400°C . Temperatur in $^{\circ}\text{F}$: -454°F ... 752°F .	FLOAT

Bemerkungen

Das Modul tastet die Temperatur regelmäßig ab (Einstellen der Abtast-rate siehe [TC_SET_RATE](#)). Die Anweisung **TC_READ_T** gibt den jeweils zuletzt ermittelten Temperaturwert zurück.

Für die Ermittlung der Thermospannung und der Temperaturwerte in $^{\circ}\text{C}$ und $^{\circ}\text{F}$ wird die Grundwertreihe der IEC 584-1 verwendet. Der Messwert ist deswegen nur für Temperaturfühler richtig, die dieser Norm entsprechen.

Siehe auch

[TC_READ_B](#), [TC_READ_E](#), [TC_READ_J](#), [TC_READ_K](#), [TC_READ_N](#), [TC_READ_R](#), [TC_READ_S](#), [TC_SET_RATE](#)

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

EVENT:
REM Temperatur in  $^{\circ}\text{C}$  am Kanal 5 abfragen
FPAR_1 = TC_READ_T(1,5,1)
```

TC_READ_T

TC_SET_RATE

TC_SET_RATE stellt die Abtastrate für das angegebene Modul ein.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
TC_SET_RATE(module, rate)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
rate	Kennzahl für die gewählte Abtastrate (siehe Tabelle); Voreinstellung: 15.	LONG

Kennzahl	Abtastrate [Hz]	ADC-Rauschen [nV]
1	3520	23000
2	1760	3500
3	880	2000
4	440	1400
5	220	1000
6	110	750
7	55	510
8	27,5	375
9	13,75	250
15	6,875	200

Bemerkungen

Die Abtastrate gilt für alle Kanäle gleichermaßen.

Mit steigender Abtastrate steigt das Rauschsignal, das am ADC eines Kanals entsteht und das das eintreffende Signal überlagert (siehe Tabelle).

Siehe auch

TC_READ_B, TC_READ_E, TC_READ_J, TC_READ_K, TC_READ_N, TC_READ_R, TC_READ_S, TC_READ_T

Gültig für Module

TC-8-ISO

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
INIT:
```

```
REM Abtastrate auf 27,5 Hz einstellen
```

```
TC_SET_RATE(1,8)
```


`CAN_MSG` ist ein eindimensionales Feld bestehend aus 9 Elementen, in dem die Message-Objekte (Nachrichten) des CAN-Busses beim Senden und Empfangen gespeichert sind oder werden.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
CAN_MSG[n] = value
```

oder

```
ret_val = CAN_MSG[n]
```

Parameter

<code>n</code>	Nummer eines Feldelements (1... 9).	LONG
<code>value</code>	Berechnungs-Ausdruck, dessen Wert (0...256) in das Message-Objekt geschrieben wird.	LONG
<code>ret_val</code>	Wert (0...256), der aus dem Message-Objekt gelesen wird.	LONG

Bemerkungen

Die ersten 8 Elemente des Felds enthalten die Datenbytes 1...8, das 9. Feldelement enthält die Anzahl der gültigen Datenbytes der Nachricht. Hier muss ein Wert von 0 bis 8 eingetragen werden.

Die Datenbytes belegen in den Feldelementen nur die Bits 7...0, die Bits 31...8 werden ignoriert.

Die Werte im Feld `CAN_MSG[]` müssen vor der Ausführung des Befehls **TRANSMIT** eingetragen werden.

Siehe auch

[EN_RECEIVE](#), [EN_TRANSMIT](#), [READ_MSG](#), [TRANSMIT](#)

Gültig für Module

CAN-1, CAN-2

CAN_MSG

Beispiel

REM Sendet eine 32 Bit-FLOAT-Zahl (hier: Pi) als Folge von
REM 4 Bytes in einem Message-Objekt
REM (Empfangen einer Fließkomma-Zahl siehe Bsp. bei [READ_MSG](#))

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE pi 3.14159265
DIM i AS LONG

INIT:
  INIT_CAN(1,1)           'CAN-Controller initialisieren

  REM Initialisiere das Message-Objekt 6
  REM zum Senden von CAN-Nachrichten mit dem Identifier 40
  EN_TRANSMIT(1,1,6,40,0)

  REM Bitmuster von Pi mit Datenformat Long erzeugen
  PAR_1 = CAST_FLOATTOLONG(pi)

  REM Bitmuster (32 Bit) in 4 Bytes aufteilen
  CAN_MSG[4] = PAR_1 AND 0FFh 'LSB zuweisen
  FOR i = 1 TO 3
    CAN_MSG[4-i] = SHIFT_RIGHT(PAR_1,8*i) AND 0FFh
  NEXT i
  CAN_MSG[9] = 4           'Länge der Nachricht in Bytes

EVENT:
  TRANSMIT(1,1,6)          'Message-Objekt 6 senden
```

EN_INTERRUPT konfiguriert ein Message-Objekt des angegebenen Moduls, dass es bei Eintreffen einer Nachricht einen Event-Signal (Interrupt) erzeugt.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

EN_INTERRUPT (module, channel, objectno)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
objectno	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 **EVENTENABLE** freigegeben ist. Konfigurieren Sie zuerst alle gewünschten Message-Objekte und geben Sie erst anschließend das Event-Signal frei.

In einem System darf – zusätzlich zum Prozessor-Modul – nur ein einziger Event-Eingang aktiv sein, d.h. Sie müssen einen eventuell aktiven Event-Eingang sperren, bevor Sie den Event-Eingang an einem anderen Modul aktivieren.

Siehe auch

[EN_RECEIVE](#), [EVENTENABLE](#), [GET_CAN_REG](#), [INIT_CAN](#)

Gültig für Module

CAN-1, CAN-2

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
INIT:
    REM CAN-Controller 1 auf dem CAN-Modul 1 initialisieren
    INIT_CAN(1,1)
    REM Message-Objekte 3 und 15 zum Lesen konfigurieren
    EN_RECEIVE(1,1,3,1,0)
    EN_RECEIVE(1,1,15,385,0)
    REM Interrupt für Message-Objekte 3 und 15 konfigurieren
    EN_INTERRUPT(1,1,3)
    EN_INTERRUPT(1,1,15)
    REM Event-Signal freigegeben
    EVENTENABLE(1,ext,1)
EVENT:
    REM Interruptregister lesen und Wert in Objektnr. wandeln (s.u.)
    PAR_13 = GET_CAN_REG(1,1,5Fh)
    IF (PAR_13 = 2) THEN
        PAR_13 = 15
    ELSE
        PAR_13 = PAR_13 - 2
    ENDIF
```

Der Wert im Interrupt-Register 5Fh entspricht einem der Message-Objekte nach folgendem Schema:

Wert	2	3	4	...	16
Nummer Message-Objekt	15	1	2	...	14

EN_INTERRUPT



EN_RECEIVE

EN_RECEIVE gibt ein Message-Objekt für den Empfang von Nachrichten auf dem angegebenen Modul frei.

Für das Message-Objekt werden der CAN-Kanal, die Länge des Nachrichten-Identifiers und der Identifier selbst festgelegt.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
EN_RECEIVE (module, channel, objectno, id, extend)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
<code>objectno</code>	Nummer (1...15) des Message-Objektes im CAN-Controller.	LONG
<code>id</code>	Identifier der Nachrichten, die in diesem Message-Objekt empfangen werden sollen ($0 \dots 2^{11}$ oder $0 \dots 2^{29}$).	LONG
<code>extend</code>	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 es zuvor mit **EN_RECEIVE** freigegeben wurde.

Siehe auch

[CAN_MSG](#), [EN_TRANSMIT](#), [INIT_CAN](#), [READ_MSG](#), [TRANSMIT](#)

Gültig für Module

CAN-1, CAN-2

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
REM Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
INIT_CAN (1,1)
REM Message-Objekt 1 freigegeben für den Empfang von
REM CAN-Nachrichten mit dem 11 Bit-Identifier 200
EN_RECEIVE (1,1,1,200,0)
```

EN_TRANSMIT gibt ein Message-Objekt für das Senden von Nachrichten auf dem angegebenen Modul frei.

Für das Message-Objekt werden der CAN-Kanal, die Länge des Nachrichten-Identifiers und der Identifier selbst festgelegt.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

EN_TRANSMIT (module, channel, objectno, id, extend)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
<code>objectno</code>	Nummer (1...14) des Message-Objektes im CAN-Controller.	LONG
<code>id</code>	Identifier der Nachrichten, die in diesem Message-Objekt gesendet werden sollen ($0 \dots 2^{11}$ oder $0 \dots 2^{29}$).	LONG
<code>extend</code>	Merker für die Länge des Identifiers: 0: 11 Bit Identifier. 1: 29 Bit Identifier.	LONG

Bemerkungen

Ein Message-Objekt kann nur Nachrichten auf dem CAN-Bus senden, wenn es zuvor mit **EN_TRANSMIT** freigegeben wurde.

Siehe auch

[CAN_MSG](#), [EN_RECEIVE](#), [INIT_CAN](#), [READ_MSG](#), [TRANSMIT](#)

Gültig für Module

CAN-1, CAN-2

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
INIT:
    REM Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
    INIT_CAN(1,1)
    REM Message-Objekt 6 freigeben für das Senden von
    REM CAN-Nachrichten mit dem 11 Bit-Identifier 40
    EN_TRANSMIT(1,1,6,40,0)
```

EN_TRANSMIT

GET_CAN_REG

GET_CAN_REG gibt den Inhalt eines bestimmten Registers auf einem CAN-Controller des angegebenen Moduls zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = GET_CAN_REG(module, channel, regno)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
regno	Register-Nummer des CAN-Controllers (0...255).	LONG
ret_val	Inhalt des CAN-Controller-Registers.	LONG

Bemerkungen

Die anzugebende Registernummer entspricht der Registernummer des CAN-Controllers (siehe Datenblatt AN82527 von Intel®), z. B.:

- Adresse 00h: Kontroll-Register
- Adresse 01h: Status-Register
- Adresse 5fh: Interrupt-Register

Siehe auch

[EN_INTERRUPT](#), [INIT_CAN](#), [SET_CAN_BAUDRATE](#), [SET_CAN_REG](#)

Gültig für Module

CAN-1, CAN-2

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
INIT:  
    REM Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1  
    INIT_CAN(1,1)  
    REM Das Kontroll-Register des CAN-Controller 1, Modul 1 auslesen  
    PAR_1 = GET_CAN_REG(1,1,0)
```

INIT_CAN initialisiert einen der CAN-Controller auf dem angegebenen Modul und bringt ihn in einen definierten Anfangszustand.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
INIT_CAN(module, channel)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	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 werden auf „must match“ setzen.
- Clockout-Register auf 0 setzen (= externe Frequenz wird nicht geteilt).
- Bus-Configuration-Register auf 0 setzen.
- Übertragungsrate für den CAN-Bus auf 1 MBit/s setzen.
- Alle Message-Objekte sperren.

Diese Anweisung muss zu Beginn des Prozesses (am besten im Prozessabschnitt **LOWINIT** oder **INIT**) ausgeführt werden, bevor andere Befehle auf den CAN-Controller zugreifen.

Bei Low speed CAN beträgt die Übertragungsrate maximal 125kBit/s und muss deswegen auf jeden Fall mit **SET_CAN_BAUDRATE** neu eingestellt werden.

Siehe auch

[EN_RECEIVE](#), [EN_TRANSMIT](#), [GET_CAN_REG](#), [SET_CAN_BAUDRATE](#), [SET_CAN_REG](#)

Gültig für Module

CAN-1, CAN-2

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
INIT:
    REM Initialisierung des CAN-Controllers 1 auf dem CAN-Modul 1
    INIT_CAN(1,1)
```

INIT_CAN



READ_MSG

READ_MSG gibt zurück, ob eine neue Nachricht in einem Message-Objekt eines der CAN-Controller auf dem angegebenen Modul empfangen wurde. Falls ja, wird der Nachrichteninhalt in das Feld **CAN_MSG** kopiert und der Identifier zurückgegeben.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
ret_val = READ_MSG(module, channel, msgno)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
msgno	Nummer (1... 15) des Message-Objektes im CAN-Controller.	LONG
ret_val	Nachrichtenstatus und -inhalt: ≥-1: Eine neue Nachricht ist eingegangen, der Wert ist der Identifier des Message-Objektes. -1: keine neue Nachricht vorhanden.	LONG

Bemerkungen

Das angegebene Message-Objekt muss zuvor mit der Anweisung **EN_TRANSMIT** für den Empfang freigegeben werden.

Wenn eine neue Nachricht in dem Message-Objekt empfangen wurde, werden dessen Daten ausgelesen und in das Feld **CAN_MSG []** kopiert. Der Rückgabewert ist in diesem Fall der Identifier der empfangenen Nachricht. Die Bits, die den Empfang einer neuen Nachricht anzeigen, werden zurückgesetzt, so dass eine weitere Nachricht empfangen werden kann.

Diese Anweisung eignet sich auch für die Abfrage der Message-Objekte im Polling-Modus, da hier zunächst eine Überprüfung stattfindet, ob neue Nachrichten angekommen sind. Wenn dies nicht der Fall ist, tritt kein Fehler im Programm auf.

Siehe auch

[CAN_MSG](#), [EN_RECEIVE](#), [EN_TRANSMIT](#), [INIT_CAN](#), [TRANSMIT](#)

Gültig für Module

CAN-1, CAN-2

Beispiel

```
REM Wenn eine neue Nachricht mit dem passenden Identifier
REM empfangen wurde, werden die Daten gelesen. Die
REM ersten 4 Bytes der Nachricht werden zu einer Fließkomma-
REM Zahl mit 32 Bit Länge zusammengesetzt (Senden einer
REM Fließkomma-Zahl siehe Bsp. bei TRANSMIT).
#include ADwinPRO_ALL.inc
DIM n AS LONG

INIT:
  PAR_1 = 0
  INIT_CAN(1,1)          'CAN-Controller 1 initialisieren
  EN_RECEIVE(1,1,8,40,0) 'Message-Objekt 8 initialisieren
                        'zum Empfangen von CAN-Nachrichten
                        'mit dem Identifier 40

EVENT:
  REM Wenn das Message-Objekt geändert wurde, werden die
  REM empfangenen Daten aus Objekt 8 gelesen und der
  REM Identifier an PAR_9 übergeben.
  REM Die Daten stehen im Feld CAN_MSG[] bereit.
  PAR_9 = READ_MSG(1,1,8)

  IF (PAR_9 = 40) THEN
    REM Für das Message-Objekt ist eine neue Nachricht mit dem
    REM Identifier 40 eingetroffen
    PAR_1 = CAN_MSG[1]      'High-Byte auslesen
    FOR n = 2 TO 4          'Mit restlichen 3 Bytes zu 32 Bit-Zahl
      PAR_1 = SHIFT_LEFT(PAR_1,8) + CAN_MSG[n] 'zusammenfügen
    NEXT n
    REM Das Bitmuster in PAR_1 in den Datentyp FLOAT wandeln und
    REM der Variablen FPAR_1 zuweisen.
    FPAR_1 = CAST_LONGTOFLOAT(PAR_1)
  ENDIF
```

SET_CAN_BAUDRATE

SET_CAN_BAUDRATE stellt die angegebene Baudrate auf einem der Controller auf dem angegebenen Modul ein und gibt zurück, ob dies erfolgreich geschehen ist.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
ret_val = SET_CAN_BAUDRATE(module, channel, rate)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
rate	Baudrate des CAN-Controllers: High speed CAN: 5000...1 000 000 Bit/s. Low speed CAN: 5000 ... 125 000 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“. 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 Handbuch „Pro-Hardware“.

Die Anweisung sollte in den Programm-Abschnitten **LOWINIT**: oder **INIT**: aufgerufen werden, und zwar erst nach der Anweisung **INIT_CAN**, weil sonst die eingestellte Baudrate wieder mit der Standardeinstellung (1 MBit/s) überschrieben wird.

Siehe auch

[GET_CAN_REG](#), [INIT_CAN](#), [SET_CAN_REG](#)

Gültig für Module

CAN-1, CAN-2



Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM status AS LONG
INIT:
  INIT_CAN(1,1) 'Initialisierung des CAN-Controllers
  status = SET_CAN_BAUDRATE(1,1,125000) 'Baudrate = 125 kBit/s
```

Einstellbare Baudraten

Einstellbare Baudraten [Bit/s]				
1000000.0000	888888.8889	800000.0000	727272.7273	666666.6667
615384.6154	571428.5714	533333.3333	500000.0000	470588.2353
444444.4444	421052.6316	400000.0000	380952.3810	363636.3636
347826.0870	333333.3333	320000.0000	307692.3077	296296.2963
285714.2857	266666.6667	250000.0000	242424.2424	235294.1176
222222.2222	210526.3158	205128.2051	200000.0000	190476.1905
181818.1818	177777.7778	173913.0435	166666.6667	160000.0000
156862.7451	153846.1538	148148.1481	145454.5455	142857.1429
140350.8772	133333.3333	126984.1270	125000.0000	123076.9231
121212.1212	117647.0588	115942.0290	114285.7143	111111.1111
106666.6667	105263.1579	103896.1039	102564.1026	100000.0000
98765.4321	95238.0952	94117.6471	90909.0909	88888.8889
87912.0879	86956.5217	84210.5263	83333.3333	81632.6531
80808.0808	80000.0000	78431.3725	76923.0769	76190.4762
74074.0741	72727.2727	71428.5714	70175.4386	69565.2174
68376.0684	67226.8908	66666.6667	66115.7025	64000.0000
63492.0635	62500.0000	61538.4615	60606.0606	60150.3759
59259.2593	58823.5294	57971.0145	57142.8571	55944.0559
55555.5556	54421.7687	53333.3333	52631.5789	52287.5817
51948.0519	51282.0513	50000.0000	49689.4410	49382.7160
48484.8485	47619.0476	47337.2781	47058.8235	46783.6257
45714.2857	45454.5455	44444.4444	43956.0440	43478.2609
42780.7487	42328.0423	42105.2632	41666.6667	41025.6410
40816.3265	40404.0404	40000.0000	39215.6863	38647.3430
38461.5385	38277.5120	38095.2381	37037.0370	36363.6364
36199.0950	35714.2857	35555.5556	35087.7193	34782.6087
34632.0346	34482.7586	34188.0342	33613.4454	33333.3333
33057.8512	32921.8107	32388.6640	32258.0645	32000.0000
31746.0317	31620.5534	31372.5490	31250.0000	30769.2308
30651.3410	30303.0303	30075.1880	29629.6296	29411.7647
29304.0293	29090.9091	28985.5072	28673.8351	28571.4286
28070.1754	27972.0280	27777.7778	27681.6609	27586.2069
27210.8844	27027.0270	26936.0269	26755.8528	26666.6667
26315.7895	26143.7908	25974.0260	25806.4516	25641.0256
25396.8254	25078.3699	25000.0000	24844.7205	24767.8019
24691.3580	24615.3846	24390.2439	24242.4242	24024.0240
23809.5238	23668.6391	23529.4118	23460.4106	23391.8129
23255.8140	23188.4058	22988.5057	22857.1429	22792.0228
22727.2727	22408.9636	22222.2222	22160.6648	22038.5675
21978.0220	21739.1304	21680.2168	21621.6216	21505.3763
21390.3743	21333.3333	21276.5957	21220.1592	21164.0212
21052.6316	20833.3333	20779.2208	20671.8346	20512.8205
20460.3581	20408.1633	20202.0202	20050.1253	20000.0000
19851.1166	19753.0864	19704.4335	19656.0197	19607.8431
19512.1951	19323.6715	19230.7692	19138.7560	19047.6190

Einstellbare Baudraten [Bit/s]				
18912.5296	18867.9245	18823.5294	18648.0186	18604.6512
18518.5185	18433.1797	18390.8046	18306.6362	18181.8182
18140.5896	18099.5475	18018.0180	17857.1429	17777.7778
17738.3592	17582.4176	17543.8596	17429.1939	17391.3043
17316.0173	17241.3793	17204.3011	17094.0171	17021.2766
16949.1525	16913.3192	16842.1053	16806.7227	16771.4885
16666.6667	16632.0166	16563.1470	16528.9256	16460.9053
16393.4426	16326.5306	16260.1626	16227.1805	16194.3320
16161.6162	16129.0323	16000.0000	15873.0159	15810.2767
15779.0927	15686.2745	15625.0000	15594.5419	15503.8760
15473.8878	15444.0154	15384.6154	15325.6705	15238.0952
15180.2657	15151.5152	15122.8733	15094.3396	15065.9134
15037.5940	15009.3809	14842.3006	14814.8148	14705.8824
14652.0147	14571.9490	14545.4545	14519.0563	14492.7536
14414.4144	14336.9176	14311.2701	14285.7143	14260.2496
14184.3972	14109.3474	14035.0877	13986.0140	13937.2822
13913.0435	13888.8889	13840.8304	13793.1034	13722.1269
13675.2137	13605.4422	13582.3430	13559.3220	13513.5135
13468.0135	13445.3782	13377.9264	13333.3333	13289.0365
13223.1405	13157.8947	13136.2890	13114.7541	13093.2897
13071.8954	13008.1301	12987.0130	12903.2258	12882.4477
12820.5128	12800.0000	12759.1707	12718.6010	12698.4127
12578.6164	12558.8697	12539.1850	12500.0000	12422.3602
12403.1008	12383.9009	12345.6790	12326.6564	12307.6923
12288.7865	12195.1220	12158.0547	12121.2121	12066.3650
12030.0752	12012.0120	11994.0030	11922.5037	11904.7619
11851.8519	11834.3195	11764.7059	11730.2053	11695.9064
11661.8076	11627.9070	11611.0305	11594.2029	11544.0115
11494.2529	11477.7618	11428.5714	11396.0114	11379.8009
11363.6364	11347.5177	11299.4350	11220.1964	11204.4818
11188.8112	11111.1111	11080.3324	11034.4828	11019.2837
10989.0110	10943.9124	10928.9617	10884.3537	10869.5652
10840.1084	10810.8108	10796.2213	10781.6712	10752.6882
10695.1872	10666.6667	10638.2979	10610.0796	10582.0106
10540.1845	10526.3158	10457.5163	10430.2477	10416.6667
10389.6104	10335.9173	10322.5806	10296.0103	10269.5764
10256.4103	10230.1790	10204.0816	10101.0101	10088.2724
10062.8931	10025.0627	10012.5156	10000.0000	9937.8882
9925.5583	9876.5432	9852.2167	9828.0098	9803.9216
9791.9217	9768.0098	9756.0976	9696.9697	9685.2300
9661.8357	9615.3846	9603.8415	9569.3780	9523.8095
9456.2648	9433.9623	9411.7647	9400.7051	9367.6815
9356.7251	9324.0093	9302.3256	9291.5215	9259.2593
9227.2203	9216.5899	9195.4023	9153.3181	9142.8571
9090.9091	9070.2948	9049.7738	9039.5480	9009.0090
8958.5666	8928.5714	8918.6176	8888.8889	8879.0233

Einstellbare Baudraten [Bit/s]				
8869.1796	8859.3577	8771.9298	8743.1694	8714.5969
8695.6522	8658.0087	8648.6486	8620.6897	8602.1505
8592.9108	8556.1497	8547.0085	8510.6383	8483.5631
8474.5763	8465.6085	8456.6596	8421.0526	8403.3613
8385.7442	8333.3333	8281.5735	8264.4628	8255.9340
8230.4527	8205.1282	8196.7213	8163.2653	8130.0813
8113.5903	8105.3698	8097.1660	8088.9788	8080.8081
8064.5161	8000.0000	7976.0718	7944.3893	7936.5079
7905.1383	7843.1373	7812.5000	7804.8780	7797.2710
7774.5384	7751.9380	7736.9439	7729.4686	7714.5612
7692.3077	7662.8352	7655.5024	7619.0476	7590.1328
7575.7576	7561.4367	7547.1698	7532.9567	7518.7970
7469.6545	7441.8605	7421.1503	7407.4074	7400.5550
7386.8883	7352.9412	7326.0073	7285.9745	7272.7273
7259.5281	7246.3768	7187.7808	7168.4588	7142.8571
7136.4853	7130.1248	7111.1111	7098.4916	7092.1986
7054.6737	7017.5439	6993.0070	6956.5217	6944.4444
6926.4069	6902.5022	6896.5517	6861.0635	6820.1194
6808.5106	6802.7211	6791.1715	6779.6610	6734.0067
6688.9632	6683.3751	6666.6667	6611.5702	6578.9474
6568.1445	6562.7564	6557.3770	6535.9477	6530.6122
6493.5065	6456.8200	6451.6129	6441.2238	6410.2564
6400.0000	6379.5853	6349.2063	6324.1107	6289.3082
6274.5098	6269.5925	6250.0000	6245.1210	6211.1801
6172.8395	6163.3282	6153.8462	6144.3932	6102.2121
6060.6061	6046.8632	6037.7358	5997.0015	5961.2519
5952.3810	5925.9259	5895.3574	5865.1026	5847.9532
5818.1818	5797.1014	5772.0058	5747.1264	5714.2857
5702.0670	5681.8182	5649.7175	5614.0351	5610.0982
5555.5556	5521.0490	5517.2414	5464.4809	5434.7826
5423.7288	5376.3441	5333.3333	5291.0053	5245.9016
5208.3333	5161.2903	5079.3651	5000.0000	

SET_CAN_REG schreibt einen Wert in ein Register des gewählten CAN-Controllers auf dem angegebenen Modul.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
SET_CAN_REG(module, channel, regno, value)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
regno	Register-Nummer (0...255) des CAN-Controllers.	LONG
value	Wert (0...255), der in das Controller-Register geschrieben werden soll.	LONG

Bemerkungen

Die anzugebende Registernummer entspricht der Registernummer des CAN-Controllers (siehe Datenblatt AN82527 von Intel®). Beispielsweise hat das Kontroll-Register die Adresse 0 und das Status-Register die Adresse 1.

Siehe auch

[INIT_CAN](#), [SET_CAN_BAUDRATE](#), [GET_CAN_REG](#)

Gültig für Module

CAN-1, CAN-2

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
INIT:
    INIT_CAN(1,1)           'Initialisierung des CAN-Controllers
    SET_CAN_REG(1,1,0,1)    'Setze Control-Register auf den Wert 1
```

SET_CAN_REG

TRANSMIT

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
TRANSMIT (module, channel, msgno)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Nummer (1, 2) des CAN-Kanals, der den CAN-Controller bestimmt.	LONG
<code>msgno</code>	Nummer (1... 14) des Message-Objektes im CAN-Controller.	LONG

Bemerkungen

Dieser Befehl erfüllt seine Funktion nur, wenn das entsprechende Message-Objekt zuvor mit **EN_TRANSMIT** zum Senden konfiguriert wurde.

Die Nachrichten-Daten müssen vor der Ausführung des Befehls **TRANSMIT** im Feld `CAN_MSG[]` eingetragen werden.

Siehe auch

[CAN_MSG](#), [EN_RECEIVE](#), [EN_TRANSMIT](#), [READ_MSG](#)

Gültig für Module

CAN-1, CAN-2

Beispiel

```
REM Sendet eine 32 Bit-FLOAT-Zahl (hier: Pi) als Folge von
REM 4 Bytes in einem Message-Objekt
REM (Empfangen einer Fließkomma-Zahl siehe Bsp. bei READ_MSG)
```

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE pi 3.14159265
DIM i AS LONG
```

INIT:

```
INIT_CAN (1,1) 'CAN-Controller initialisieren

REM Initialisiere das Message-Objekt 6
REM zum Senden von CAN-Nachrichten mit dem Identifier 40
EN_TRANSMIT (1,1,6,40,0)
```

```
REM Bitmuster von Pi mit Datenformat Long erzeugen
PAR_1 = CAST_FLOATTOLONG(pi)
```

```
REM Bitmuster (32 Bit) in 4 Bytes aufteilen
CAN_MSG[4] = PAR_1 AND 0FFh 'LSB zuweisen
FOR i = 1 TO 3
    CAN_MSG[4-i] = SHIFT_RIGHT(PAR_1,8*i) AND 0FFh
NEXT i
CAN_MSG[9] = 4 'Länge der Nachricht in Bytes
```

EVENT:

```
TRANSMIT (1,1,6) 'Message-Objekt 6 senden
```


CHANGED_DATA überprüft, ob sich auf dem angegebenen Modul seit dem letzten Zugriff der Applikation auf das DP-RAM die Daten im Ausgangsbereich geändert haben.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = CHANGED_DATA(module, outsize)
```

Parameter

module	Eingestellte Moduladresse (1...4).	LONG
outsize	Größe des zu prüfenden Ausgangsbereichs in Byte.	LONG
ret_val	Merker für Datenänderungen im Ausgangsbereich: 0: Daten sind unverändert. ≠0: neue Daten vorhanden.	LONG

Bemerkungen

Diese Anweisung kann verwendet werden um Rechenzeit zu sparen, indem man nur dann Daten ausliest, wenn sie sich geändert haben.

Der Merker, der dieser Anweisung zu Grunde liegt, wird zurückgesetzt, wenn das Zugriffsrecht auf den Ausgangsbereich von der Applikation zur Feldbusseite wechselt. Dies ist unabhängig davon, ob die Anweisung **CHANGED_DATA** aufgerufen wurde oder nicht.

Die Anweisung kann nur genutzt werden, wenn sie während der Initialisierung freigegeben wurde (siehe auch **INIT_SLAVE**).



Gültig für Module

Inter-SL, Profi-SL

Siehe auch

CHECK_ACCESS, GET_PRO_BYTE, GET_READ_BUFFER, INIT_SLAVE, REQUEST_ACCESS, REQUEST_RELEASE_ACCESS, SET_PRO_BYTE, SET_WRITE_BUFFER

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

EVENT:

```
REQUEST_ACCESS(1,2)      'Zugriff auf DP-RAM beantragen
                          '(Ausgangsbereich)
PAR_1 = CHECK_ACCESS(1)   'Status der Zugriffsrechte lesen
IF (PAR_1 = 2) THEN       'Wenn Zugriffsrecht erteilt ist ...
    IF (CHANGED_DATA(1,10) <> 0) THEN 'und neue Daten vorhanden ...
        PAR_2 = GET_PRO_BYTE(1,10) 'ein Byte lesen
    ENDIF
ENDIF
REQUEST_RELEASE_ACCESS(1,2) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „[Datenaustausch mit dem Feldbus](#)“ auf Seite 335.

CHECK_ACCESS

CHECK_ACCESS gibt zurück, auf welche Bereiche des DP-RAM im angegebenen Modul die Applikation das Zugriffsrecht hat.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = CHECK_ACCESS(module)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...4).	LONG
<code>ret_val</code>	Bitmuster für den Zugriffsstatus der Bereiche: Bit = 0: Applikation hat kein Zugriffsrecht. Bit = 1: Applikation hat Zugriffsrecht.	LONG

Bit-Nr.	31:3	2	1	0
Daten-bereich	–	Eingang	Ausgang	Kontroll-Register

Bemerkungen

Wenn das Zugriffsrecht für einen Datenbereich des DP-RAM nicht bei der Applikation liegt, kann auf diesen Bereich nicht zugegriffen werden. Dies ist erforderlich, weil sonst die Daten inkonsistent werden und das System in einen undefinierten Zustand geraten kann.

Gültig für Module

Inter-SL, Profi-SL

Siehe auch

CHANGED_DATA, GET_PRO_BYTE, GET_READ_BUFFER, INIT_SLAVE, REQUEST_ACCESS, REQUEST_RELEASE_ACCESS, SET_PRO_BYTE, SET_WRITE_BUFFER

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

EVENT:

```
REQUEST_ACCESS(1,1)      'Zugriff auf DP-RAM beantragen
                          '(Kontrollregister)
PAR_1 = CHECK_ACCESS(1)  'Zugriffsrechte lesen
IF (PAR_1 = 1) THEN      'Leserecht erteilt?
    PAR_2 = GET_PRO_BYTE(1,7F6h) 'ein Byte lesen
ENDIF
REQUEST_RELEASE_ACCESS(1,1) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „Datenaustausch mit dem Feldbus“ auf Seite 335.

GET_PRO_BYTE gibt ein Byte aus einer bestimmten Speicheradresse des DP-RAM des Feldbus-Moduls zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = GET_PRO_BYTE(module, byteno)
```

Parameter

module	Eingestellte Moduladresse (1...4).	LONG
byteno	Speicheradresse im DP-RAM.	LONG
ret_val	Inhalt der Speicheradresse aus dem DP-RAM (0...256).	LONG

Bemerkungen

Die Anweisung kann auf jede Speicherstelle zugreifen, unabhängig davon, ob die Applikation Zugriffsrecht hat oder nicht. Wenn die Applikation Daten ohne Zugriffsrecht ausliest, können fehlerhafte Daten übertragen oder ein Busfehler erzeugt werden.

Achten Sie daher darauf, die Anweisung nicht auf einen unerlaubten Bereich oder zu einem unerlaubten Zeitpunkt verwenden.



Gültig für Module

Inter-SL, Profi-SL

Siehe auch

CHANGED_DATA, CHECK_ACCESS, GET_READ_BUFFER, INIT_SLAVE, REQUEST_ACCESS, REQUEST_RELEASE_ACCESS, SET_PRO_BYTE, SET_WRITE_BUFFER

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

INIT:
REM Inhalt von Byte-Nummer 7CDh (Feldbustyp) lesen
PAR_1 = GET_PRO_BYTE(1, 7CDh)
```

GET_READ_BUFFER

GET_READ_BUFFER kopiert einen definierten Datenblock aus dem Speicherbereich des DP-RAM in das angegebene Zielfeld.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
GET_READ_BUFFER(module,array[],start,count)
```

Parameter

module	Eingestellte Moduladresse (1...4).	LONG
array[]	Name des Zielfelds.	ARRAY LONG
start	Erstes Element des zu kopierenden Datenblocks im DP-RAM.	LONG
count	Anzahl der zu kopierenden Datenbytes aus dem DP-RAM.	LONG

Bemerkungen

Der Befehl darf nur genutzt werden, wenn das Zugriffsrecht für den entsprechenden Teil des DP-RAM bei der Applikation liegt.

Das Zielfeld, in das die Daten übernommen werden sollen, muss bereits deklariert sein und zwar mindestens mit sovielen Feldelementen wie Datenbytes kopiert werden.

Gültig für Module

Inter-SL, Profi-SL

Siehe auch

CHANGED_DATA, CHECK_ACCESS, GET_PRO_BYTE, INIT_SLAVE, REQUEST_ACCESS, REQUEST_RELEASE_ACCESS, SET_PRO_BYTE, SET_WRITE_BUFFER

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM DATA_1[100] AS LONG  
  
EVENT:  
    REQUEST_ACCESS(1,2)      'Zugriff auf DP-RAM beantragen  
                             '(Ausgangsbereich)  
    PAR_1 = CHECK_ACCESS(1)  'Status der Zugriffsrechte lesen  
    IF (PAR_1 = 2) THEN      'Wenn Zugriffsrecht erteilt ist...  
        IF (CHANGED_DATA(1,10) <> 0) THEN 'und neue Daten vorhanden...  
            GET_READ_BUFFER(1,DATA_1,200h,10) '10 Bytes lesen  
        ENDIF  
    ENDIF  
    REQUEST_RELEASE_ACCESS(1,2) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „[Datenaustausch mit dem Feldbus](#)“ auf [Seite 335](#).

INIT_SLAVE initialisiert den Feldbus-Slave und ist nur nach einem Einschalten (power up) möglich.

INIT_SLAVE

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = INIT_SLAVE(module, IO_in, PAR_in, IO_out,
'par_out, in_hdl, out_hdl, intr)
```

Parameter

module	Eingestellte Moduladresse (1...4).	LONG																
IO_in	Größe des Eingangs-Datenbereichs für zyklische Daten in Byte.	LONG																
par_in	Länge des Eingangs-Datenbereichs für azyklische Daten in Byte.	LONG																
IO_out	Länge des Ausgangs-Datenbereichs für zyklische Daten in Byte.	LONG																
par_out	Länge des Ausgangs-Datenbereichs für azyklische Daten in Byte.	LONG																
in_hdl	Bitmuster zur Handhabung des Eingangs-Datenbereichs:	LONG																
<table><tr><td>Bit-Nr.</td><td>31:2</td><td>1</td><td>0</td></tr><tr><td>Bit = 0</td><td>–</td><td>Merker deaktivieren</td><td>Eingangsbereich für CHANGED_DATA löschen, sobald die Applikation stoppt.</td></tr><tr><td>Bit = 1</td><td>–</td><td>Merker aktivieren</td><td>für Eingangsbereich einfrieren, sobald die Applikation stoppt.</td></tr></table>			Bit-Nr.	31:2	1	0	Bit = 0	–	Merker deaktivieren	Eingangsbereich für CHANGED_DATA löschen, sobald die Applikation stoppt.	Bit = 1	–	Merker aktivieren	für Eingangsbereich einfrieren, sobald die Applikation stoppt.				
Bit-Nr.	31:2	1	0															
Bit = 0	–	Merker deaktivieren	Eingangsbereich für CHANGED_DATA löschen, sobald die Applikation stoppt.															
Bit = 1	–	Merker aktivieren	für Eingangsbereich einfrieren, sobald die Applikation stoppt.															
out_hdl	Bitmuster zur Handhabung des Ausgangs-Datenbereichs:	LONG																
<table><tr><td>Bit-Nr.</td><td>31:2</td><td>1</td><td>0</td></tr><tr><td>Bit = 0</td><td>–</td><td>Ausgangsbereich Feldbus Off-Line starten, sobald derten. Feldbus stoppt.</td><td></td></tr><tr><td>Bit = 1</td><td>–</td><td>Ausgangsbereich ein-Feldbus On-Line starten, sobald derten. Feldbus stoppt.</td><td></td></tr></table>			Bit-Nr.	31:2	1	0	Bit = 0	–	Ausgangsbereich Feldbus Off-Line starten, sobald derten. Feldbus stoppt.		Bit = 1	–	Ausgangsbereich ein-Feldbus On-Line starten, sobald derten. Feldbus stoppt.					
Bit-Nr.	31:2	1	0															
Bit = 0	–	Ausgangsbereich Feldbus Off-Line starten, sobald derten. Feldbus stoppt.																
Bit = 1	–	Ausgangsbereich ein-Feldbus On-Line starten, sobald derten. Feldbus stoppt.																
intr	Bitmuster zur Handhabung des Interrupts.	LONG																
<table><tr><td>Bit-Nr.</td><td>31:2</td><td>1</td><td>0</td></tr><tr><td></td><td>–</td><td>Interrupt auslösen, wenn der Feldbus On-Line geht:</td><td>Interrupt auslösen, wenn Merker für CHANGED_DATA aktiviert ist:</td></tr><tr><td>Bit = 0.</td><td>–</td><td>Nein</td><td>Nein</td></tr><tr><td>Bit = 1.</td><td>–</td><td>Ja</td><td>Ja</td></tr></table>			Bit-Nr.	31:2	1	0		–	Interrupt auslösen, wenn der Feldbus On-Line geht:	Interrupt auslösen, wenn Merker für CHANGED_DATA aktiviert ist:	Bit = 0.	–	Nein	Nein	Bit = 1.	–	Ja	Ja
Bit-Nr.	31:2	1	0															
	–	Interrupt auslösen, wenn der Feldbus On-Line geht:	Interrupt auslösen, wenn Merker für CHANGED_DATA aktiviert ist:															
Bit = 0.	–	Nein	Nein															
Bit = 1.	–	Ja	Ja															
ret_val	Bitmuster als Status der Initialisierung: Bit = 0: kein Fehler. Bit = 1: Fehler aufgetreten (Bedeutung s.u.).	LONG																

Bit-Nr.	31:16	15	14	13:8	7	6	5:3	2	1	0
Fehler.	–	A ₇	A ₆	–	A ₅	A ₄	–	A ₃	A ₂	A ₁

- A₁: Fehler bei Hardware-Überprüfung.
- A₂: Fehler beim Start der Initialisierung.
- A₃: Fehler bei der Initialisierung.
- A₄: Fehler im DP-RAM.
- A₅: Fehler beim Abschluss der Initialisierung.
- A₆: Keine Nachricht empfangen.
- A₇: Keine Nachricht gesendet.

Bemerkungen

Diese Anweisung muss vor dem Arbeiten mit dem Slave-Modul ausgeführt werden.

Eine zweite Initialisierung nach dem Einschalten ist nicht möglich.

Während der Initialisierung mit **INIT_SLAVE** wird die Größe des Ein- und Ausgangsbereichs festgelegt (getrennt für zyklische und azyklische Daten). Die Größe muss mit der projektierten Größe im zugehörigen Master übereinstimmen. Die maximale Größe der einzelnen Bereiche ist je nach Feldbus unterschiedlich.

Nach der erfolgreichen Initialisierung ist der Slave bereit, am Datenverkehr teilzunehmen und es können Parameter aus dem DP-RAM ausgelesen werden. Die Applikation darf nur Informationen ins DP-RAM schreiben, wenn sie die Zugriffsrechte dazu besitzt!

Wurden bei der ersten Initialisierung die falschen Daten übergeben, muss das System ausgeschaltet werden. Erst nach einem erneuten Einschalten kann das Modul neu initialisiert werden.

Die Anweisung löst eine Ablaufkette aus, die etwa 2-3 Sekunden dauert. Wenn die Anweisung aus einem (nicht unterbrechbaren) hochprioritären Prozess aufgerufen wird, kann in dieser Zeit keine Kommunikation zwischen PC und ADwin-System stattfinden. Es ist deshalb empfehlenswert, die Initialisierung aus einem niedrigprioritären Prozess oder einem Prozessabschnitt mit niedriger Priorität (z.B. **LOWINIT**:) heraus aufzurufen.



Gültig für Module

Inter-SL, Profi-SL

Siehe auch

CHANGED_DATA, **CHECK_ACCESS**, **GET_PRO_BYTE**, **GET_READ_BUFFER**, **REQUEST_ACCESS**, **REQUEST_RELEASE_ACCESS**, **SET_PRO_BYTE**, **SET_WRITE_BUFFER**

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
REM Slave initialisieren:
REM Nur zyklische Daten (10 IO_in / 10 IO_out),
REM CHANGE_DATA-Funktion ein, keine Interrupts,
REM Ausgänge werden gelöscht, wenn Bus OFF-Line.
PAR_1 = INIT_SLAVE(1,10,0,10,0,2,0,0)
```

REQUEST_ACCESS

REQUEST_ACCESS beantragt das Zugriffsrecht auf das DP-RAM des Slaves für die Applikation.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
REQUEST_ACCESS (module, area)
```

Parameter

module	Eingestellte Moduladresse (1...4).	LONG
area	Bitmuster für den Bereich, für den der Zugriffsstatus geändert wird: Bit = 0: Zugriffsrecht bleibt unverändert. Bit = 1: Zugriffsrecht wird beantragt.	LONG

Bit-Nr.	31:3	2	1	0
Bereich	–	Eingang	Ausgang	Kontroll-Register

Bemerkungen

Wenn das Zugriffsrecht für mehrere Bereiche gleichzeitig beantragt wird, kann es sein, dass die Feldbusseite den Zugriff auf einen Teilbereich verweigert. Dann kann auf die anderen Bereiche trotzdem zugegriffen werden.

Liegt das Zugriffsrecht für das DP-RAM nicht bei der Applikation, darf nicht auf den Bereich zugegriffen werden, da die Daten inkonsistent werden und das System in einen undefinierten Zustand kommen kann.

Nach dem Datenaustausch mit dem DP-RAM sollte das Zugriffsrecht mit **REQUEST_RELEASE_ACCESS** wieder an die Feldbusseite zurückgegeben werden, damit die Daten an den Master weitergeleitet und die Daten vom Master in das DP-RAM geschrieben werden können. Gibt die Applikation das Zugriffsrecht nicht aktiv an die Busseite zurück, fällt es nach 1 Sekunde automatisch zurück.

Gültig für Module

Inter-SL, Profi-SL

Siehe auch

[CHANGED_DATA](#), [CHECK_ACCESS](#), [GET_PRO_BYTE](#), [SET_PRO_BYTE](#), [GET_READ_BUFFER](#), [INIT_SLAVE](#), [SET_WRITE_BUFFER](#), [REQUEST_RELEASE_ACCESS](#)

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

EVENT:
    REQUEST_ACCESS (1,1)           'Zugriff auf DP-RAM beantragen
                                   '(Kontroll-Register)
    PAR_1 = CHECK_ACCESS (1)       'Lese Zugriffsrecht-Status
    IF (PAR_1 = 1) THEN            'Wenn Zugriffsrecht erteilt ist,
        PAR_2 = GET_PRO_BYTE (1,7F6h) 'ein Byte lesen
    ENDIF
    REQUEST_RELEASE_ACCESS (1,1)   'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „[Datenaustausch mit dem Feldbus](#)“ auf Seite 335.



REQUEST_RELEASE_ACCESS beantragt, das Zugriffsrecht auf das DP-RAM des Slaves an den Feldbus zurückzugeben.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

REQUEST_RELEASE_ACCESS (module, area)
```

Parameter

module	Eingestellte Moduladresse (1...4).	LONG
area	Bitmuster für den Bereich, für den der Zugriffsstatus geändert wird: Bit = 0: Zugriffsrecht bleibt unverändert. Bit = 1: Zugriffsrecht wird zurückgegeben.	LONG

Bit-Nr.	31:3	2	1	0
Datenbereich	–	Eingang	Ausgang	Kontroll-Register

Bemerkungen

Das Zugriffsrecht kann für mehrere Bereiche gleichzeitig zurückgegeben werden.

Nach dem Datenaustausch mit dem DP-RAM sollte das Zugriffsrecht wieder an die Feldbusseite zurückgegeben werden, damit die Daten an den Master weitergeleitet und die Daten vom Master in das DP-RAM geschrieben werden können. Gibt die Applikation das Zugriffsrecht nicht aktiv an die Busseite zurück, fällt es nach 1 Sekunde automatisch zurück.

Gültig für Module

Inter-SL, Profi-SL

Siehe auch

CHANGED_DATA, CHECK_ACCESS, GET_PRO_BYTE, GET_READ_BUFFER, INIT_SLAVE, REQUEST_ACCESS, SET_PRO_BYTE, SET_WRITE_BUFFER

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

EVENT:

```
REQUEST_ACCESS (1,1)      'Zugriff auf DP-RAM beantragen
                           '(Kontroll-Register).

PAR_1 = CHECK_ACCESS (1)  'Lese Status der Zugriffsrechte
IF (PAR_1 = 1) THEN      'Wenn Zugriffsrecht erteilt ist,
    PAR_2 = GET_PRO_BYTE (1,7F6h) 'ein Byte lesen
ENDIF
REQUEST_RELEASE_ACCESS (1,1) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „Datenaustausch mit dem Feldbus“ auf Seite 335.

REQUEST_RELEASE_ACCESS

SET_PRO_BYTE

SET_PRO_BYTE schreibt einen Wert in eine Speicherstelle des DP-RAM des Feldbus-Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
SET_PRO_BYTE(module,byteno,value)
```

Parameter

module	Eingestellte Moduladresse (1...4).	LONG
byteno	Speicheradresse im DP-RAM.	LONG
value	Wert, der in die Speicheradresse geschrieben werden soll (0...255).	LONG

Bemerkungen

Die Anweisung kann auf jede Speicherstelle zugreifen, unabhängig davon, ob die Applikation Zugriffsrecht hat oder nicht. Wenn die Applikation Daten ohne Zugriffsrecht ausliest, können fehlerhafte Daten übertragen oder ein Busfehler erzeugt werden.



Achten Sie daher darauf, die Anweisung nicht auf einen unerlaubten Bereich oder zu einem unerlaubten Zeitpunkt verwenden.

Gültig für Module

Inter-SL, Profi-SL

Siehe auch

CHANGED_DATA, CHECK_ACCESS, GET_PRO_BYTE, GET_READ_BUFFER, INIT_SLAVE, REQUEST_ACCESS, REQUEST_RELEASE_ACCESS, SET_WRITE_BUFFER

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
INIT:  
  REM Byte 0h mit dem Wert 2 beschreiben (zyklische Eingangsdaten)  
  SET_PRO_BYTE(1,0h,2)
```

SET_WRITE_BUFFER kopiert die Daten eines Felds in einen definierten Speicherbereich des DP-RAM.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

SET_WRITE_BUFFER(module,array[],start,count)
```

Parameter

module	Eingestellte Moduladresse (1...4).	LONG
array[]	Name des Quellfelds.	ARRAY LONG
start	Erste Speicherstelle im DP-RAM, die beschrieben wird.	LONG
count	Anzahl der Speicherstellen im DP-RAM, die insgesamt beschrieben werden.	LONG

Bemerkungen

Der Befehl darf nur genutzt werden, wenn das Zugriffsrecht für den entsprechenden Teil des DP-RAM bei der Applikation liegt.

Das Quellfeld, aus dem die Daten übernommen werden sollen, muss bereits deklariert sein und zwar mindestens mit so vielen Feldelementen wie Datenbytes kopiert werden.

Gültig für Module

Inter-SL, Profi-SL

Siehe auch

CHANGED_DATA, CHECK_ACCESS, GET_PRO_BYTE, GET_READ_BUFFER, INIT_SLAVE, REQUEST_ACCESS, REQUEST_RELEASE_ACCESS, SET_PRO_BYTE

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[100] AS LONG

EVENT:
  REQUEST_ACCESS(1,4) 'Zugriff auf DP-RAM beantragen
                      '(Eingangsbereich)
  PAR_1 = CHECK_ACCESS(1) 'Lese Zugriffsrecht-Status
  IF (PAR_1 = 2) THEN 'Wenn Zugriffsrecht erteilt ist ...
    IF (CHANGED_DATA(1,10) <> 0) THEN 'und neue Daten vorhanden:
      REM 10 Byte schreiben
      SET_WRITE_BUFFER(1,DATA_1,200h,10)
    ENDIF
  ENDIF
  REQUEST_RELEASE_ACCESS(1,4) 'Zugriff auf DP-RAM zurückgeben
```

Siehe auch Programmbeispiel „[Daten austausch mit dem Feldbus](#)“ auf [Seite 335](#).

SET_WRITE_BUFFER

CHECK_SHIFT_REG

CHECK_SHIFT_REG gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO des Kanals (auf dem angegebenen Modul) geschrieben wurden.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
ret_val = CHECK_SHIFT_REG(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Kanal, dessen Sende-Status gelesen werden soll (1, 2 oder 1...4).	LONG
ret_val	Sende-Status: 0: Daten sind gesendet (= keine Daten im Sende-FIFO vorhanden). 1: Noch nicht alle Daten gesendet (= im Sende-FIFO sind noch Daten vorhanden).	LONG

Bemerkungen

Bei dem Rückgabewert 0 ist sowohl das Sende-FIFO als auch das Ausgangs-Shiftregister leer. Bei dem Rückgabewert 1 ist mindestens ein Bit noch nicht gesendet.

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

[GET_RS](#), [READ_FIFO](#), [RS_INIT](#), [RS_RESET](#), [RS485_SEND](#), [SET_RS](#), [WRITE_FIFO](#)

Gültig für Module

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
  
EVENT :  
...  
    PAR_1 = CHECK_SHIFT_REG(1,1) 'Prüft, ob Schnittstelle 1 noch  
                                'Daten zu senden hat  
...  

```

GET_RS liest den Inhalt eines bestimmten Controller-Registers auf dem angegebenen Modul aus.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
ret_val = GET_RS(module, REGISTER)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
register	Adresse des zu lesenden Controller-Registers.	LONG
ret_val	Inhalt des Controller-Registers.	LONG

Bemerkungen

Benutzen Sie diesen Befehl nur, wenn Sie sich bereits eingehend mit dem eingesetzten Controller vertraut gemacht haben (Datenblatt des Herstellers Texas Instruments). Für allgemeine Anwendungen stehen Ihnen komfortablere Befehle aus der Include-Datei zur Verfügung.

Siehe auch

CHECK_SHIFT_REG, READ_FIFO, RS_INIT, RS_RESET, RS485_SEND, SET_RS, WRITE_FIFO

Gültig für Module

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

-/-

GET_RS

READ_FIFO

READ_FIFO liest einen Wert aus dem Eingangs-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = READ_FIFO(module, channel)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Nummer des auszulesenden Kanals (1, 2 oder 1...4).	LONG
ret_val	Inhalt des Eingangs-FIFO: -1: FIFO ist leer. ≥0: Übertragener Datenwert.	LONG

Bemerkungen

-/-

Siehe auch

[CHECK_SHIFT_REG](#), [GET_RS](#), [RS_INIT](#), [RS_RESET](#), [RS485_SEND](#), [SET_RS](#), [WRITE_FIFO](#)

Gültig für Module

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
  
INIT:  
  RS_RESET(1)  
  RS_INIT(1,1,9600,0,8,0,1) 'Initialisierung von Kanal 1 auf Modul  
                             '1 mit 9600 Baud, ohne Parität,  
                             '8 Datenbits, 1 Stoppbit und  
                             'Hardwarehandshake (nur RS232).  
  
EVENT:  
  PAR_1 = READ_FIFO(1,1) 'Einen Wert aus dem FIFO holen. Wenn  
                         'der FIFO leer ist, wird -1  
                         'zurückgeliefert.
```

Siehe auch weitere [Beispiele für RS232 und RS485](#) ab [Seite 336](#).

RS_INIT initialisiert einen bestimmten Kanal auf dem angegebenen Modul.

Die folgenden Parameter werden gesetzt:

- Übertragungsgeschwindigkeit in Baud
- Anwendung von Prüf-Bits
- Datenlänge
- Anzahl der Stopp-Bits
- Übertragungs-Protokoll (Handshake)

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

RS_INIT(module, channel, baud, parity, bits, stop,
        handshake)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...255).	LONG
<code>channel</code>	Kanal, der initialisiert werden soll (1, 2 oder 1...4).	LONG
<code>baud</code>	Übertragungsgeschwindigkeit in Baud: RS232: 35 ... 115200 Baud. RS422, RS485: 35...2304000 Baud.	LONG
<code>parity</code>	Anwendung von Prüf-Bits: 0: ohne Paritäts-Bit. 1: gerade Parität (even). 2: ungerade Parität (odd).	LONG
<code>bits</code>	Anzahl der Daten-Bits (5, 6, 7 oder 8).	LONG
<code>stop</code>	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
<code>handshake</code>	Übertragungs-Protokoll: 0: kein Handshake. 1: Hardware-Handshake (RTS/CTS), nur RS232. 2: Software-Handshake (Xon/Xoff). 3: RS485.	LONG

Bemerkungen

Diese Anweisung ist vor dem ersten Arbeiten mit dem gewählten Kanal notwendig, um die Schnittstellen-Parameter einzustellen. Sie müssen für eine korrekte Übertragung mit der Gegenstelle identisch sein.

Die Initialisierung ist auch dann erforderlich, nachdem Sie auf dem Modul mit **RS_RESET** einen Hardware-Reset ausgeführt haben.

Die Baudrate wird vom Grundtakt (2304000Hz) des moduleigenen Taktgebers abgeleitet. Es ist jede Baudrate einstellbar, die sich durch ganzzahlige Division des Grundtakts ergibt. Der Teiler kann Werte im Bereich 1...0FFFFh annehmen.

Die Baudrate ergibt sich aus der grundlegenden Taktrate von 2304MHz der Schnittstelle, dividiert durch einen ganzzahligen Divisor. Der Wertebereich des Divisors von 1 ... 0FFFFh ergibt eine Bandbreite von 35 ... 2304 000 Bit/s. Entsprechend der Spezifikation ist die RS232-Schnittstelle auf 115200 Bit/s beschränkt. Die folgende Liste zeigt einige übliche Baudraten.

RS_INIT



Übliche Baudraten [Bit/s]		
2304000	57600	2400
1152000	38400	1200
460800	19200	600
230400	9600	300
115200	4800	

Siehe auch

[CHECK_SHIFT_REG](#), [GET_RS](#), [READ_FIFO](#), [RS_RESET](#), [RS485_SEND](#), [SET_RS](#), [WRITE_FIFO](#)

Gültig für Module

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
RS_RESET(1)           'RS-Modul zurücksetzen
RS_INIT(1,1,9600,0,8,0,1) 'Initialisierung von Kanal 1 auf
                        'Modul 1 mit 9600 Baud, ohne Parität,
                        '8 Datenbits, 1 Stoppbit und
                        'Hardware-Handshake (nur RS232).
```

Siehe auch weitere [Beispiele für RS232 und RS485](#) ab [Seite 336](#).

RS_RESET führt auf dem angegebenen Modul einen Hardware-Reset durch und löscht dabei die Einstellungen für alle Kanäle.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

RS_RESET(module)
```

Parameter

module Eingestellte Moduladresse (1...255). LONG

Bemerkungen

Die Anweisung sendet einen Reset-Impuls auf den entsprechenden Eingang des Controllers TL16C754. Sie können dem Datenblatt des Controllers 16C754 von Texas Instruments entnehmen, auf welche Werte die Register durch den Hardware-Reset gesetzt werden.

Nach einem Hardware-Reset muss eine Initialisierung mit **RS_INIT** folgen, um den Controller zu initialisieren und die gewünschten Schnittstellen-Parameter einzustellen.

Siehe auch

[CHECK_SHIFT_REG](#), [GET_RS](#), [READ_FIFO](#), [RS_INIT](#), [RS485_SEND](#), [SET_RS](#), [WRITE_FIFO](#)

Gültig für Module

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

INIT:

```
RS_RESET(1)                    'RS-Modul zurücksetzen
RS_INIT(1,1,9600,0,8,0,1) 'Initialisierung von Kanal 1 auf
                              'Modul 1 mit 9600 Baud, ohne Parität,
                              '8 Datenbits, 1 Stoppbit und
                              'Hardware-Handshake (nur RS232).
```

Siehe auch weitere [Beispiele für RS232 und RS485](#) ab [Seite 336](#).

RS_RESET

RS485_SEND

RS485_SEND legt die Übertragungsrichtung für einen bestimmten Kanal des angegebenen Moduls fest.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
RS485_SEND (module, channel, dir)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Einzustellender Kanal (1, 2 oder 1...4).	LONG
dir	Übertragungsrichtung des Kanals: 0: Kanal als Empfänger einstellen. 1: Kanal als Sender einstellen. 2: Kanal als Sender einstellen, der gleichzeitig die gesendeten Daten empfängt.	LONG

Bemerkungen

Die Einstellung der Übertragungsrichtung bedeutet:

- Empfänger: Der Kanal kann Daten auf dem Bus ausschließlich lesen, auch wenn Daten im Ausgangs-FIFO des Controllers für diesen Kanal liegen.
- Sender: Der Kanal kann Daten auf den Bus legen, die von anderen Teilnehmern gelesen werden können.
- Sender/Empfänger: Der Kanal kann Daten auf den Bus legen und gleichzeitig zurücklesen. Dadurch ist eine Überprüfung der ausgegebenen Daten möglich.

Siehe auch

[CHECK_SHIFT_REG](#), [GET_RS](#), [READ_FIFO](#), [RS_INIT](#), [RS_RESET](#), [SET_RS](#), [WRITE_FIFO](#)

Gültig für Module

RS485-2, RS485-4

Beispiel

Siehe Beispiel „[RS485: Empfangen und senden](#)“ auf [Seite 341](#).

SET_RS schreibt einen Wert in ein bestimmtes Register des angegebenen Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
SET_RS (module, REGISTER, value)
```

Parameter

module	Eingestellte Moduladresse (1...255).	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

[CHECK_SHIFT_REG](#), [GET_RS](#), [READ_FIFO](#), [RS_INIT](#), [RS_RESET](#), [RS485_SEND](#), [WRITE_FIFO](#)

Gültig für Module

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

-/-

SET_RS

WRITE_FIFO

WRITE_FIFO schreibt einen Wert in den Sende-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
ret_val = WRITE_FIFO(module, channel, value)
```

Parameter

module	Eingestellte Moduladresse (1...255).	LONG
channel	Kanalnummer, dessen Sende-FIFO beschrieben wird (1, 2 oder 1...4).	LONG
value	Wert der ins Sende-FIFO geschrieben werden soll.	LONG
ret_val	Statusmeldung: 0: Daten wurden erfolgreich geschrieben. 1: Daten konnten nicht geschrieben werden, Sende-FIFO ist voll.	LONG

Bemerkungen

Die Anweisung prüft zuerst, ob noch mindestens ein Speicherplatz im Sende-FIFO frei ist. Ist dies der Fall, wird der übergebene Wert ins FIFO geschrieben (Rückgabewert 0); anderenfalls wird eine 1 zurückgeliefert, die angibt, dass das FIFO voll ist und ein Schreiben nicht möglich war.

Der zu übertragende Wert *value* kann auch ein einzelnes ASCII-Zeichen oder ein ASCII-Befehl sein (Zeichen werden intern mit dem Datentyp Long gleich gesetzt). Die Hardware-Dokumentation enthält ein Beispiel für das Senden einer Zeichenfolge.

Siehe auch

[CHECK_SHIFT_REG](#), [GET_RS](#), [READ_FIFO](#), [RS_INIT](#), [RS_RESET](#), [RS485_SEND](#), [SET_RS](#)

Gültig für Module

RS232-2, RS232-4, RS422-2, RS422-4, RS485-2, RS485-4

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc  
DIM val AS LONG  
  
INIT:  
  RS_RESET(1)  
  RS_INIT(1,1,9600,0,8,0,1) 'Initialisierung von Kanal 1 auf  
                             'Modul 1 mit 9600 Baud, keine Parität,  
                             '8 Datenbits, 1 Stoppbit und  
                             'Hardware-Handshake (nur RS232).  
  
EVENT:  
  PAR_1 = WRITE_FIFO(1,1,val)  
  REM Wenn das FIFO-Feld nicht voll ist, wird val ins FIFO-Feld  
  REM geschrieben. Anderenfalls enthält PAR_1 den Wert 1 und zeigt  
  REM damit an, dass das FIFO-Feld nicht beschrieben werden konnte  
  REM (FIFO voll).
```

Siehe auch weitere Beispiele für RS232 und RS485 ab Seite 336.



LS_DIO_INIT

LS_DIO_INIT initialisiert ein Modul vom Typ HSM-24V am LS-Bus über eine Schnittstelle des Pro-Moduls und gibt den Fehlerstatus zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = LS_DIO_INIT(module, channel, ls-module)
```

Parameter

<code>module</code>	Eingestellte Adresse des Pro-Moduls (1...255).	LONG
<code>channel</code>	Nummer (1, 2) der LS-Bus-Schnittstelle auf dem Pro-Modul.	LONG
<code>ls-module</code>	Eingestellte Moduladresse am LS-Bus (1...15).	LONG
<code>ret_val</code>	Bitmuster, das den Fehlerstatus angibt. Bit = 0: kein Fehler. Bit = 1: Fehler aufgetreten.	LONG

Bit-Nr.	31...8	7	6	5...4	3	2	1	0
Status	–	Temp2	Temp1	–	WD	Time	Ovr	Par

- :don't care (mit 0CFh ausmaskieren)

Par: Parity-Fehler bei der Datenübertragung auf dem LS-Bus.

Ovr: Overrun-Fehler bei der Datenübertragung auf dem LS-Bus.

Time: Timeout-Fehler bei der Datenübertragung auf dem LS-Bus.

WD: Watchdog hat ausgelöst. Die Kanaltreiber sind deaktiviert.

Temp1: Übertemperatur am Treiber für Kanäle 1...16. Treiber ist deaktiviert.

Temp2: Übertemperatur am Treiber für Kanäle 17...32. Treiber ist deaktiviert.

Bemerkungen

Die Anweisung soll nur im Abschnitt **INIT**: verwendet werden, weil sie eine lange Ausführungszeit hat.

Die Initialisierung setzt folgende Einstellungen:

- Alle DIO-Kanäle werden als Eingang programmiert.
Andere Einstellungen siehe **LS_DIGPROG**.
- Der Status für Überstrom (> ca. 500mA) wird zurückgesetzt.
- Der Fehlerstatus für Übertemperatur wird zurückgesetzt.
- Der Fehlerstatus für Timeout auf dem LS-Bus wird zurückgesetzt.

Der Fehler „Übertemperatur“ eines Treibers kann nur auftreten, wenn auf mehreren Kanälen gleichzeitig ein Überstrom im Bereich von 150...500mA anliegt. Unabhängig davon wird bei einem Überstrom über 500mA der betroffene Kanal automatisch abgeschaltet.

Die Kanäle des Moduls HSM-24V dürfen nur im Bereich von 0...150mA betrieben werden.

Gültig für Module

LS-2 Rev. A

Siehe auch

[LS_DIGPROG](#), [LS_WATCHDOG_INIT](#), [LS_DIG_IO](#)



Beispiel

```
REM Example prozess for one module HSM-24V and ADwin-Pro-LS2
#include ADwinPRO_ALL.inc

INIT:
    PROCESSDELAY = 4000000    '10Hz HP
    PAR_1 = LS_DIO_INIT(1,2,1)
    PAR_2 = LS_DIGPROG(1,2,1,0Fh) 'channels 1...32 as output
    PAR_3 = LS_WATCHDOG_INIT(1,2,1,1,1100) 'watchdog time 1.1 sec

EVENT:
    REM set one channel to high, rotating from 1 to 32
    INC PAR_10
    IF (PAR_10>=32) THEN PAR_10=0
    PAR_11 = SHIFT_LEFT(1,PAR_10)
    REM set channels and read back real state
    PAR_12 = LS_DIG_IO(1,2,PAR_11)
```

LS_DIGPROG

LS_DIGPROG programmiert die digitalen Kanäle 1...32 eines Moduls vom Typ HSM-24V am LS-Bus über eine Schnittstelle des Pro-Moduls in Gruppen zu 8 als Ein- oder Ausgang.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc  
  
ret_val = LS_DIGPROG(module, channel, ls-module,  
                    pattern)
```

Parameter

module	Eingestellte Adresse des Pro-Moduls (1...255).	LONG
channel	Nummer (1, 2) der LS-Bus-Schnittstelle auf dem Pro-Modul.	LONG
ls-module	Eingestellte Moduladresse (1...15) am LS-Bus.	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.	–	32:25	24:17	16:9	8:1

ret_val	Bitmuster, das den Fehlerstatus angibt. Bit = 0: kein Fehler. Bit = 1: Fehler aufgetreten.	LONG
---------	--	------

Bit-Nr.	31...8	7	6	5...4	3	2	1	0
Status	–	Temp2	Temp1	–	WD	Time	Ovr	Par

- :don't care (mit 0CFh ausmaskieren)

Par:Parity-Fehler bei der Datenübertragung auf dem LS-Bus.

Ovr: Overrun-Fehler bei der Datenübertragung auf dem LS-Bus.

Time: Timeout-Fehler bei der Datenübertragung auf dem LS-Bus.

WD: Watchdog hat ausgelöst. Die Kanaltreiber sind deaktiviert.

Temp1: Übertemperatur am Treiber für Kanäle 1...16. Treiber ist deaktiviert.

Temp2: Übertemperatur am Treiber für Kanäle 17...32. Treiber ist deaktiviert.

Bemerkungen

Die Anweisung soll nur im Abschnitt **INIT**: verwendet werden, weil sie eine lange Ausführungszeit hat.

Nach der Initialisierung mit **LS_DIO_INIT** 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).

Gültig für Module

LS-2 Rev. A

Siehe auch

[LS_DIO_INIT](#), [LS_WATCHDOG_INIT](#), [LS_DIG_IO](#)

Beispiel

```
REM Example prozess for one module HSM-24V and ADwin-Pro-LS2
#include ADwinPRO_ALL.inc

INIT:
    PROCESSDELAY = 4000000    '10Hz HP
    PAR_1 = LS_DIO_INIT(1,2,1)
    PAR_2 = LS_DIGPROG(1,2,1,0Fh) 'channels 1...32 as output
    PAR_3 = LS_WATCHDOG_INIT(1,2,1,1,1100) 'watchdog time 1.1 sec

EVENT:
    REM set one channel to high, rotating from 1 to 32
    INC PAR_10
    IF (PAR_10>=32) THEN PAR_10=0
    PAR_11 = SHIFT_LEFT(1,PAR_10)
    REM set channels and read back real state
    PAR_12 = LS_DIG_IO(1,2,PAR_11)
```

LS_WATCHDOG_INIT

LS_WATCHDOG_INIT aktiviert oder deaktiviert den Watchdog-Zähler eines Moduls am LS-Bus über eine Schnittstelle des Pro-Moduls.

Beim Aktivieren erhält der Zähler seinen Startwert und wird gestartet.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = LS_WATCHDOG_INIT(module, channel,
                             ls-module, enable, time)
```

Parameter

<code>module</code>	Eingestellte Adresse des Pro-Moduls (1...255).	LONG
<code>channel</code>	Nummer (1, 2) der LS-Bus-Schnittstelle auf dem Pro-Modul.	LONG
<code>ls-module</code>	Eingestellte Moduladresse am LS-Bus (1...15).	LONG
<code>enable</code>	Status des Watchdog-Zählers einstellen: 0 : Watchdog-Zähler deaktivieren. 1 : Watchdog-Zähler aktivieren.	LONG
<code>time</code>	Auslösezeit (0...107374) des Zählers in Millisekunden.	LONG
<code>ret_val</code>	Bitmuster, das den Fehlerstatus angibt. Bit = 0: kein Fehler. Bit = 1: Fehler aufgetreten.	LONG

Bit-Nr.	31...8	7	6	5...4	3	2	1	0
Status	–	Temp2	Temp1	–	WD	Time	Ovr	Par

- : don't care (mit 0CFh ausmaskieren)
 Par: Parity-Fehler bei der Datenübertragung auf dem LS-Bus.
 Ovr: Overrun-Fehler bei der Datenübertragung auf dem LS-Bus.
 Time: Timeout-Fehler bei der Datenübertragung auf dem LS-Bus.
 WD: Watchdog hat ausgelöst. Die Kanaltreiber sind deaktiviert.
 Temp1: Übertemperatur am Treiber für Kanäle 1...16. Treiber ist deaktiviert.
 Temp2: Übertemperatur am Treiber für Kanäle 17...32. Treiber ist deaktiviert.

Bemerkungen

Die Anweisung soll nur im Abschnitt **INIT**: verwendet werden, weil sie eine lange Ausführungszeit hat.

Solange der Watchdog-Zähler aktiv ist, dekrementiert er den Zählerstand kontinuierlich. Nach der eingestellten Auslösezeit erreicht der Zählerstand 0 (Null). Das Modul nimmt nun eine Fehlfunktion an und wird gestoppt; dadurch werden alle Ausgangssignale zurückgesetzt.

Nach dem Einschalten des Moduls ist der Zähler auf den Startwert 10ms eingestellt und der Watchdog ist aktiv.

Setzen Sie den aktiven Watchdog-Zähler während seines Zählvorgangs mindestens einmal zurück, um die Funktion des Moduls zu gewährleisten. Zum Zurücksetzen können modulspezifische Befehle verwendet werden.

Die Watchdog-Funktion dient zur Verbindungsüberwachung zwischen ADwin-System und LS-Bus-Modul.

Gültig für Module

LS-2 Rev. A



Siehe auch

[LS_DIO_INIT](#), [LS_DIGPROG](#), [LS_DIG_IO](#)

Beispiel

REM Example prozess for one module HSM-24V and ADwin-Pro-LS2

#INCLUDE ADwinPRO_ALL.inc

INIT:

PROCESSDELAY = 4000000 '10Hz HP

PAR_1 = **LS_DIO_INIT**(1,2,1)

PAR_2 = **LS_DIGPROG**(1,2,1,0Fh) 'channels 1...32 as output

PAR_3 = **LS_WATCHDOG_INIT**(1,2,1,1,1100) 'watchdog time 1.1 sec

EVENT:

REM set one channel to high, rotating from 1 to 32

INC **PAR_10**

IF (**PAR_10**>=32) **THEN** **PAR_10**=0

PAR_11 = **SHIFT_LEFT**(1,**PAR_10**)

REM set channels and read back real state

PAR_12 = **LS_DIG_IO**(1,2,**PAR_11**)

LS_DIG_IO

LS_DIG_IO setzt alle Digital-Ausgänge des Moduls HSM-24V am LS-Bus über eine Schnittstelle des Pro-Moduls auf den Pegel High oder Low und gibt den Zustand aller Kanäle als Bitmuster zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

ret_val = LS_DIG_IO(module, channel, pattern)
```

Parameter

<code>module</code>	Eingestellte Adresse des Pro-Moduls (1...255).	LONG
<code>channel</code>	Nummer (1, 2) der LS-Bus-Schnittstelle auf dem Pro-Modul.	LONG
<code>pattern</code>	Bitmuster, mit dem die digitalen Ausgänge gesetzt werden (siehe Tabelle). Bit = 0: Ausgang auf Pegel Low setzen. Bit = 1: Ausgang auf Pegel High setzen.	LONG
<code>ret_val</code>	Bitmuster mit dem Ist-Zustand aller digitalen Kanäle (siehe Tabelle). Bit = 0: Pegel Low liegt an. Bit = 1: Pegel High liegt an.	LONG

Bitnr.	31	30	29	...	2	1	0
Eingang	32	31	30	...	3	2	1

Bemerkungen

LS_DIG_IO arbeitet nur korrekt, wenn folgende Voraussetzungen erfüllt sind:

- Am LS-Bus ist nur ein Modul angeschlossen.
- Das Modul ist vom Typ HSM-24V.
- Am Modul ist die Moduladresse 1 eingestellt.

Die Kanäle werden mit **LS_DIGPROG** als Ein- oder Ausgänge programmiert.

Das Bitmuster `pattern` wird nur für die Kanäle angewendet, die als Ausgang programmiert sind. Bits für Eingänge werden ignoriert.

Der Rückgabewert enthält den tatsächlichen Schaltzustand sowohl von Eingängen wie von Ausgängen. Beachten Sie: Die Eingänge haben einen Filter mit ca. 12µs Verzögerung.

LS_DIG_IO setzt den Watchdog-Zähler des Moduls auf den Startwert zurück. Der Zähler bleibt aktiv. Der Startwert wird mit **LS_WATCHDOG_INIT** eingestellt.

Setzen Sie den aktiven Watchdog-Zähler während seines Zählvorgangs mindestens einmal zurück, um die Funktion des Moduls zu gewährleisten.

Gültig für Module

LS-2 Rev. A

Siehe auch

[LS_DIO_INIT](#), [LS_DIGPROG](#), [LS_WATCHDOG_INIT](#),



Beispiel

```
REM Example prozess for one module HSM-24V and ADwin-Pro-LS2
#include ADwinPRO_ALL.inc

INIT:
    PROCESSDELAY = 4000000    '10Hz HP
    PAR_1 = LS_DIO_INIT(1,2,1)
    PAR_2 = LS_DIGPROG(1,2,1,0Fh) 'channels 1...32 as output
    PAR_3 = LS_WATCHDOG_INIT(1,2,1,1,1100) 'watchdog time 1.1 sec

EVENT:
    REM set one channel to high, rotating from 1 to 32
    INC PAR_10
    IF (PAR_10>=32) THEN PAR_10=0
    PAR_11 = SHIFT_LEFT(1,PAR_10)
    REM set channels and read back real state
    PAR_12 = LS_DIG_IO(1,2,PAR_11)
```

4 ADbasic-Anweisungen für ADwin-Pro II-Module

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

Im Anhang finden Sie außerdem sortierte Befehlsübersichten:

- [Alphabetische Befehlsübersicht](#)
- [Befehlsübersicht nach Modulen](#)

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

- [Thematische Befehlsübersicht](#)

Befehle für *ADwin-Pro I*- und *ADwin-Pro II*-Module sind meist gleichlautend, jedoch beginnen Pro II-Befehle zur Unterscheidung mit dem Kürzel `P2_`.

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

```
#INCLUDE ADwinPRO_ALL.INC
```

Zu jeder Befehlsbeschreibung gehören

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

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

4.1 Pro II: Alle Module

Alle Pro II-Module, die von aktiven *ADbasic*-Programmen angesprochen werden, müssen korrekt eingesteckt sein. Anderenfalls steigt die Prozessorauslastung an, 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.

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 Module

Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/18 Rev. E, AOut-4/16 Rev. E, AOut-8/16 Rev. E, CPU-T11

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_CHECK_LED

CPU_DIGIN

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

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = CPU_DIGIN(channel)
```

Parameter

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

Bemerkungen

Die Anweisung **CPU_DIGIN** hat nur eine Funktion, wenn der gewählte DIG I/O-Kanal mit **CPU_DIG_IO_CONFIG** als Eingang konfiguriert ist. Mit **CPU_DIG_IO_CONFIG** wird festgelegt, ob **CPU_DIGIN** auf steigende oder auf fallende Flanken reagiert. Nach einem Neustart sind die DIG I/O-Kanäle als Eingang und für fallende Flanken konfiguriert.

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

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

Siehe auch

[CPU_DIGOUT](#), [CPU_DIG_IO_CONFIG](#), [CPU_DIGIN](#) (T9, T10)

Gültig für Module

CPU-T11

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM dummy AS LONG

INIT:
    REM Beide DIG I/O Kanäle als Eingang mit steigender Flanke
    REM einstellen
    CPU_DIG_IO_CONFIG(100010b)
    REM Statusmeldung an DIG I/O 1 lesen und dadurch zurücksetzen
    dummy = CPU_DIGIN(1)

EVENT:
    ...
    IF (CPU_DIGIN(1) = 1) THEN 'Bei steigender Flanke ...
        END                    '... das Programm beenden
    ENDIF
    ...
```


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 Module

CPU-T11

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
```

EVENT:

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

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(module_pattern)
```

Parameter

module_pattern Bitmuster zur Einstellung von Kanal- und Flankentyp am Eingang DIG I/O n:
 Bit = 0: Kanal als Eingang oder Flankentyp fallend.
 Bit = 1: Kanal als Ausgang oder Flankentyp steigend.

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

Bemerkungen

Der Flankentyp kann nur für Eingänge eingestellt werden.

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

Siehe auch

[CPU_DIGIN](#), [CPU_DIGOUT](#), [CPU_EVENT_CONFIG](#)

Gültig für Module

CPU-T11

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM dummy AS LONG

INIT:
  REM Beide DIG I/O Kanäle als Eingang mit steigender Flanke
  REM einstellen
  CPU_DIG_IO_CONFIG(100010b)
  REM Statusmeldung an DIG I/O 1 lesen und dadurch zurücksetzen
  dummy = CPU_DIGIN(1)
  ...
```

CPU_EVENT_CONFIG konfiguriert den **EVENT IN**-Kanal des Prozessormoduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

CPU_EVENT_CONFIG(min_hold, edge, prescale)
```

Parameter

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

Bemerkungen

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

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

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

Siehe auch

[P2_EVENT_CONFIG](#), [P2_EVENT_ENABLE](#), [EVENTENABLE](#)

Gültig für Module

CPU-T11

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
```

INIT:

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

EVENT:

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

...

CPU_EVENT_CONFIG

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, value)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>value</code>	0 : externes Event-Signal sperren (Default). 1 : externes Event-Signal zulassen.	LONG

Bemerkungen

Sie können einen hochprioren *ADbasic*-Prozess (d.h. dessen zyklischen Abschnitt **EVENT** :) durch ein externes Event-Signal aufrufen lassen und damit z.B. mit einem externen Prozess synchronisieren (vgl. *ADbasic*-Handbuch).

Die meisten Module verfügen über einen Event-Eingang. Konfigurieren Sie den Event-Eingang zuerst mit **P2_EVENT_CONFIG**. Sobald Sie mit **P2_EVENT_ENABLE** den Eingang aktiviert haben, wird ein anliegendes Signal an das Prozessormodul weitergeleitet. Das Prozessormodul erkennt den eingestellten Flankentyp (positiv oder negativ) als Event-Signal und der eingestellte Prozess reagiert.

Beachten Sie bei Modulen mit mehreren Event-Eingängen die Einstellung mit **P2_EVENT2_CONFIG**. Die Konfiguration von **P2_EVENT_CONFIG** bezieht sich dann auf das resultierende Event-Signal.

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

In einem System darf – zusätzlich zum Prozessor-Modul – nur ein einziger Event-Eingang aktiv sein, d.h. Sie müssen einen eventuell aktiven Event-Eingang sperren, bevor Sie den Event-Eingang an einem anderen Modul aktivieren.

Siehe auch

[P2_EVENT_CONFIG](#), [P2_EVENT2_CONFIG](#), [P2_EVENT_READ](#)

Gültig für Module

Aln-F-8/14 Rev. E, Aln-F-8/18 Rev. E, AOut-4/16 Rev. E, AOut-8/16 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 eine Flanke anliegen muss, damit sie akzeptiert wird: 0: 15ns (Default). 1: 50ns.	LONG
edge	Flankentyp, der akzeptiert wird: 1: positive Flanke (Default). 2: negative Flanke. 3: positive und negative Flanke.	LONG
prescale	Anzahl (1...255) an Flanken, nach der ein Event-Signal erzeugt wird (Default: 1).	LONG

Bemerkungen

Ein Event-Eingang muss mit der Anweisung **P2_EVENT_ENABLE** aktiviert werden, damit ein anliegendes Signal verarbeitet werden kann. Konfigurieren Sie den Event-Eingang zuerst mit **P2_EVENT_CONFIG** und aktivieren den Eingang danach.

Beachten Sie bei Modulen mit mehreren Event-Eingängen die Einstellung mit **P2_EVENT2_CONFIG**. Die Konfiguration von **P2_EVENT_CONFIG** bezieht sich dann auf das resultierende Event-Signal.



Siehe auch

[P2_EVENT_ENABLE](#), [P2_EVENT2_CONFIG](#), [P2_EVENT_READ](#)

Gültig für Module

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

Parameter

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

Bemerkungen

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

Das resultierende Event-Signal wird mit **P2_EVENT_CONFIG** konfiguriert. **P2_EVENT_ENABLE** gibt das resultierende Event-Signal für die Steuerung des Event-Prozesses frei.

Je nach Modul stehen verschiedene Modi zur Verfügung:

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

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

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

Ein Aufruf von **P2_EVENT2_CONFIG** mit **mode**=0 sperrt den Eingang **EVENT/A** wieder.

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

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

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

Ohne Vorverarbeitung

Signal nach
Freigabeimpuls

Signal aus AB-Modus

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

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

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

Siehe auch

[P2_EVENT_ENABLE](#), [P2_EVENT_CONFIG](#), [P2_EVENT_READ](#)

Gültig für Module

Aln-F-8/14 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 nach Freigabe aus
AB-Modus**

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

<code>module</code>	Moduladresse (0...15): 0: CPU-Modul. 1...15: Eingestellte Moduladresse.	LONG
<code>ret_val</code>	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

Siehe auch

[P2_EVENT_ENABLE](#), [P2_EVENT_CONFIG](#), [P2_EVENT2_CONFIG](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
```

INIT:

```
REM Event-Eingang am Modul 1 (AIIn-F-8/14) konfigurieren für  
REM Mindestzeit 15 ns, neg. Flanken, 4 Flanken  
P2_EVENT_READ(1,0,2,4)  
REM Externes Event-Signal am Modul 1 freigeben  
P2_EVENT_ENABLE(1,1)
```


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,pattern)
```

Parameter

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

Siehe auch

[P2_CHECK_LED](#)

Gültig für Module

Aln-32/18 Rev. E, Aln-8/18 Rev. E, Aln-F-8/14 Rev. E, Aln-F-8/18 Rev. E, AOut-4/16 Rev. E, AOut-8/16 Rev. E, CPU-T11

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
INIT:
    P2_SET_LED(1,1)          'LED am Modul 1 einschalten

EVENT:
    ...

FINISH:
    P2_SET_LED(1,0)          'LED am Modul 1 ausschalten
```

P2_SET_LED

P2_SYNC_ALL

P2_SYNC_ALL startet auf den angegebenen Modulen synchron eine bestimmte Aktion.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

P2_SYNC_ALL(module_pattern)
```

Parameter

module_pattern Bitmuster zum Ansprechen der Modul-LONG adressen, die synchron starten sollen:
Bit = 0: Modul ignorieren.
Bit = 1: Modul synchron starten.

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

Bemerkungen

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

Modultyp	Aktion
Analog-Eingang	A/D-Wandlung auf allen freigegebenen ADC starten, siehe P2_START_CONV / P2_START_CONVF . Die Wandlung kann eine Einzelmessung oder Teil einer Burst-Messreihe sein.
Analog-Ausgang	D/A-Wandlung auf allen freigegebenen DAC starten mit dem Wert aus dem DAC-Register, siehe P2_START_DAC

Als Voreinstellung nehmen alle Ein- oder Ausgänge der gewählten Module an der Aktion teil. Mit **P2_SYNC_ENABLE** können Sie ein oder mehrere Ein- oder Ausgänge eines Moduls für die Synchronaktion sperren oder freigeben.

Mit den folgenden Befehlen können ebenfalls Module synchron angesprochen werden:

- **P2_ADCF_MODE**: Automatische Wandlung auf mehreren Modulen starten.
- **P2_BURST_START**: Burst-Messreihe auf mehreren Modulen starten.

Siehe auch

[P2_SYNC_ENABLE](#), [P2_SYNC_STAT](#)

[P2_ADCF_MODE](#), [P2_BURST_START](#), [P2_START_CONV](#), [P2_START_CONVF](#), [P2_START_DAC](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM i AS LONG
DIM DATA_1[1000], DATA_2[1000], DATA_3[1000] AS LONG
DIM outvall[1000] AS LONG

INIT:
  REM Kanäle 1+2 auf den Modulen 1, 2 und 4 aktivieren,
  REM alle anderen deaktivieren
  P2_SYNC_ENABLE(1,11b)
  P2_SYNC_ENABLE(2,11b)
  P2_SYNC_ENABLE(4,11b)
  i=1                                'Index initialisieren

EVENT:
  REM Wandlung für Module 1,2,4 und 5 synchron starten
  P2_SYNC_ALL(11011b)
  P2_WAIT_EOC(1)                    'Auf des Ende der Wandlung warten
  DATA_1[i]=P2_READ_ADC(1) 'A/D Wandler Modul 1 auslesen
  DATA_2[i]=P2_READ_ADC(2) 'A/D Wandler Modul 2 auslesen
  DATA_3[i]=P2_READ_ADC(4) 'A/D Wandler Modul 4 auslesen
  REM Wert in Ausgangsregister 1 des D/A Moduls 5 schreiben
  P2_WRITE_DAC(5,1,outval[i])
  IF (i=1000) THEN END              'Ende nach 1000 Durchläufen
  INC(i)                             'Index erhöhen
```

P2_SYNC_ENABLE

P2_SYNC_ENABLE aktiviert oder deaktiviert die gewählten Ein- oder Ausgänge 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 Ein- oder Ausgänge, die aktiviert oder deaktiviert werden:
 Bit = 0: deaktivieren.
 Bit = 1: aktivieren

Bits in <code>channel</code>	31:08	07	06	05	04	03	02	01	00
Kanalnr. in Analog-Eingangsmodulen	–	8	7	6	5	4	3	2	1
Pro-Aln-F-x/x Rev. E									
Kanalnr. in Analog-Ausgangsmodulen	–	8	7	6	5	4	3	2	1

Bemerkungen

Nach dem Einschalten des Geräts ist für alle Module der Wert `0FFFFh` voreingestellt, d.h. alle Ein- oder Ausgänge sind aktiviert.

Der Befehl beeinflusst alle Kanäle des Moduls gleichzeitig. Wenn Sie nur einen einzigen Kanal aktivieren oder deaktivieren möchten, müssen Sie den Zustand der übrigen Kanäle unverändert mit angeben.

Das Synchron-Signal wird mit **P2_SYNC_ALL** ausgelöst.

Siehe auch

[P2_SYNC_ALL](#), [P2_SYNC_STAT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM i AS LONG
DIM DATA_1[1000], DATA_2[1000], DATA_3[1000] AS LONG
DIM outval[1000] AS LONG

INIT:
  REM Kanäle 1+2 auf den Modulen 1, 2 und 4 aktivieren,
  REM alle anderen deaktivieren
  P2_SYNC_ENABLE(1,11b)
  P2_SYNC_ENABLE(2,11b)
  P2_SYNC_ENABLE(4,11b)
  i=1                                'Index initialisieren

EVENT:
  REM Wandlung für Module 1, 2, 4 und 5 synchron starten
  P2_SYNC_ALL(11011b)
  P2_WAIT_EOC(1)                    'Auf des Ende der Wandlung warten
  DATA_1[i]=P2_READ_ADC(1) 'A/D Wandler Modul 1 auslesen
  DATA_2[i]=P2_READ_ADC(2) 'A/D Wandler Modul 2 auslesen
  DATA_3[i]=P2_READ_ADC(4) 'A/D Wandler Modul 4 auslesen
  REM Wert in Ausgangsregister des D/A Moduls 5 schreiben
  P2_WRITE_DAC(5,1,outval[i])
  IF (i=1000) THEN END              'Ende nach 1000 Durchläufen
  INC(i)                            'Index erhöhen
```

P2_SYNC_STAT

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 auf den Ein- oder Ausgängen: Bit = 0: Kanal ist deaktiviert. Bit = 1: Kanal ist aktiviert.	LONG

Bits in channel	31:08	07	06	05	04	03	02	01	00
Kanalnr. in Analog-Eingangsmodulen Pro-Aln-F-x/x Rev. E	–	8	7	6	5	4	3	2	1
Kanalnr. in Analog-Ausgangsmodulen Pro-AOut-x/16 Rev. E	–	8	7	6	5	4	3	2	1

Bemerkungen

Sie stellen die Synchronoption der Kanäle mit **P2_SYNC_ENABLE** ein.

Siehe auch

[P2_SYNC_ALL](#), [P2_SYNC_ENABLE](#)

Gültig für Module

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

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
```

4.2 Pro II: Eingangsmodule

Die Include-Datei `ADwinPRO_ALL.INC` beinhaltet alle Funktionen und Prozeduren, die zum Ansprechen der *ADwin-Pro II*-Eingangsmodule benötigt werden. Fügen Sie deshalb die folgende *ADbasic*-Anweisung in Ihr Programm ein:

```
#INCLUDE ADwinPRO_ALL.INC
```

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

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



P2_ADC

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

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = P2_ADC(module, input_no)
```

Parameter

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

Bemerkungen

Die Anweisung **P2_ADC** ist eine Zusammenstellung von aufeinander folgenden Funktionen:

P2_SET_MUX	...	P2_START_CONV	P2_WAIT_EOC	P2_READ_ADC
Multiplexer auf einen Eingangskanal setzen	Einschwingen des Multiplexers abwarten	A/D-Wandlung starten	Ende der Wandlung abwarten	Gewandelten Wert auslesen

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

Siehe auch

P2_ADC24, P2_ADC_READ_LIMIT, P2_ADC_SET_LIMIT, P2_READ_ADC, P2_START_CONV, P2_WAIT_EOC

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value AS LONG
```

EVENT:

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


P2_ADC24 führt eine komplette Messung auf einem ADC des angegebenen Moduls durch. Der Rückgabewert hat 24 Bit Auflösung.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC
ret_val = P2_ADC24(module, input_no)
```

Parameter

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

Bemerkungen

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

P2_SET_MUX	→	...	→	P2_START_CONV	→	P2_WAIT_EOC	→	P2_READ_ADC24
Multiplexer auf einen Eingangskanal setzen		Einschwingen des Multiplexers abwarten		A/D-Wandlung starten		Ende der Wandlung abwarten		Gewandelten Wert auslesen

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

Siehe auch

P2_ADC, P2_ADC_READ_LIMIT, P2_ADC_SET_LIMIT, P2_READ_ADC24, P2_START_CONV, P2_WAIT_EOC

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value AS LONG

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

P2_ADC24

P2_ADC_READ_LIMIT

P2_ADC_READ_LIMIT liest die Flags für Grenzwertüber- und unterschreitungen auf 16 ADC 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 Grenzwert-LONG über- und unterschreitungen:

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

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

Bemerkungen

Sie stellen die Grenzwerte mit **P2_ADC_SET_LIMIT** ein.

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

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

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_ADC_SET_LIMIT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM flags AS LONG

INIT:
  P2_SE_DIFF(1,1)           'Differentielle Eingänge
  P2_ADC_SET_LIMIT(1, 2, 42768,256) 'Grenzwerte Kanal 2 setzen
  P2_SEQ_INIT(1,3,0,10b,0) 'continuous max mode, Kanal 2
  P2_SEQ_START(1)           'Messreihe starten
  P2_SEQ_WAIT(1)
  flags = P2_ADC_READ_LIMIT(1,1) 'Flags durch Lesen rücksetzen

EVENT:
  flags = P2_ADC_READ_LIMIT(1,1) 'Flags der Kanäle 1...16 lesen
  IF ((flags AND 10b) = 10b) THEN
    REM Untergrenze auf Kanal 2 ist unterschritten
    INC PAR_1
  ENDIF
  IF ((flags AND 20000h) = 20000h) THEN
    REM Obergrenze auf Kanal 2 ist überschritten
    INC PAR_2
  ENDIF
```

P2_ADC_SET_LIMIT

P2_ADC_SET_LIMIT setzt den oberen und unteren Grenzwert für einen ADC 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 Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC  
DIM flags AS LONG  
  
INIT:  
  P2_SE_DIFF(1,1) 'Differentielle Eingänge  
  P2_ADC_SET_LIMIT(1, 2, 42768,256) 'Grenzwerte Kanal 2 setzen  
  P2_SEQ_INIT(1,3,0,10b,0) 'continuous max mode, Kanal 2  
  P2_SEQ_START(1) 'Messreihe starten  
  P2_SEQ_WAIT(1)  
  flags = P2_ADC_READ_LIMIT(1,1) 'Flags durch Lesen rücksetzen  
  
EVENT:  
  flags = P2_ADC_READ_LIMIT(1,1) 'Flags der Kanäle 1...16 lesen  
  IF ((flags AND 10b) = 10b) THEN  
    REM Untergrenze auf Kanal 2 ist unterschritten  
    INC PAR_1  
  ENDIF  
  IF ((flags AND 20000h) = 20000h) THEN  
    REM Obergrenze auf Kanal 2 ist überschritten  
    INC PAR_2  
  ENDIF
```

P2_READ_ADC liest das Wandlungsergebnis des angegebenen Moduls aus. Das Ergebnis hat 16 Bit Auflösung.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC
ret_val = P2_READ_ADC(module)
```

Parameter

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

Bemerkungen

-/-

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_START_CONV](#), [P2_WAIT_EOC](#), [P2_ADC_READ_LIMIT](#), [P2_ADC_SET_LIMIT](#), [P2_READ_ADC_SCONV](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value1 AS LONG 'Deklaration

EVENT:
P2_START_CONV(1,1) 'Start AD-Wandlung
P2_WAIT_EOC(1,1) 'Warten auf Wandlung-Ende
value1 = P2_READ_ADC(1,1) 'Wert vom ADC einlesen
```

P2_READ_ADC

P2_READ_ADC24

P2_READ_ADC24 liest das Wandlungsergebnis des angegebenen Moduls aus. Das Ergebnis hat 24 Bit Auflösung.

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

-/-

Siehe auch

[P2_ADC24](#), [P2_START_CONV](#), [P2_WAIT_EOC](#), [P2_ADC_READ_LIMIT](#), [P2_ADC_SET_LIMIT](#), [P2_READ_ADC_SCONV24](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC  
DIM value1 AS LONG 'Deklaration  
  
EVENT:  
  P2_START_CONV(1,1) 'Start AD-Wandlung  
  P2_WAIT_EOC(1,1) 'Warten auf Wandlung-Ende  
  value1 = P2_READ_ADC24(1,1) '24Bit-Wert vom ADC einlesen
```

P2_READ_ADC_SCONV liest das Wandlungsergebnis des angegebenen Moduls aus und startet sofort eine neue Konvertierung.

Der Rückgabewert hat eine Auflösung von 16 Bit.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = P2_READ_ADC_SCONV(module)
```

Parameter

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

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_READ_ADC](#), [P2_READ_ADC_SCONV24](#),
[P2_START_CONV](#), [P2_WAIT_EOC](#)

Gültig für Module

Aln-32/18 Rev. E, Aln-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_CONV(1,1) 'A/D-Wandler starten

EVENT:
  P2_WAIT_EOC(1,1)
  DATA_1[i] = P2_READ_ADC_SCONV(1,1) 'A/D-Wandler auslesen +
                                     'starten
  INC(i) 'Index erhöhen
  IF (i=1001) THEN END 'Nach 1000 Messwerten Prozess beenden
```

P2_READ_ADC_SCONV

P2_READ_ADC_SCONV24

P2_READ_ADC_SCONV24 liest das Wandlungsergebnis des angegebenen Moduls aus und startet sofort eine neue Konvertierung.

Der Rückgabewert hat eine Auflösung von 24 Bit.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC  
  
ret_val = P2_READ_ADC_SCONV24 (module)
```

Parameter

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

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_READ_ADC](#), [P2_READ_ADC_SCONV](#), [P2_START_CONV](#), [P2_WAIT_EOC](#)

Gültig für Module

Aln-32/18 Rev. E, Aln-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_CONV(1,1) 'A/D-Wandler starten  
  
EVENT:  
  P2_WAIT_EOC(1,1)  
  DATA_1[i] = P2_READ_ADC_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_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, choice)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
choice	Betriebsart der Analog-Eingänge. 0: single ended. 1: differentiell (Default).	LONG

Bemerkungen

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

Siehe auch

[P2_ADC](#)

Gültig für Module

Aln-32/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
```

```
INIT:
```

```
P2_SE_DIFF(1,0)      'Modul mit der Adresse 1 wird
                     'auf SE gesetzt
P2_SE_DIFF(2,1)      'Modul mit der Adresse 2 wird
                     'auf DIFF gesetzt
```

P2_SE_DIFF

P2_SEQ_INIT

P2_SEQ_INIT initialisiert das angegebene Modul für den Betrieb mit Ablaufsteuerung.

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

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
```

```
P2_SEQ_INIT(module, mode, gain, channels, mux_time)
```

Parameter

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

mode Arbeitsmodus der Ablaufsteuerung: LONG
 0: Einzelmessung (Default), ohne Ablaufsteuerung.
 1: Modus „single shot“, einfacher Messzyklus.
 2: Modus „continuous“, regelmäßiger Messzyklus.
 3: Modus „continuous max“ Messzyklus mit maximaler Geschwindigkeit.

gain Verstärkungsfaktor (nur für die Modi 1...3): LONG
 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.

channels Bitmuster, das die zu wandelnden Kanäle, die Messgruppe, bestimmt. LONG
 Bit = 0: Kanal nicht wandeln.
 Bit = 1: Kanal wandeln.

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

muxtime Anzahl der Zeiteinheiten, aus der sich die Einschwingzeit der Ablaufsteuerung ergibt: LONG
 0: Werkseinstellung (125 = 2,5µs).
 125...2³¹: Einschwingzeit in Einheiten von 20ns.

Bemerkungen

Nach dem Einschalten ist der Modus 0 aktiv.

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

Die Modi unterscheiden sich wie folgt:

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

- 1 single shot: Die Ablaufsteuerung wird mit **P2_SEQ_START** gestartet; die Ablaufsteuerung endet, sobald die gewählten Kanäle je einmal gewandelt sind.
Das Ende der Ablaufsteuerung wird mit **P2_SEQ_WAIT** abgefragt und die Messwerte mit **P2_SEQ_READ** eingelesen.
- 2 continuous: Die Ablaufsteuerung wandelt für jeden Prozesszyklus einen Satz neuer Messwerte.
Die Wandlung wird mit **P2_SEQ_START** gestartet. Letzter Befehl im Abschnitt INIT.; danach das PROCESSDELAY nicht mehr ändern. Das Wandlungsende (für alle Kanäle) wird automatisch mit dem Beginn des nächsten Prozesszyklus synchronisiert. Daher können – und sollten auch – alle Messwerte bereits zu Beginn des Prozesszyklus mit **P2_SEQ_READ** gelesen werden.
- 3 continuous max: Die Ablaufsteuerung wandelt ununterbrochen neue Messwerte an den gewählten Kanälen, d.h. Wandlung und Prozesszyklus laufen asynchron.
Die Wandlung wird mit **P2_SEQ_START** gestartet. Im Prozesszyklus wird mit **P2_SEQ_READ** der jeweils neueste Messwert gelesen.

Sie können in der Messgruppe eine beliebige Auswahl aus den Kanälen des Moduls zusammenstellen. Die Kanäle einer Messgruppe werden automatisch in aufsteigender Reihenfolge der Kanalnummern sortiert, d.h. die Ablaufsteuerung wandelt den Kanal mit der niedrigsten Nummer zuerst.

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

Bei den 32kanaligen Modulen müssen die Eingänge mit **P2_SE_DIFF** als single ended oder differentiell eingestellt werden.

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

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

Die Einschwingzeit berechnet sich nach folgender Formel:

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

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

Siehe auch

P2_ADC, **P2_SEQ_READ**, **P2_SEQ_READ24**, **P2_SEQ_READ_PACKED**, **P2_SEQ_START**, **P2_SEQ_WAIT**

Gültig für Module

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



Beispiel

```
#DEFINE module 1
#include ADwinPRO_ALL.inc

DIM DATA_1[16] AS LONG AT DM_LOCAL

INIT:
  P2_SE_DIFF(module,0)      'Eingänge auf single ended stellen
  REM Ablaufsteuerung: Continuous Mode, Verstärkungsfaktor 1
  REM ungeradzahlige Kanäle des Moduls AIN-32
  REM Standard-Einschwingzeit
  P2_SEQ_INIT(module,3,0,55555555h,0)
  REM Messsequenzen auf dem Modul starten
  P2_SEQ_START(SHIFT_LEFT(1,module-1))
  P2_SEQ_WAIT(module)       'Warten, bis alle angegebenen Kanäle
                             'einmal gemessen sind

EVENT:
  REM Messwerte lesen und in DATA_1 kopieren
  P2_SEQ_READ(module,16,DATA_1,1)
```

P2_SEQ_READ kopiert eine bestimmte Anzahl an Messwerten (16 Bit) von dem angegebenen Modul in ein Ziel-Feld.

In jedes Feldelement wird jeweils 1 Messwert kopiert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

P2_SEQ_READ(module, count, array[], array_idx)
```

Parameter

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

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **P2_SEQ_INIT** die Ablaufsteuerung des Moduls aktiviert wurde.

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

Siehe auch

[P2_SEQ_INIT](#), [P2_SEQ_READ](#), [P2_SEQ_READ24](#), [P2_SEQ_READ_PACKED](#), [P2_SEQ_START](#), [P2_SEQ_WAIT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[16] AS LONG AT DM_LOCAL

INIT:
  P2_SE_DIFF(1,0)           'Single-Ended Eingänge
  REM Ablaufsteuerung: Continuous Mode, Verstärkungsfaktor 1
  REM ungeradzahlige Kanäle, Standard-Einschwingzeit
  P2_SEQ_INIT(1,3,0,55555555h,0)
  P2_SEQ_START(1)           'Messsequenzen starten
  P2_SEQ_WAIT(1)            'Warten, bis einmal alle angegebenen
                              'Kanäle gemessen wurden

EVENT:
  P2_SEQ_READ(1,16,DATA_1,1) 'Aktuelle Messwerte von dem Modul
                              'in DATA_1 umkopieren
```

P2_SEQ_READ

P2_SEQ_READ24

P2_SEQ_READ24 kopiert eine bestimmte Anzahl an Messwerten (24 Bit) von dem angegebenen Modul in ein Ziel-Feld.

In jedes Feldelement wird jeweils 1 Messwert kopiert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

P2_SEQ_READ24(module, count, array[], array_idx)
```

Parameter

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

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **P2_SEQ_INIT** die Ablaufsteuerung des Moduls aktiviert wurde.

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

Siehe auch

[P2_SEQ_INIT](#), [P2_SEQ_READ](#), [P2_SEQ_READ_PACKED](#), [P2_SEQ_START](#), [P2_SEQ_WAIT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[16] AS LONG AT DM_LOCAL

INIT:
    P2_SE_DIFF(1,0)           'Single-Ended Eingänge
    REM Ablaufsteuerung: Continuous Mode, Verstärkungsfaktor 1
    REM ungeradzahlige Kanäle, Standard-Einschwingzeit
    P2_SEQ_INIT(1,3,0,5555555h,0)
    P2_SEQ_START(1)           'Messesequenzen starten
    P2_SEQ_WAIT(1)            'Warten, bis einmal alle angegebenen
                                'Kanäle gemessen wurden

EVENT:
    P2_SEQ_READ24(1,16,DATA_1,1) 'Aktuelle Messwerte von dem Modul
                                'in DATA_1 umkopieren
```

P2_SEQ_READ_PACKED kopiert eine gerade Anzahl an Messwerten (16 Bit) paarweise von dem angegebenen Modul in ein Ziel-Feld.
In jedes Feldelement werden jeweils 2 Messwerte kopiert.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc

P2_SEQ_READ_PACKED (module, count, array[], array_idx)
```

Parameter

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

Bemerkungen

Diese Anweisung ist nur sinnvoll einsetzbar, wenn vorher mit **P2_SEQ_INIT** die Ablaufsteuerung des Moduls aktiviert wurde.

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

Siehe auch

[P2_SEQ_INIT](#), [P2_SEQ_READ](#), [P2_SEQ_READ24](#), [P2_SEQ_START](#), [P2_SEQ_WAIT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc

DIM DATA_1[32], DATA_2[32] AS LONG AT DM_LOCAL

INIT:
  P2_SE_DIFF(5,0)           'Single-Ended Eingänge
  REM Modul 1+5: Ablaufsteuerung auf Continuous Mode,
  REM Verstärkungsfaktor 1, geradzahlige Kanäle des Moduls,
  REM Standard-Einschwingzeit
  P2_SEQ_INIT(1,3,0,0AAAAAAAh,0)
  P2_SEQ_INIT(5,3,0,0AAAAAAAh,0)
  P2_SEQ_START(10001b)      'Messsequenz auf Modulen 1+5 starten
  P2_SEQ_WAIT(1)           'Warten bis einmal alle angegebenen
                           'Kanäle 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_1,1)
```

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

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

Siehe auch

[P2_SEQ_INIT](#), [P2_SEQ_READ](#), [P2_SEQ_READ24](#), [P2_SEQ_READ_PACKED](#), [P2_SEQ_WAIT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE module 4 'Moduladresse

DIM DATA_1[32] AS FLOAT 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 Messgruppen gewandelt und gespeichert hat.

Syntax

```
#INCLUDE ADwinPRO_ALL.inc
P2_SEQ_WAIT(module)
```

Parameter

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

Bemerkungen

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

Siehe auch

[P2_SEQ_INIT](#), [P2_SEQ_READ](#), [P2_SEQ_READ24](#), [P2_SEQ_READ_PACKED](#), [P2_SEQ_START](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
#DEFINE module 4                    'Moduladresse

DIM DATA_1[32] AS LONG AT DM_LOCAL
DIM DATA_2[32] AS FLOAT AT DM_LOCAL

DIM i AS LONG

INIT:
  PROCESSDELAY=100000
  P2_SE_DIFF(module,1)            'Eingänge differentiell
  REM Ablaufsteuerung auf Single Shot,
  REM Verstärkungsfaktor 1, alle Kanäle des Moduls,
  REM Standard-Einschwingzeit
  P2_SEQ_INIT(module,1,0,0FFFFFFFh,0)
  P2_SEQ_START(SHIFT_LEFT(1,module-1)) 'Messsequenz starten

EVENT:
  P2_SEQ_WAIT(module) 'Ende der Messung abwarten
  P2_SEQ_READ(module,32,DATA_1,1) 'Alle 32 Kanäle einlesen ...
  FOR i=1 TO 32
    REM Digit in Volt umrechnen und speichern
    DATA_2[i] = (DATA_1[i]-32768)*20/65536
  NEXT i
  P2_SEQ_START(SHIFT_LEFT(1,module-1)) 'Messsequenz starten
```

P2_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 8...9 stellen die Verstärkung ein, die Bits 0...4 die Nummer des Eingangs.	LONG

Bit-Nr.								
31:7	9	8	7...5	4	3	2	1	0
ohne Funktion	Verstärkung	–	Multiplexer-Eingang					
	1 = 00b	–	Eingang 1: 00000b					
	2 = 01b		Eingang 2: 00001b					
	4 = 10b		...					
	8 = 11b		Eingang 32: 11111b					

Bemerkungen

Kombinieren Sie für die gewünschte Multiplexer-Einstellung die passenden Bitkombinationen für Verstärkung und Multiplexer-Eingang.

Sie können die Bits im Parameter **pattern** im Binärformat verwenden oder sie in Hexadezimal- oder Dezimal-Format umrechnen. Beachten Sie für Hex- und Binärformat die angehängten Buchstaben **h** und **b**.

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

Siehe auch

[P2_ADC](#), [P2_START_CONV](#), [P2_WAIT_EOC](#), [P2_READ_ADC](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.inc
DIM value1 AS LONG 'Deklaration

EVENT:
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) '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

-/-

Siehe auch

[P2_ADC](#), [P2_ADC24](#), [P2_READ_ADC](#), [P2_WAIT_EOC](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value AS LONG            'Deklaration

EVENT:
  P2_START_CONV(1,1)        'Start AD-Wandlung auf Kanal 1
  P2_WAIT_EOC(1,1)        'Warten auf Wandlung-Ende
  value = P2_READ_ADC(1,1) '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_START_CONV](#), [P2_READ_ADC](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC  
DIM value AS LONG      'Deklaration  
  
EVENT:  
  P2_START_CONV(1,1)      'Start AD-Wandlung  
  P2_WAIT_EOC(1,1)      'Warten auf das Ende der Konvertierung  
  value = P2_READ_ADC(1,1) 'Wert vom ADC einlesen
```

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

Syntax

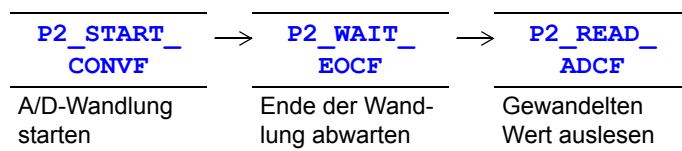
```
#INCLUDE ADwinPRO_ALL.INC
ret_val = P2_ADCF(module, input_no)
```

Parameter

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

Bemerkungen

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



Siehe auch

[P2_ADCF24](#), [P2_ADCF_MODE](#), [P2_ADCF_READ_LIMIT](#), [P2_ADCF_SET_LIMIT](#), [P2_READ_ADCF](#), [P2_START_CONVF](#), [P2_WAIT_EOCF](#)

Gültig für Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value AS LONG

EVENT:
REM 16Bit-Wert am analogen Eingg 4 messen
value = P2_ADCF(1, 4)
```

P2_ADCF

P2_ADCF24

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

Syntax

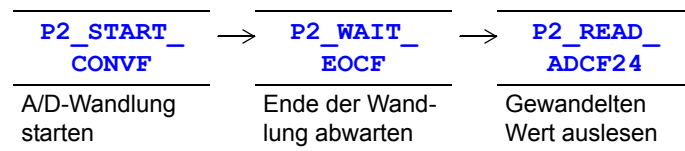
```
#INCLUDE ADwinPRO_ALL.INC
ret_val = P2_ADCF24(module, input_no)
```

Parameter

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

Bemerkungen

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



Siehe auch

P2_ADCF, P2_ADCF_MODE, P2_ADCF_READ_LIMIT, P2_ADCF_SET_LIMIT, P2_READ_ADCF24, P2_START_CONVF, P2_WAIT_EOCF

Gültig für Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value AS LONG

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

P2_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:
Bit = 0: Moduladresse ignorieren.
Bit = 1: Moduladresse ansprechen.

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

mode Arbeitsmodus des Moduls: LONG

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

Bemerkungen

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

Im Standard-Modus startet das Prozessormodul jede Wandlung für jeden Kanal einzeln, z.B. mit dem Befehl **P2_START_CONV**.

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

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

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

Im Detail geschieht Folgendes:

Die Anweisung **P2_ADCF_MODE** übergibt das eingestellte **PROCESSDELAY** des Prozesses an das Modul. Eine bestimmte Zeit später beginnt das Modul eigenständig mit der Wandlung auf allen Kanälen. Der moduleigene Timer startet die Wandlung zyklisch und – durch das übergebene **PROCESSDELAY** – synchron zum Prozesstakt; die maximale Wandlungsrate ist in der Hardware-Modulbeschreibung angegeben.


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

P2_ADCF_MODE



Standard-Modus

Timer-Modus



Timer-Modus mit Multiplex-Option

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

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

Die Messwerte können nur paarweise gelesen werden, also mit den Befehlen `P2_READ_ADCF32`, `P2_READ_ADCF4_PACKED`, `P2_READ_ADCF8_PACKED`. Der ältere der beiden Werte steht im oberen Wort, der neuere Wert im unteren Wort.

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

Das `PROCESSDELAY` des Prozesses muss geradzahlig sein, damit die Wandlung synchron getaktet wird.

Event-Modus

Im Event-Modus startet jedes Event-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.

Event-Modus mit Multiplex-Option

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_CONV`, `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 Module

AI-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC  
DIM value[4] AS LONG
```

INIT:

```
...  
P2_ADCF_MODE(1,1)      'Timer-Modus einschalten.  
                        'Letzter Befehl im Abschnitt!
```

EVENT:

```
P2_READ_ADCF4(1, value, 1) 'Werte der ADC 1-4 einlesen  
REM Werte verarbeiten
```

P2_ADCF_READ_LIMIT

P2_ADCF_READ_LIMIT liest die Flags für Grenzwertüber- und unterschreitungen auf allen F-ADC des angegebenen Moduls aus.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = P2_ADCF_READ_LIMIT(module)
```

Parameter

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

ret_val Bitmuster aus den Flags für Grenzwertüber- und unterschreitungen: LONG

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

Bemerkungen

Sie stellen die Grenzwerte mit **P2_ADCF_SET_LIMIT** ein.

Das Lesen der Flags setzt alle Flags auf Null zurück.

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

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_ADCF_MODE](#), [P2_ADCF_SET_LIMIT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM flags AS LONG

INIT:
    P2_ADCF_SET_LIMIT(1, 2, 32768,256) 'Grenzwerte Kanal 2 setzen
    flags = P2_ADCF_READ_LIMIT(1) 'Flags lesen und rücksetzen

EVENT:
    flags = P2_ADCF_READ_LIMIT(1) 'Flags lesen
    IF (flags AND 10b = 10b)
        REM Untergrenze ist unterschritten
    ...
    ENDIF
    IF (flags AND 2000h = 2000h)
        REM Obergrenze ist überschritten
    ...
    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, input_no, high, low)
```

Parameter

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

Bemerkungen

Dieser Befehl ist nur sinnvoll, wenn das Modul nicht im Standard-Arbeitsmodus arbeitet (siehe **P2_ADCF_MODE**).

Wenn ein Messwert den oberen Grenzwert überschreitet, wird für diesen Kanal ein Flag gesetzt, das mit **P2_ADCF_READ_LIMIT** gelesen und zurückgesetzt wird.

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

Grenzwertübertretungen können keine Event-Signale auslösen.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_ADCF_MODE](#), [P2_ADCF_READ_LIMIT](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM flags AS LONG

INIT:
  P2_ADCF_SET_LIMIT(1, 2, 32768,256) 'Grenzwerte Kanal 2 setzen
  flags = P2_ADCF_READ_LIMIT(1) 'Flags lesen und rücksetzen

EVENT:
  flags = P2_ADCF_READ_LIMIT(1) 'Flags lesen
  IF (flags AND 10b = 10b)
    REM Untergrenze ist unterschritten
    ...
  ENDIF
  IF (flags AND 20000h = 20000h)
    REM Obergrenze ist überschritten
    ...
  ENDIF
```

P2_ADCF_SET_LIMIT

P2_READ_ADCF

P2_READ_ADCF liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus. Das Ergebnis hat 16 Bit Auflösung.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC  
  
ret_val = P2_READ_ADCF(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Nummer des zu lesenden ADC (1...4 oder 1...8).	LONG
ret_val	Im F-ADC-Register enthaltener Messwert (0...65535).	LONG

Bemerkungen

Mit den Befehlen **P2_READ_ADCF4**, **P2_READ_ADCF8**, **P2_READ_ADCF4_PACKED**, **P2_READ_ADCF8_PACKED** können mehrere Ergebnisse sehr schnell ausgelesen werden.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_START_CONV](#), [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 Module

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

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC  
DIM value1 AS LONG  
  
EVENT:  
  REM Start AD-Wandlung; nicht erforderlich für Aln-F-8/14  
  P2_START_CONV(1,1)  
  REM Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14  
  P2_WAIT_EOCF(1,1)  
  value1 = P2_READ_ADCF(1,1) 'Wert vom ADC einlesen
```

P2_READ_ADCF24 liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus. Das Ergebnis hat 24 Bit Auflösung.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = P2_READ_ADCF24(module, adc_no)
```

Parameter

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

Bemerkungen

Mit den Befehlen **READ_ADCF4_24B**, **READ_ADCF8_24B** können mehrere Ergebnisse sehr schnell ausgelesen werden.

Siehe auch

[P2_ADCF24](#), [P2_START_CONVF](#), [P2_WAIT_EOCF](#), [P2_ADCF_MODE](#), [P2_ADCF_READ_LIMIT](#), [P2_ADCF_SET_LIMIT](#), [P2_READ_ADCF_SCONV24](#), [P2_READ_ADCF4_24B](#), [P2_READ_ADCF8_24B](#)

Gültig für Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value1 AS LONG 'Deklaration

EVENT:
  P2_START_CONVF(1,1) 'Start AD-Wandlung
  P2_WAIT_EOCF(1,1) 'Warten auf Wandlung-Ende
  value1 = P2_READ_ADCF24(1,1) '24Bit-Wert vom ADC einlesen
```

P2_READ_ADCF24

P2_READ_ADCF4

P2_READ_ADCF4 liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus.

Syntax

```
#INCLUDE ADWINPRO_ALL.INC  
P2_READ_ADCF4(module, array[], index)
```

Parameter

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_CONV](#), [P2_READ_ADCF](#), [P2_READ_ADCF8](#), [P2_READ_ADCF4_PACKED](#), [P2_READ_ADCF8_PACKED](#), [P2_READ_ADCF_SCONV](#), [P2_WAIT_EOCF](#)

Gültig für Module

AIIn-F-8/14 Rev. E, AIIn-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 AIIn-F-8/14  
    P2_START_CONV(1,0Fh)  
  
EVENT:  
    REM Warten auf Wandlung-Ende; nicht erforderlich für AIIn-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 AIIn-F-8/14  
    P2_START_CONV(1,0Fh)
```

P2_READ_ADCF4_24B liest die Wandlungsergebnisse aus den ersten 4 F-ADC des angegebenen Moduls aus. Die Ergebnisse haben eine Auflösung von 24 Bit.

Syntax

```
#INCLUDE ADWINPRO_ALL.INC

P2_READ_ADCF4_24B(module, array[], index)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Zielfeld, in dem die Messwerte (24 Bit Auflösung) gespeichert werden.	ARRAY LONG FLOAT
<code>index</code>	Index des Elements im Zielfeld, in dem der erste Messwert gespeichert wird.	LONG

Bemerkungen

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

Die Anweisung ist wesentlich schneller als das mehrfache Auslesen mit **P2_READ_ADCF24**.

Siehe auch

[P2_ADCF24](#), [P2_START_CONV](#), [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 Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC
DIM value[4] AS LONG      'Feld für Messwerte

INIT:
    P2_START_CONV(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_CONV(1,0Fh)   'Neue AD-Wandlung starten
```

P2_READ_ADCF4_24B

P2_READ_ADCF8

P2_READ_ADCF8 liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus.

Syntax

```
#INCLUDE ADWINPRO_ALL.INC  
P2_READ_ADCF8(module, array[], index)
```

Parameter

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_CONV](#), [P2_WAIT_EOCF](#), [P2_READ_ADCF4](#), [P2_READ_ADCF4_PACKED](#), [P2_READ_ADCF8_PACKED](#), [P2_READ_ADCF_SCONV](#)

Gültig für Module

AIIn-F-8/14 Rev. E, AIIn-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 AIIn-F-8/14  
    P2_START_CONV(1,0FFh)  
  
EVENT:  
    REM Warten auf Wandlung-Ende; nicht erforderlich für AIIn-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 AIIn-F-8/14  
    P2_START_CONV(1,0FFh)
```


P2_READ_ADCF8_24B liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus. Die Ergebnisse haben eine Auflösung von 24 Bit.

Syntax

```
#INCLUDE ADWINPRO_ALL.INC

P2_READ_ADCF8_24B(module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Zielfeld, in dem die Messwerte (24 Bit Auflösung) gespeichert werden.	ARRAY LONG FLOAT
index	Index des Elements im Zielfeld, in dem der erste Messwert gespeichert wird.	LONG

Bemerkungen

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

Die Anweisung ist wesentlich schneller als das mehrfache Auslesen mit **P2_READ_ADCF24**.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_START_CONVF](#), [P2_READ_ADCF8](#), [P2_READ_ADCF4_PACKED](#), [P2_READ_ADCF8_PACKED](#), [P2_READ_ADCF_SCONV24](#), [P2_WAIT_EOCF](#)

Gültig für Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC
DIM value[8] AS LONG      'Feld für Messwerte

INIT:
    P2_START_CONVF(1,0FFh)  'Start AD-Wandlung Kanäle 1...8

EVENT:
    P2_WAIT_EOCF(1,0FFh)    'Warten auf Wandlungsende
    P2_READ_ADCF8_24B(1,value,1) 'Werte der ADC 1...8 lesen
    P2_START_CONVF(1,0FFh)  'Neue AD-Wandlung starten
```

P2_READ_ADCF8_24B

P2_READ_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

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Zielfeld, in dem die Messwerte gespeichert werden.	ARRAY LONG FLOAT
<code>index</code>	Index des Elements im Zielfeld, in dem der erste Messwert gespeichert wird.	LONG

Bemerkungen

Es werden immer die Messwerte der F-ADC 1...8 des Moduls gelesen. 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
<code>array[index]</code>	F-ADC 2	F-ADC 1
<code>array[index+1]</code>	F-ADC 4	F-ADC 3
<code>array[index+2]</code>	F-ADC 6	F-ADC 5
<code>array[index+3]</code>	F-ADC 8	F-ADC 7

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_START_CONV](#), [P2_WAIT_EOCF](#), [P2_READ_ADCF4](#), [P2_READ_ADCF8](#), [P2_READ_ADCF4_PACKED](#), [P2_READ_ADCF_SCONV](#)

Gültig für Module

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

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC
DIM value[8] AS LONG      'Feld für Messwerte

INIT:
REM Start AD-Wandlung Kanäle 1...8; nicht erforderl. für Aln-F-8/14
P2_START_CONV(1, 0FFh)

EVENT:
REM Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14
P2_WAIT_EOCF(1, 0FFh)
P2_READ_ADCF8_PACKED(1, value, 1) 'Werte der ADC 1...8 lesen
REM Neue AD-Wandlung starten; nicht erforderlich für Aln-F-8/14
P2_START_CONV(1, 0FFh)
```

P2_READ_ADCF8_PACKED

P2_READ_ADCF32

P2_READ_ADCF32 liest die Wandlungsergebnisse aus 2 aufeinander folgenden F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = P2_READ_ADCF32(module, adc_no)
```

Parameter

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

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

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

ret_val Die in dem F-ADC-Registern enthaltenen Messwerte (jeweils 0...65535); je ein Messwert ist im unteren und im oberen Wort. LONG

Bemerkungen

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

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

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_READ_ADCF](#), [P2_READ_ADCF4](#), [P2_READ_ADCF8](#), [P2_READ_ADCF4_PACKED](#), [P2_READ_ADCF8_PACKED](#), [P2_READ_ADCF_SCONV](#), [P2_READ_ADCF_SCONV32](#), [P2_START_CONVF](#), [P2_WAIT_EOCF](#)

Gültig für Module

Aln-F-8/14 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 erforderl. für Aln-F-8/14
P2_START_CONVF(1,11b)
REM Warten auf Wandlung-Ende; nicht erforderlich für Aln-F-8/14
P2_WAIT_EOCF(1,3)
value1 = P2_READ_ADCF32(1,1) 'Wert von ADC1 und ADC2 einlesen
```

P2_READ_ADCF_SCONV liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = P2_READ_ADCF_SCONV(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
adc_no	Nummer des zu lesenden ADC (1...4 oder 1...8).	LONG
ret_val	Im F-ADC-Register enthaltener Messwert (0...65535).	LONG

Siehe auch

P2_ADCF, P2_ADCF24, P2_READ_ADCF, P2_READ_ADCF4, P2_READ_ADCF8, P2_READ_ADCF4_PACKED, P2_READ_ADCF8_PACKED, P2_READ_ADCF32, P2_READ_ADCF_SCONV32, P2_START_CONVF, P2_WAIT_EOCF

Gültig für Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM i AS LONG
DIM DATA_1[1000] AS LONG 'Feld für Messwerte

INIT:
  i=1
  P2_START_CONVF(1,1)      'A/D-Wandler starten

EVENT:
  P2_WAIT_EOCF(1,1)
  DATA_1[i] = P2_READ_ADCF_SCONV(1,1) 'A/D-Wandler auslesen +
                                      'starten
  INC(i)                        'Index erhöhen
  IF (i=1001) THEN END         'Nach 1000 Messwerten Prozess beenden
```

P2_READ_ADCF_SCONV

P2_READ_ADCF_SCONV24

P2_READ_ADCF_SCONV24 liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung. Der Rückgabewert hat eine Auflösung von 24 Bit.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC  
  
ret_val = P2_READ_ADCF_SCONV24(module, adc_no)
```

Parameter

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

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_READ_ADCF](#), [P2_READ_ADCF4](#), [P2_READ_ADCF8](#), [P2_READ_ADCF4_PACKED](#), [P2_READ_ADCF8_PACKED](#), [P2_READ_ADCF32](#), [P2_READ_ADCF_SCONV32](#), [P2_START_CONV](#), [P2_WAIT_EOCF](#)

Gültig für Module

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_CONV(1,1) 'A/D-Wandler starten  
  
EVENT:  
  P2_WAIT_EOCF(1,1)  
  DATA_1[i] = P2_READ_ADCF_SCONV24(1,1) 'A/D-Wandler 24 Bit  
                                     'auslesen + starten  
  INC(i) 'Index erhöhen  
  IF (i=1001) THEN END 'Nach 1000 Messwerten Prozess beenden
```

P2_READ_ADCF_SCONV32 liest die Wandlungsergebnisse aus 2 F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.

Anschließend wird sofort eine neue Konvertierung gestartet.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = P2_READ_ADCF_SCONV32(module, adc_no)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG										
adc_no	Nummer des ersten zu lesenden F-ADC (1...2 oder 1...4).	LONG										
<table><tr><td>adc_no</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>F-ADC-Nr.</td><td>1, 2</td><td>3, 4</td><td>5, 6</td><td>7, 8</td></tr></table>			adc_no	1	2	3	4	F-ADC-Nr.	1, 2	3, 4	5, 6	7, 8
adc_no	1	2	3	4								
F-ADC-Nr.	1, 2	3, 4	5, 6	7, 8								
ret_val	Der Rückgabewert (32 Bit) enthält die Messdaten von 2 aufeinanderfolgenden F-ADC (je 16 Bit: 0...65535); je ein Messwert ist im unteren und im oberen Wort.	LONG										

Siehe auch

P2_ADCF, P2_ADCF24, P2_READ_ADCF, P2_READ_ADCF4, P2_READ_ADCF8, P2_READ_ADCF4_PACKED, P2_READ_ADCF8_PACKED, P2_READ_ADCF32, P2_READ_ADCF_SCONV, P2_START_CONVF, P2_WAIT_EOCF

Gültig für Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value AS LONG          'Deklaration

INIT:
    P2_START_CONVF(1,3)     'Start AD-Wandlung

EVENT:
    P2_WAIT_EOCF(1,3)       'Warten auf das Ende der Konvertierung
    value = P2_READ_ADCF_SCONV32(1,1) 'Wert vom ADC1 und ADC2
                                     'einlesen und die Wandlung
                                     'beider ADC neu starten
```

P2_READ_ADCF_SCONV32

P2_START_CONV

P2_START_CONV startet die Wandlung auf einem oder mehreren F-ADC des angegebenen Moduls.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

P2_START_CONV(module, adc_no)
```

Parameter

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

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

Bemerkungen

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

Sie können eine Wandlung mit dem Befehl **P2_SYNC_ALL** synchron zu anderen Messungen starten, falls Sie das Modul mittels **P2_SYNC_ENABLE** zur Synchronisation frei gegeben haben.

Sie können mehrere Wandlungen ebenfalls synchron ausführen, wenn Sie die entsprechenden Module mit **P2_SYNC_MODE** zur Synchronisation frei geben.

Sobald Sie auf dem Master-Modul eine Wandlung starten, starten zeitgleich auch Wandlungen auf allen Kanälen der Slave-Module. Bei Event-gesteuerten Modulen geschieht das Gleiche, sobald am gewünschten Event-Eingang ein Signal eingeht.

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_READ_ADCF](#), [P2_READ_ADCF4](#), [P2_READ_ADCF8](#), [P2_READ_ADCF4_PACKED](#), [P2_READ_ADCF8_PACKED](#), [P2_WAIT_EOCF](#), [P2_SYNC_ALL](#)

Gültig für Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value AS LONG 'Deklaration

EVENT:
P2_START_CONV(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

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

Bemerkungen

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

Siehe auch

[P2_ADCF](#), [P2_ADCF24](#), [P2_START_CONV](#), [P2_READ_ADCF](#), [P2_READ_ADCF4](#), [P2_READ_ADCF8](#), [P2_READ_ADCF4_PACKED](#), [P2_READ_ADCF8_PACKED](#)

Gültig für Module

Aln-F-8/18 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
DIM value AS LONG 'Deklaration

EVENT:
P2_START_CONV(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

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

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

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit 1 Kanal eingerichtet wurde (siehe **P2_BURST_INIT**, Parameter `mode`, `channels`).

Die Anweisung liest die Anzahl `count` der zuletzt im Modulspeicher abgelegten Messwerte. `count` sollte etwa um den Faktor 2 kleiner sein als die bei **P2_BURST_INIT** festgelegte Anzahl der durchzuführenden Messungen (`samples`).

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

In 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_UNPACKED4](#), [P2_BURST_CREAD_UNPACKED8](#), [P2_BURST_START](#), [P2_BURST_STATUS](#)

Gültig für Module

Aln-F-8/14 Rev. E



Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
#DEFINE module 4

DIM DATA_1[1000] AS LONG
DIM pattern AS LONG

INIT:
  REM Kont. Burst-Messreihe für Kanal 1 einrichten mit 20ns
  REM Periodendauer, 2^26 Daten speichern ab Adresse 0.
  P2_BURST_INIT (module,1,0,67108864,1,010b)
  REM Burst-Messreihe starten
  pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
  P2_BURST_START(pattern)
  PROCESSDELAY=10000000

EVENT:
  REM Die letzten 1000 Messwerte des Kanals (langsam) lesen und in
  REM DATA_1 ablegen
  P2_BURST_CREAD_UNPACKED1 (module,1000,DATA_1,1,1)
```

P2_BURST_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

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

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit 2 Kanälen eingerichtet wurde (siehe **P2_BURST_INIT**, Parameter `mode`, `channels`).

Die Anweisung liest die Anzahl `count` der zuletzt im Modulspeicher abgelegten Messwerte. `count` sollte etwa um den Faktor 2 kleiner sein als die bei **P2_BURST_INIT** festgelegte Anzahl der durchzuführenden Messungen (`samples`).

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

In 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 Module

AI-F-8/14 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 einrichten mit 60ns
  REM Periodendauer, 2^26 Messwerte je Kanal ab Adresse 0.
  P2_BURST_INIT (module,3,0,67108860,3,010b)
  REM Burst-Messreihe starten
  pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
  P2_BURST_START(pattern)
  P2_SET_LED(module,1)
  PROCESSDELAY=1000000

EVENT:
  REM Die letzten 1000 Messwerte je Kanal (langsam) lesen und in
  REM den Feldern DATA_1 bis DATA_2 ablegen
  P2_BURST_CREAD_UNPACKED2(module,1000,DATA_1,DATA_2,1,1)
```

P2_BURST_CREAD_UNPACKED4

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

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

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit 4 Kanälen eingerichtet wurde (siehe **P2_BURST_INIT**, Parameter `mode`, `channels`).

Die Anweisung liest die Anzahl `count` der zuletzt im Modulspeicher abgelegten Messwerte. `count` sollte etwa um den Faktor 2 kleiner sein als die bei **P2_BURST_INIT** festgelegte Anzahl der durchzuführenden Messungen (`samples`).

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

In 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_UNPACKED3](#), [P2_BURST_CREAD_UNPACKED4](#), [P2_BURST_START](#), [P2_BURST_STATUS](#)

Gültig für Module

Aln-F-8/14 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 einrichten mit 40ns
  REM Periodendauer, 2^25 je Kanal speichern ab Adresse 0.
  P2_BURST_INIT (module,15,0,3355444,2,010b)
  REM Burst-Messreihe starten
  pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
  P2_BURST_START(pattern)
  PROCESSDELAY=50000000

EVENT:
  REM Die letzten 1000 Messwerte je Kanal (schnell) lesen und in
  REM den Feldern DATA_1 bis DATA_4 ablegen
  P2_BURST_CREAD_UNPACKED4(module,1000,DATA_1,DATA_2,DATA_3,
    DATA_4,1,3)
```

P2_BURST_CREAD_UNPACKED8

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

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

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine kontinuierliche Burst-Messreihe mit 8 Kanälen eingerichtet wurde (siehe **P2_BURST_INIT**, Parameter `mode`, `channels`).

Die Anweisung liest die Anzahl `count` der zuletzt im Modulspeicher abgelegten Messwerte. `count` sollte etwa um den Faktor 2 kleiner sein als die bei **P2_BURST_INIT** festgelegte Anzahl der durchzuführenden Messungen (`samples`).

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

In 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 Module

AIIn-F-8/14 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 einrichten mit 40ns
  REM Periodendauer, 1Mio. Messwerte je Kanal speichern ab
  REM Adresse 100.
  P2_BURST_INIT (module,255,100,1000000,2,010b)
  REM Burst-Messreihe starten
  pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
  P2_BURST_START(pattern)
  PROCESSDELAY=10000000

EVENT:
  REM Die letzten 10000 Messwerte je Kanal (langsam) lesen und in
  REM den Feldern DATA_1 bis DATA_8 ablegen
  P2_BURST_CREAD_UNPACKED8(module,1000,DATA_1,DATA_2,DATA_3,
    DATA_4,DATA_5,DATA_6,DATA_7,DATA_8,1,3)
```

P2_BURST_INIT

P2_BURST_INIT legt die Kennwerte für eine Burst-Messreihe auf dem angegebenen Modul fest.

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

Syntax

```
#INCLUDE ADwinPRO_ALL.INC
```

```
P2_BURST_INIT (module, channels, startadr, samples,  
              pulses, mode)
```

Parameter

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

channels legt Anzahl und Nummern der Messkanäle fest. LONG
Gemeinsam mit der Speichergröße ergibt sich die max. Anzahl der Messwerte pro Kanal:

chan- nels	Kanalnr.	max. Anzahl der Messwerte pro Kanal für startadr=0:
1	1	134217720 = 7FFFFFF0H
3	1...2	67108860 = 3FFFFFFCh
15	1...4	33554428 = 1FFFFFFCh
255	1...8	16777212 = 0FFFFFFCh

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

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

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

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

pulses legt die Periodendauer für eine Messreihe fest als Anzahl der Zeittakte; nur gültig in Verbindung mit zeitgesteuerter Taktrate (siehe **mode**): LONG
Periodendauer = **pulses** * 20ns.
Der Wertebereich ist 1...65535; bei 8 Kanälen (**mode** = 255 oder 127) beginnt der Wertebereich mit 2.
Eine Periodendauer ist die Zeit vom Beginn einer Messung bis zum Beginn der nächsten Messung.

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

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

Bemerkungen

Sie können mit **P2_READ_ADCF** den aktuellen Messwert auch eines solchen Kanals einlesen, der nicht abgespeichert wird, z.B. zum Prüfen einer Trigger-Bedingung.

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

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

Bei einer einfachen Burst-Messreihe erfasst das Modul eine feste Anzahl von Messungen; Einstellungen siehe **P2_BURST_INIT**. Sobald alle Messwerte gewandelt sind, endet die Burst-Messreihe. Die Messdaten sollen mit **P2_BURST_READ_UNPACKED...** gelesen werden.

Bei einer kontinuierlichen Burst-Messreihe wandelt das Modul dauernd Messwerte mit der festgelegten Periodendauer. Sie brechen die Messreihe mit **P2_BURST_STOP** ab und lesen mit **P2_BURST_CREAD_UNPACKED...** die Messwerte aus. Das Modul speichert die Messwerte im reservierten Speicher (siehe **P2_BURST_INIT**) in einem Ringspeicher, d.h. die jüngsten Messwerte überschreiben die jeweils ältesten Daten.

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

Bei extern gesteuerter Taktrate löst jedes Event-Signal eine Burst-Messung aus; beachten Sie die Einstellung mit **P2_EVENT_CONFIG**. Die maximale Taktrate beträgt 50MHz.

Siehe auch

P2_BURST_CREAD_UNPACKED1, P2_BURST_CREAD_UNPACKED2, P2_BURST_CREAD_UNPACKED4, P2_BURST_CREAD_UNPACKED8, 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 Module

AIIn-F-8/14 Rev. E

Betriebsart

Taktrate

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
#DEFINE module 4

DIM DATA_1[1000] AS LONG
DIM pattern AS LONG

INIT:
    REM Kont. Burst-Messreihe für Kanal 1 einrichten mit 20ns
    REM Periodendauer, 2^26 Daten speichern ab Adresse 0.
    P2_BURST_INIT (module,1,0,67108864,1,010b)
    REM Burst-Messreihe starten
    pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
    P2_BURST_START(pattern)
    PROCESSDELAY=10000000

EVENT:
    REM Die letzten 1000 Messwerte des Kanals (langsam) lesen und in
    REM DATA_1 ablegen
    P2_BURST_CREAD_UNPACKED1 (module,1000,DATA_1,1,1)
```

P2_BURST_READ_INDEX gibt die Adresse im Modulspeicher zurück, an der zuletzt Messwerte abgelegt wurden.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

ret_val = P2_BURST_READ_INDEX(module)
```

Parameter

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

Bemerkungen

P2_BURST_READ_INDEX ist ein elementarer Befehl, der in Verbindung mit **P2_BURST_READ** spezielle Lösungen ermöglicht, dafür aber auch besondere Sorgfalt und Kenntnisse bei der Programmierung voraussetzt. Die einfachere Alternative sind die Befehle **P2_BURST_READ_UNPACKED...** oder **P2_BURST_CREAD_UNPACKED...**.

Aus der zurückgegebenen Adresse **ret_val** kann die Anzahl **n** der gespeicherten Messdaten berechnet werden. Startadresse und Kanalzahl werden mit **P2_BURST_INIT** festgelegt:

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

Der Modulspeicher wird immer in 4er-Schritten (4 mal 32 Bit) adressiert. Nach den Befehlen **P2_BURST_INIT** und **P2_BURST_RESET** steht der Adresszeiger auf der letztmöglichen Adresse des reservierten Modulspeichers, also auf **startadr + samples * (Kanalzahl) - 4**.

Von welchen Kanälen die zuletzt im Speicher abgelegten Messwerte stammen, hängt vom Messmodus ab. Näheres über die Zuordnung siehe **P2_BURST_READ**.

Siehe auch

[P2_BURST_INIT](#), [P2_BURST_READ](#)

[P2_BURST_READ_UNPACKED1](#), [P2_BURST_READ_UNPACKED2](#),
[P2_BURST_READ_UNPACKED4](#), [P2_BURST_READ_UNPACKED8](#),
[P2_BURST_RESET](#), [P2_BURST_START](#), [P2_BURST_STATUS](#), [P2_READ_ADC](#)

Gültig für Module

Aln-F-8/14 Rev. E

P2_BURST_READ_INDEX

Beispiel

```
#INCLUDE ADWINPRO_ALL.INC

#DEFINE module 4
#DEFINE samples 500000
#DEFINE channels 4
#DEFINE frq_Hz 5000
#DEFINE mem_idx PAR_1
#DEFINE count PAR_2
#DEFINE overflow PAR_3

DIM DATA_1[samples], DATA_2[samples] AS LONG
DIM DATA_3[samples], DATA_4[samples] AS LONG
DIM i, prev_mem_idx, start_idx AS LONG

LOWINIT:
  FOR i = 1 TO samples
    DATA_1[i] = 0 : DATA_2[i] = 0 : DATA_3[i] = 0 : DATA_4[i] = 0
  NEXT i

INIT:
  PROCESSDELAY = 300000000 / frq_Hz
  P2_SET_LED(module, 1)      'LED einschalten
  REM Kont. Burst-Messreihe, Kanäle 1...4, 100ns Periodendauer
  P2_BURST_INIT(module, 15, 0, samples, 5, 2)
  P2_BURST_START(SHIFT_LEFT(1, module - 1))
  start_idx = 1
  prev_mem_idx = 0
  overflow = 0

EVENT:
  REM Aktuelle Speicheradresse
  mem_idx = P2_BURST_READ_INDEX(module)
  REM Anzahl neuer Messwerte/Kanal seit dem letzten Zyklus
  count = (mem_idx - prev_mem_idx) * 2 / channels

  IF (count > 0) THEN
    REM Messwerte aus dem F8/14 Modul auslesen
    P2_BURST_READ_UNPACKED4(module, count, prev_mem_idx,
      DATA_1, DATA_2, DATA_3, DATA_4, start_idx, 0)
    REM Start-Index für den nächsten Zyklus
    start_idx = start_idx + count
    REM Index im F8/14 Modul merken
    prev_mem_idx = mem_idx
  ENDIF

  IF (count < 0) THEN
    REM Anzahl der Messwerte bis zum Data Ende
    count = samples - prev_mem_idx * 2 / channels
    REM Messwerte aus dem F8/14 Modul auslesen
    P2_BURST_READ_UNPACKED4(module, count, prev_mem_idx,
      DATA_1, DATA_2, DATA_3, DATA_4, start_idx, 0)
    REM Start-Index im Data für den nächsten Zyklus
    start_idx = 1
    REM Index im F8/14 Modul für den nächsten Zyklus
    prev_mem_idx = 0
    INC(overflow)           'Überlaufzähler erhöhen
  ENDIF

FINISH:
  P2_SET_LED(module, 0)    'LED ausschalten
```

P2_BURST_READ 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

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

Bemerkungen

P2_BURST_READ ist ein elementarer Befehl, der in Verbindung mit **P2_BURST_READ_INDEX** spezielle Lösungen ermöglicht, dafür aber auch besondere Sorgfalt und Kenntnisse bei der Programmierung voraussetzt. Die einfachere Alternative sind die Befehle **P2_BURST_READ_UNPACKED...** oder **P2_BURST_CREAD_UNPACKED...**.

P2_BURST_READ kopiert die 32 Bit-Werte aus dem Speicher, ohne sie zu verändern; ein 32 Bit-Wert enthält 2 Messwerte zu 16 Bit. Welchen Kanälen die Messwerte zugeordnet sind, hängt von der Kanalanzahl ab (siehe **P2_BURST_INIT**, Parameter `channels`). Die folgende Übersicht zeigt, wie die 16 Bit-Messwerte M den Kanalnummern K zugeordnet sind:

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

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

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
...
4 Kanäle		

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

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

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

Siehe auch

[P2_BURST_INIT](#), [P2_BURST_READ_INDEX](#)
[P2_BURST_READ_UNPACKED1](#), [P2_BURST_READ_UNPACKED2](#),
[P2_BURST_READ_UNPACKED4](#), [P2_BURST_READ_UNPACKED8](#),
[P2_BURST_START](#), [P2_BURST_STATUS](#), [P2_BURST_STOP](#), [P2_READ_ADC](#)

Gültig für Module

Aln-F-8/14 Rev. E

Beispiel

siehe [P2_BURST_READ_INDEX](#)

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,  
    array1[], array_idx, flowrate)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>count</code>	Anzahl der zu übertragenden Messwerte. Die Anzahl muss durch 8 teilbar sein.	LONG
<code>startadr</code>	Startadresse (0...268435452 = $2^{28} - 4$) im Modulspeicher: Adresse, ab der die Messwerte gelesen werden. Die Adresse muss durch 4 teilbar sein.	LONG
<code>array1[]</code>	Ziel-Feld, in das die Messwerte übertragen werden. Der Datentyp Float und FIFO-Felder sind nicht erlaubt.	ARRAY LONG FLOAT
<code>array_idx</code>	Ziel-Startindex: Feldelement, ab dem die Messwerte abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine Burst-Messreihe mit einem einzigen Kanal eingerichtet wurde (siehe **P2_BURST_INIT**, Parameter `channels`).

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

In 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 Module

AIIn-F-8/14 Rev. E

P2_BURST_READ_UNPACKED1



Beispiel

Siehe auch Beispiele für [Kontinuierliche Messwertwandlung](#) auf [Seite 342](#).

```
#INCLUDE ADwinPRO_ALL.INC
#DEFINE module 1

DIM DATA_1[1000] AS LONG
DIM state AS LONG
DIM rest AS LONG
DIM pattern AS LONG

INIT:
    REM Einfache Burst-Messreihe für Kanal 1 einrichten mit 20ns
    REM Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
    P2_BURST_INIT (module,1,0,1000,1,0)
    REM Burst-Messreihe starten
    pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
    P2_BURST_START(pattern)
    PROCESSDELAY=10000000
    state=0                                'Status: Burst-Messreihe läuft

EVENT:
    REM Anzahl der restlichen Messwerte holen
    rest = P2_BURST_STATUS(module)
    REM Alle Messwerte liegen vor: Status ändern
    IF (rest=0) THEN state=1
    IF (state=1) THEN
        REM Alle Messwerte liegen vor: 1000 Messwerte (schnell)
        REM abholen und in DATA_1 ablegen
        P2_BURST_READ_UNPACKED1(module,1000,0,DATA_1,1,3)
        REM Nächste Burst-Messreihe starten
        state=0
        P2_BURST_RESET(pattern)
        P2_BURST_START(pattern)
    ENDIF
```

P2_BURST_READ_UNPACKED2 kopiert die Messwerte von 2 Kanälen aus dem Speicher des angegebenen Moduls in 2 Felder.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC
```

```
P2_BURST_READ_UNPACKED2(module, count, startadr,  
    array1[], array2[], array_idx, flowrate)
```

Parameter

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

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine Burst-Messreihe mit 2 Kanälen eingerichtet wurde (siehe **P2_BURST_INIT**, Parameter `channels`).

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

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

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

Siehe auch

[P2_BURST_INIT](#), [P2_BURST_READ_UNPACKED1](#), [P2_BURST_READ_UNPACKED4](#), [P2_BURST_READ_UNPACKED8](#), [P2_BURST_RESET](#), [P2_BURST_START](#), [P2_BURST_STATUS](#)

Gültig für Module

AI-F-8/14 Rev. E

P2_BURST_READ_UNPACKED2



Beispiel

Siehe auch Beispiele für [Kontinuierliche Messwertwandlung](#) auf Seite 342.

```
#INCLUDE ADwinPRO_ALL.INC
#DEFINE module 1

DIM DATA_1[1000], DATA_2[1000] AS LONG
DIM state AS LONG
DIM rest AS LONG
DIM pattern AS LONG

INIT:
    REM Einfache Burst-Messreihe für Kanäle 1+2 einrichten mit 40ns
    REM Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
    P2_BURST_INIT (module,3,0,1000,2,0)
    REM Burst-Messreihe starten
    pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
    P2_BURST_START(pattern)
    PROCESSDELAY=10000000
    state=0

EVENT:
    REM Anzahl der restlichen Messwerte holen
    rest = P2_BURST_STATUS(module)
    REM Alle Messwerte liegen vor: Status ändern
    IF (rest = 0) THEN state=1
    IF (state = 1) THEN
        REM Alle Messwerte liegen vor: Von jedem Kanal 1000 Messwerte
        REM (schnell) abholen und in DATA_1 ablegen
        P2_BURST_READ_UNPACKED2(module,1000,0,DATA_1,DATA_2,1,3)
        REM Nächste Burst-Messreihe starten
        state=0
        P2_BURST_RESET(pattern)
        P2_BURST_START(pattern)
    ENDIF
```

P2_BURST_READ_UNPACKED4 kopiert die Messwerte von 4 Kanälen aus dem Speicher des angegebenen Moduls in 4 Felder.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC
```

```
P2_BURST_READ_UNPACKED4(module, count, startadr,  
    array1[], array2[], array3[], array4[],  
    array_idx, flowrate)
```

Parameter

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

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine Burst-Messreihe mit 4 Kanälen eingerichtet wurde (siehe **P2_BURST_INIT**, Parameter `channels`).

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

In 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 Module

AI-F-8/14 Rev. E

P2_BURST_READ_UNPACKED4



Beispiel

Siehe auch Beispiele für [Kontinuierliche Messwertwandlung](#) auf [Seite 342](#).

```
#INCLUDE ADwinPRO_ALL.INC
#DEFINE module 4

DIM DATA_1[1000], DATA_2[1000] AS LONG
DIM DATA_3[1000], DATA_4[1000] AS LONG
DIM state AS LONG
DIM rest AS LONG
DIM pattern AS LONG

INIT:
  REM Einfache Burst-Messreihe für Kanäle 1...4 einrichten mit 40ns
  REM Periodendauer, 3000 Messwerte je Kanal ab Adr. 0 speichern.
  P2_BURST_INIT (module,15,0,3000,2,0)
  REM Burst-Messreihe starten
  pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
  P2_BURST_START(pattern)
  PROCESSDELAY=10000000
  state=0

EVENT:
  REM Anzahl der restlichen Messwerte holen
  rest=P2_BURST_STATUS(module)
  REM Alle Messwerte liegen vor: Status ändern
  IF (rest=0) THEN state=1
  IF (state=1) THEN
    REM Alle Messwerte liegen vor: Von jedem Kanal 3000 Messwerte
    REM (schnell) abholen und in DATA_1 bis DATA_4 ablegen
    P2_BURST_READ_UNPACKED4(module,1000,0,DATA_1,DATA_2,DATA_3,
      DATA_4,1,3)
    REM Nächste Burst-Messreihe starten
    state=0
    P2_BURST_RESET(pattern)
    P2_BURST_START(pattern)
  ENDIF
```

P2_BURST_READ_UNPACKED8 kopiert die Messwerte von 8 Kanälen aus dem Speicher des angegebenen Moduls in 8 Felder.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC
```

```
P2_BURST_READ_UNPACKED8(module, count, startadr,  
    array1[], array2[], array3[], array4[], array5[],  
    array6[], array7[], array8[], array_idx,  
    flowrate)
```

Parameter

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

Bemerkungen

Die Anweisung soll verwendet werden, wenn eine Burst-Messreihe mit 8 Kanälen eingerichtet wurde (siehe **P2_BURST_INIT**, Parameter `channels`).

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

In 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 Module

AI-F-8/14 Rev. E

P2_BURST_READ_UNPACKED8



Beispiel

Siehe auch Beispiele für [Kontinuierliche Messwertwandlung](#) auf [Seite 342](#).

```
#INCLUDE ADwinPRO_ALL.INC
#DEFINE module 4

DIM DATA_1[1000] AS LONG
DIM DATA_2[1000] AS LONG
DIM DATA_3[1000] AS LONG
DIM DATA_4[1000] AS LONG
DIM DATA_5[1000] AS LONG
DIM DATA_6[1000] AS LONG
DIM DATA_7[1000] AS LONG
DIM DATA_8[1000] AS LONG
DIM state AS LONG
DIM rest AS LONG
DIM pattern AS LONG

INIT:
    REM Einfache Burst-Messreihe für Kanal 1 einrichten mit 20ns
    REM Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
    P2_BURST_INIT (module,255,0,1000,2,0)
    REM Burst-Messreihe starten
    pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
    P2_BURST_START(pattern)
    PROCESSDELAY=10000000
    state=0

EVENT:
    REM Anzahl der restlichen Messwerte holen
    rest=P2_BURST_STATUS(module)
    REM Alle Messwerte liegen vor: Status ändern
    IF (rest=0) THEN state=1
    IF (state=1) THEN
        REM Alle Messwerte liegen vor: 1000 Messwerte (schnell)
        REM abholen und in DATA_1 ablegen
        P2_BURST_READ_UNPACKED8(module,1000,0,DATA_1,DATA_2,DATA_3,
            DATA_4,DATA_5,DATA_6,DATA_7,DATA_8,1,3)
        REM Nächste Burst-Messreihe starten
        state=0
        P2_BURST_RESET(pattern)
        P2_BURST_START(pattern)
    ENDIF
```


P2_BURST_RESET setzt den Datenzeiger der Burst-Messreihe auf allen angegebenen Modulen zurück.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC
```

```
P2_BURST_RESET(module_pattern)
```

Parameter

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

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

Bemerkungen

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

Der Datenzeiger gibt an, an welcher Adresse im Modulspeicher die letzten Messwerte abgelegt wurden. Durch das Rücksetzen werden die nächsten Messwerte ab der mit **P2_BURST_INIT** eingestellten Startadresse gespeichert. Der Datenzeiger kann mit **P2_BURST_READ_INDEX** gelesen werden.

Siehe auch

[P2_BURST_INIT](#), [P2_BURST_READ_INDEX](#), [P2_BURST_CREAD_UNPACKED1](#), [P2_BURST_CREAD_UNPACKED2](#), [P2_BURST_CREAD_UNPACKED4](#), [P2_BURST_CREAD_UNPACKED8](#), [P2_BURST_STATUS](#), [P2_BURST_STOP](#)

Gültig für Module

AIIn-F-8/14 Rev. E

P2_BURST_RESET



Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
#DEFINE module 1

DIM DATA_1[1000] AS LONG
DIM state AS LONG
DIM rest AS LONG
DIM pattern AS LONG

INIT:
    REM Einfache Burst-Messreihe für Kanal 1 einrichten mit 20ns
    REM Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
    P2_BURST_INIT (module,1,0,1000,1,0)
    REM Burst-Messreihe starten
    pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
    P2_BURST_START(pattern)
    PROCESSDELAY=10000000
    state=0 'Status: Burst-Messreihe läuft

EVENT:
    REM Anzahl der restlichen Messwerte holen
    rest = P2_BURST_STATUS(module)
    REM Alle Messwerte liegen vor: Status ändern
    IF (rest=0) THEN state=1
    IF (state=1) THEN
        REM Alle Messwerte liegen vor: 1000 Messwerte (schnell)
        REM abholen und in DATA_1 ablegen
        P2_BURST_READ_UNPACKED1(module,1000,0,DATA_1,1,3)
        REM Nächste Burst-Messreihe starten
        state=0
        P2_BURST_RESET(pattern)
        P2_BURST_START(pattern)
    ENDIF
```

P2_BURST_START startet eine Burst-Messreihe auf allen angegebenen Modulen gleichzeitig.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

P2_BURST_START(module_pattern)
```

Parameter

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

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

Bemerkungen

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



Siehe auch

P2_BURST_INIT, P2_BURST_READ_INDEX, P2_BURST_CREAD_UNPACKED1, P2_BURST_CREAD_UNPACKED2, P2_BURST_CREAD_UNPACKED4, P2_BURST_CREAD_UNPACKED8, P2_BURST_RESET, P2_BURST_STATUS, P2_BURST_STOP

Gültig für Module

Aln-F-8/14 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
#DEFINE module 4

DIM DATA_1[1000] AS LONG
DIM pattern AS LONG

INIT:
    REM Kont. Burst-Messreihe für Kanal 1 einrichten mit 20ns
    REM Periodendauer, 2^26 Daten speichern ab Adresse 0.
    P2_BURST_INIT (module,1,0,67108864,1,010b)
    REM Burst-Messreihe starten
    pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
    P2_BURST_START(pattern)
    PROCESSDELAY=10000000

EVENT:
    REM Die letzten 1000 Messwerte des Kanals (langsam) lesen und in
    REM DATA_1 ablegen
    P2_BURST_CREAD_UNPACKED1 (module,1000,DATA_1,1,1)
```

P2_BURST_STATUS

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

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>ret_val</code>	Anzahl der noch auszuführenden Messungen.	LONG

Bemerkungen

Die Anweisung soll nur bei einer einfachen Burst-Messreihe (siehe **P2_BURST_INIT**) verwendet werden.

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

Siehe auch

[P2_BURST_INIT](#), [P2_BURST_READ_INDEX](#), [P2_BURST_READ_UNPACKED1](#), [P2_BURST_READ_UNPACKED2](#), [P2_BURST_READ_UNPACKED4](#), [P2_BURST_READ_UNPACKED8](#), [P2_BURST_RESET](#), [P2_BURST_START](#), [P2_READ_ADC](#), [P2_BURST_STOP](#)

Gültig für Module

Aln-F-8/14 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC
#define module 1

DIM DATA_1[1000] AS LONG
DIM state AS LONG
DIM rest AS LONG
DIM pattern AS LONG

INIT:
  REM Einfache Burst-Messreihe für Kanal 1 einrichten mit 20ns
  REM Periodendauer, 1000 Messwerte ab Adresse 0 speichern.
  P2_BURST_INIT (module,1,0,1000,1,0)
  REM Burst-Messreihe starten
  pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
  P2_BURST_START(pattern)
  PROCESSDELAY=10000000
  REM Status: Burst-Messreihe läuft
  state=0

EVENT:
  REM Anzahl der restlichen Messwerte holen
  rest=P2_BURST_STATUS(module)
  REM Alle Messwerte liegen vor: Status ändern
  IF (rest=0) THEN state=1
  IF (state=1) THEN
    REM Alle Messwerte liegen vor: 1000 Messwerte (schnell)
    REM abholen und in DATA_1 ablegen
    P2_BURST_READ_UNPACKED1(module,1000,0,DATA_1,1,3)
    REM Nächste Burst-Messreihe starten
    state=0
    P2_BURST_RESET(pattern)
    P2_BURST_START(pattern)
  ENDIF
```

P2_BURST_STOP

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

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

Bemerkungen

Ein interner Datenzeiger gibt an, an welcher Adresse im Modulspeicher die letzten Messwerte abgelegt wurden. Der Datenzeiger kann mit **P2_BURST_READ_INDEX** gelesen werden.

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

Mit **P2_BURST_RESET** wird der Datenzeiger auf die mit **P2_BURST_INIT** eingestellte Startadresse zurückgesetzt. Bisher gespeicherte Messwerte werden dadurch überschrieben.

Siehe auch

[P2_BURST_INIT](#), [P2_BURST_CREAD_UNPACKED1](#), [P2_BURST_READ_UNPACKED1](#), [P2_BURST_READ](#), [P2_BURST_STATUS](#)

Gültig für Module

Aln-F-8/14 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 20ns Periodendauer, 2^26 Daten speichern
  REM ab Adresse 0.
  P2_BURST_INIT (module,1,0,67108864,1,1,0)
  REM Burst-Messreihe starten
  pattern = SHIFT_LEFT(1,module-1) 'nur ein Modul ansprechen
  P2_BURST_START(pattern)
  PROCESSDELAY=10000000

EVENT:
  REM Die letzten 1000 Messwerte des Kanals (langsam) lesen und in
  REM DATA_1 ablegen
  P2_BURST_CREAD_UNPACKED1(module,1000,DATA_1,1,1)
  REM Burst-Messreihe unterbrechen, wenn Grenzwert überschritten
  FOR i = 1 TO 1000
    IF (DATA_1[i]>5)
      P2_BURST_STOP(pattern)
    ENDIF
  NEXT
```

P2_SET_AVERAGE_FILTER

P2_SET_AVERAGE_FILTER legt fest, ob und aus wievielen Messwerten das angegebene Modul einen gleitenden Mittelwert berechnet.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC  
  
P2_SET_AVERAGE_FILTER(module, mode)
```

Parameter

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

Bemerkungen

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

Der gleitende Mittelwert wird immer aus den zuletzt gewandelten Messwerten berechnet.

Siehe auch

[P2_BURST_INIT](#), [P2_BURST_READ_UNPACKED1](#), [P2_BURST_CREAD_UNPACKED1](#), [P2_READ_ADC](#)

Gültig für Module

Aln-F-8/14 Rev. E

Beispiel

```
#INCLUDE ADwinPRO_ALL.INC  
  
INIT:  
  REM Mittelwert aus den 2 jeweils zuletzt gewandelten Messwerten  
  REM bilden  
  P2_SET_AVERAGE_FILTER(1,1)
```


4.3 Pro II: Ausgangsmodule

Die Include-Datei `ADwinPRO_ALL.INC` beinhaltet alle Funktionen und Prozeduren, die zum Ansprechen der *ADwin-Pro II*-Ausgangsmodule benötigt werden. Fügen Sie deshalb die folgende *ADbasic*-Anweisung in Ihr Programm ein:

```
#INCLUDE ADwinPRO_ALL.INC
```

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

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



P2_DAC

P2_DAC gibt auf einem Kanal des angegebenen Moduls eine (analoge) Spannung aus, die dem angegebenen Digitalwert entspricht.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

P2_DAC(module, dac_no, value)
```

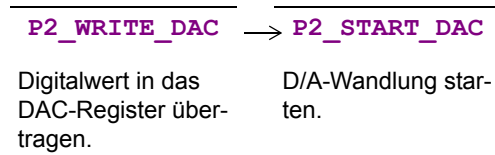
Parameter

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

Bemerkungen

Wir empfehlen, die Befehle **P2_DAC4** oder **P2_DAC8** zu verwenden, weil sie in der gleichen Zeit wie **P2_DAC** eine größere Anzahl von Werten ausgeben können.

Der Befehl **P2_DAC** besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Siehe auch

[P2_DAC4](#), [P2_DAC8_PACKED](#), [P2_START_DAC](#), [P2_WRITE_DAC](#), [P2_WRITE_DAC4](#), [P2_WRITE_DAC4_PACKED](#), [P2_WRITE_DAC8](#), [P2_WRITE_DAC8_PACKED](#), [P2_WRITE_DAC32](#)

Gültig für Module

AOut-4/16 Rev. E, AOut-8/16 Rev. E

Beispiel

```
REM Digitaler P-Regler
#include ADwinPRO_ALL.INC
DIM sp, dev AS LONG
DIM g, actuate AS LONG

EVENT:
    sp = PAR_1                'Sollwert
    g = PAR_2                 'Verstärkung
    dev = sp - P2_ADC(1,1)    'Regelabweichung berechnen
    actuate = dev * g         'Stellgröße berechnen
    P2_DAC(1,1,actuate)      'Ausgabe der Stellgröße
```

P2_DAC4 gibt 4 Digitalwerte aus einem Feld auf die DAC 1...4 des angegebenen Moduls als (analoge) Spannung aus.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

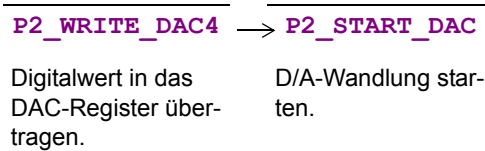
P2_DAC4(module,array[],index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535).	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Bemerkungen

Der Befehl **P2_DAC4** besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Siehe auch

P2_DAC, P2_DAC8_PACKED, P2_START_DAC, P2_WRITE_DAC, P2_WRITE_DAC4, P2_WRITE_DAC4_PACKED, P2_WRITE_DAC8, P2_WRITE_DAC8_PACKED, P2_WRITE_DAC32

Gültig für Module

AOut-4/16 Rev. E, AOut-8/16 Rev. E

Beispiel

```
REM Digitaler P-Regler für 4Kanäle
#include ADwinPRO_ALL.INC
DIM sp, dev AS LONG
DIM g, actuate AS LONG
DIM array[4] AS LONG

EVENT:
  sp = PAR_1          'Sollwert
  g = PAR_2           'Verstärkung
  P2_READ_ADCF4(1,array,1) '4 Eingangswerte lesen
  FOR i = 1 TO 4
    dev = sp - array[i] 'Regelabweichung berechnen
    array[i] = dev * g   'Stellgröße berechnen
  NEXT i
  P2_DAC4(2,array,1)    '4 Stellgrößen ausgeben
```

P2_DAC4_PACKED

P2_DAC4_PACKED gibt 4 gepackte Digitalwerte aus einem Feld auf die DAC 1...4 des angegebenen Moduls als (analoge) Spannung aus.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC
```

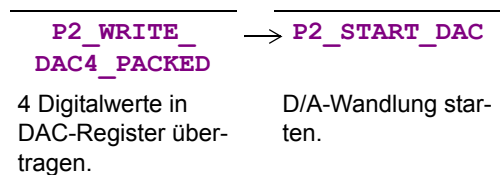
```
P2_DAC4_PACKED(module,array[],index)
```

Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Feld mit den auszugebenden Werten (0...65535) in gepackter Form: Je 2 Werte zu 16 Bit in einem 32 Bit-Wert.	ARRAY LONG FLOAT
<code>index</code>	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	<code>array[n+1]</code>		<code>array[n]</code>	
Bit-Nr.	31:16	15:00	31:16	15:00
Digitalwert für	DAC4	DAC3	DAC2	DAC1

Siehe auch

[P2_DAC](#), [P2_DAC8_PACKED](#), [P2_START_DAC](#), [P2_WRITE_DAC](#), [P2_WRITE_DAC4](#), [P2_WRITE_DAC4_PACKED](#), [P2_WRITE_DAC8](#), [P2_WRITE_DAC8_PACKED](#), [P2_WRITE_DAC32](#)

Gültig für Module

AOut-4/16 Rev. E, AOut-8/16 Rev. E

Beispiel

```
REM Digitaler P-Regler für 4Kanäle
#INCLUDE ADwinPRO_ALL.INC
DIM sp, dev1, dev2 AS LONG
DIM g AS LONG
DIM array[2] AS LONG

EVENT:
  sp = PAR_1           'Sollwert
  g = PAR_2           'Verstärkung
  P2_READ_ADCF4_PACKED(1,array,1) '4 Eingangswerte lesen
  FOR i = 1 TO 2
    REM Regelabweichungen berechnen
    dev1 = sp - (array[i] AND FFh)
    dev2 = sp - (SHIFT_RIGHT(array[i],16) AND FFh)
    REM Stellgrößen berechnen und speichern
    array[i] = SHIFT_LEFT(dev2*g, 16) + dev1*g
  NEXT i
  P2_DAC4_PACKED(2,array,1) '4 Stellgrößen ausgeben
```

P2_DAC8

P2_DAC8 gibt 8 Digitalwerte aus einem Feld auf die DAC 1...8 des angegebenen Moduls als (analoge) Spannung aus.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

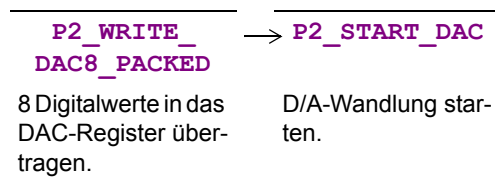
P2_DAC8(module,array[],index)
```

Parameter

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 Module

AOut-8/16 Rev. E

Beispiel

```
REM Digitaler P-Regler für 4Kanäle
#include ADwinPRO_ALL.INC
DIM sp, dev AS LONG
DIM g, actuate AS LONG
DIM array[8] AS LONG

EVENT:
  sp = PAR_1           'Sollwert
  g = PAR_2           'Verstärkung
  P2_READ_ADCF8(1,array,1) '8 Eingangswerte lesen
  FOR i = 1 TO 8
    dev = sp - array[i] 'Regelabweichung berechnen
    array[i] = dev * g 'Stellgröße berechnen
  NEXT i
  P2_DAC8(2,array,1) '8 Stellgrößen ausgeben
```

P2_DAC8_PACKED gibt die Digitalwerte aus einem Feld auf den DAC 1...8 des angegebenen Moduls als (analoge) Spannung aus.

Syntax

```
#INCLUDE ADwinPRO_ALL.INC

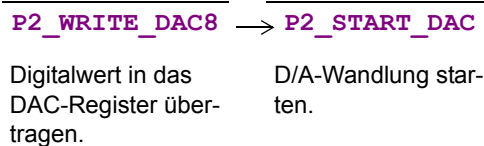
P2_DAC8_PACKED (module, array[], index)
```

Parameter

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535) in gepackter Form: Je 2 Werte zu 16 Bit in einem 32 Bit-Wert.	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Bemerkungen

Der Befehl **P2_DAC8_PACKED** besteht aus einer Sequenz von zwei Befehlen, die im folgenden Ablaufplan schematisch dargestellt ist.



Jeweils 4 Werte zu 32 Bit im Feld enthalten 8 Digitalwerte zu 16 Bit in folgender Form:

Feldelement	array[n+3]	array[n+2]	array[n+1]	array[n]
Bit-Nr.	31:16	15:00	31:16	15:00
Digitalwert für	DAC8	DAC7	DAC6	DAC5
			DAC4	DAC3
				DAC2
				DAC1

Siehe auch

P2_DAC, P2_DAC4, P2_START_DAC, P2_WRITE_DAC, P2_WRITE_DAC4, P2_WRITE_DAC4_PACKED, P2_WRITE_DAC8, P2_WRITE_DAC8_PACKED, P2_WRITE_DAC32

Gültig für Module

AOut-8/16 Rev. E

Beispiel

```
REM Digitaler P-Regler für 8 Kanäle
#include ADwinPRO_ALL.INC
DIM sp, dev1, dev2 AS LONG
DIM g AS LONG
DIM array[4] AS LONG

EVENT:
  sp = PAR_1          'Sollwert
  g = PAR_2           'Verstärkung
  P2_READ_ADCF8_PACKED (1, array, 1) '8 Eingangswerte lesen
  FOR i = 1 TO 4
    REM Regelabweichungen berechnen
    dev1 = sp - (array[i] AND FFh)
    dev2 = sp - (SHIFT_RIGHT(array[i], 16) AND FFh)
    REM Stellgrößen berechnen und speichern
    array[i] = SHIFT_LEFT(dev2*g, 16) + dev1*g
  NEXT i
  P2_DAC8_PACKED (2, array, 1) '8 Stellgrößen ausgeben
```

P2_DAC8_PACKED

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

Die Wandlung kann auch synchron mit anderen Modulen gestartet werden. Verwenden Sie hierzu den Befehl **P2_SYNC_ALL**.

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_PACKED](#), [P2_SYNC_ALL](#), [P2_WRITE_DAC](#), [P2_WRITE_DAC4](#), [P2_WRITE_DAC8_PACKED](#), [P2_WRITE_DAC32](#)

Gültig für Module

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

Bemerkungen

Wir empfehlen, die Befehle **P2_WRITE_DAC4**, **P2_WRITE_DAC8** oder **P2_WRITE_DAC32** zu verwenden, weil sie in der gleichen Zeit wie **P2_WRITE_DAC** eine größere Anzahl von Werten ausgeben können.

Siehe auch

P2_DAC, **P2_DAC4**, **P2_DAC8_PACKED**, **P2_START_DAC**, **P2_WRITE_DAC4**, **P2_WRITE_DAC4_PACKED**, **P2_WRITE_DAC8**, **P2_WRITE_DAC8_PACKED**, **P2_WRITE_DAC32**

Gültig für Module

AOut-4/16 Rev. E, AOut-8/16 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

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535).	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_PACKED](#), [P2_START_DAC](#), [P2_WRITE_DAC](#), [P2_WRITE_DAC4_PACKED](#), [P2_WRITE_DAC8](#), [P2_WRITE_DAC8_PACKED](#), [P2_WRITE_DAC32](#)

Gültig für Module

AOut-4/16 Rev. E, AOut-8/16 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

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Feld mit den auszugebenden Werten (0...65535) in gepackter Form: Je 2 Werte zu 16 Bit in einem 32 Bit-Wert.	ARRAY LONG FLOAT
<code>index</code>	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	<code>array[n+1]</code>		<code>array[n]</code>	
Bit-Nr.	31:16	15:00	31:16	15:00
Digitalwert für	DAC4	DAC3	DAC2	DAC1

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_PACKED](#), [P2_START_DAC](#), [P2_WRITE_DAC](#), [P2_WRITE_DAC4](#), [P2_WRITE_DAC8](#), [P2_WRITE_DAC8_PACKED](#), [P2_WRITE_DAC32](#)

Gültig für Module

AOut-4/16 Rev. E, AOut-8/16 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

module	Eingestellte Moduladresse (1...15).	LONG
array[]	Feld mit den auszugebenden Werten (0...65535).	ARRAY LONG FLOAT
index	Index des ersten auszugebenden Feldelements.	LONG

Siehe auch

P2_DAC, P2_DAC4, P2_DAC8_PACKED, P2_START_DAC, P2_WRITE_DAC, P2_WRITE_DAC4, P2_WRITE_DAC4_PACKED, P2_WRITE_DAC8_PACKED, P2_WRITE_DAC32

Gültig für Module

AOut-8/16 Rev. E

Beispiel

Beispiel

```
REM Simultane Ausgabe von vier verschiedenen Signalverläufen
REM auf den Ausgängen 1...8 eines D/A-Moduls.
REM Die Signalverläufe sind nacheinander in einem DATA-Feld
REM abgelegt und können vor dem Programmstart vom PC übergeben
REM werden.
```

```
#INCLUDE ADwinPRO_ALL.INC
DIM i AS LONG
DIM DATA_1[8000] AS LONG
```

```
INIT:
  i = 1
```

```
EVENT:
  REM Ausgaberegister DAC 1...8 setzen
  P2_WRITE_DAC8(1,DATA_1,(i-1)*8+i)
  P2_START_DAC(1) 'Ausgabe auf allen DAC starten
  INC(i)
  IF (i>1000) THEN i = 1
```

P2_WRITE_DAC8_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

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>array[]</code>	Feld mit den auszugebenden Werten (0...65535) in gepackter Form: Je 2 Werte zu 16 Bit in einem 32 Bit-Wert.	ARRAY LONG FLOAT
<code>index</code>	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	<code>array[n+3]</code>	<code>array[n+2]</code>	<code>array[n+1]</code>	<code>array[n]</code>
Bit-Nr.	31:16	15:00	31:16	15:00
Digitalwert für	DAC8	DAC7	DAC6	DAC5
			DAC4	DAC3
			DAC2	DAC1

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_PACKED](#), [P2_START_DAC](#), [P2_WRITE_DAC](#), [P2_WRITE_DAC4](#), [P2_WRITE_DAC4_PACKED](#), [P2_WRITE_DAC8](#), [P2_WRITE_DAC8_PACKED](#), [P2_WRITE_DAC32](#)

Gültig für Module

AOut-8/16 Rev. E

Beispiel

Beispiel

```
REM Simultane Ausgabe von vier verschiedenen Signalverläufen
REM auf den Ausgängen 1...8 eines D/A-Moduls.
REM Die Signalverläufe sind nacheinander in einem DATA-Feld
REM gepackt abgelegt und können vor dem Programmstart vom PC
REM übergeben werden.
```

```
#INCLUDE ADwinPRO_ALL.INC
DIM i AS LONG
DIM DATA_1[8000] AS LONG
```

```
INIT:
  i = 1
```

```
EVENT:
  REM Ausgaberegister DAC 1...8 setzen
  P2_WRITE_DAC8_PACKED (1, DATA_1, (i-1)*4+i)
  P2_START_DAC (1)          'Ausgabe auf allen DAC starten
  INC (i)
  IF (i>1000) THEN i = 1
```

P2_WRITE_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...0FFFFFFFFh).	LONG

Siehe auch

Das untere Wort (Bits 0...15) des Digitalwerts `value32` wird in den DAC mit der ungeraden Nummer geschrieben, das obere Wort (Bits 16...31) in den DAC mit der geraden Nummer.

Siehe auch

[P2_DAC](#), [P2_DAC4](#), [P2_DAC8_PACKED](#), [P2_START_DAC](#), [P2_WRITE_DAC](#), [P2_WRITE_DAC4](#), [P2_WRITE_DAC4_PACKED](#), [P2_WRITE_DAC8](#), [P2_WRITE_DAC8_PACKED](#)

Gültig für Module

AOut-4/16 Rev. E, AOut-8/16 Rev. E

Beispiel

REM Simultane Ausgabe von zwei verschiedenen Signalverläufen
REM auf den Ausgängen 3 und 4 eines D/A-Moduls.
REM Die Signalverläufe sind in zwei DATA-Feldern abgelegt und
REM können vor dem Programmstart vom PC übergeben werden.

```
#INCLUDE ADwinPRO_ALL.INC
DIM i AS LONG 'Deklaration
DIM DATA_1[1000], DATA_2[1000] AS LONG
DIM array[1000] AS LONG

INIT:
FOR i = 1 TO 1000
    array[i] = SHIFT_LEFT(DATA_2[i],16) + DATA_1[i]
NEXT i
i = 1

EVENT:
P2_WRITE_DAC32(1,2,array[i]) 'Ausgaberegister DAC 3+4 setzen
P2_START_DAC(1) 'Ausgabe auf allen DAC starten
INC(i)
IF (i>1000) THEN i=1
```

5 Programmbeispiele

Folgende Beispiele stehen zur Verfügung:

- [Online-Auswertung von Messwerten, Seite 331](#)
- [Digitaler Proportional-Regler, Seite 332](#)
- [Datenaustausch mit DATA-Feldern, Seite 332](#)
- [Digitaler PID-Regler, Seite 333](#)
- [Datenaustausch mit dem Feldbus, Seite 335](#)
- [Beispiele für RS232 und RS485:](#)
 - [RS232: Empfangen und senden, Seite 336](#)
 - [RS232: String-Befehl senden, Seite 337](#)
 - [RS232: String-Befehl empfangen, Seite 339](#)
 - [RS485: Empfangen und senden, Seite 341](#)
- [Kontinuierliche Messwertwandlung: 1 Kanal wandeln, Seite 342](#)

Die meisten Beispiele sind auch als Programmdateien im Verzeichnis <C:\ADwin\ADbasic\samples_ADwin_PRO> abgelegt.

5.1 Online-Auswertung von Messwerten

Das Programm <PRO_DMO1.BAS> sucht den Maximal- und Minimalwert aus 1000 Messungen von ADC1 und schreibt das Ergebnis in die Variablen `PAR_1` und `PAR_2`.

Benötigt wird ein 16-Bit-A/D-Modul mit Moduladresse 1 (Gruppe AD-Module) und ein Signal am Eingang 1 des Moduls.

```
#INCLUDE ADWINPRO_ALL.INC 'Include file
#define limit 65535        'max. 16 bit ADC-value
DIM 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 = 40000    'cycle-time of 1ms (T9)

EVENT:
  iw = ADC16(1,1)         'get sample
  IF (iw > max) THEN max = iw 'new maximum sample?
  IF (iw < min) THEN min = iw 'new minimum sample?
  INC il                 'increment index
  IF (il > 1000) THEN
    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
    ' ACTIVATE_PC      'only for use with TestPoint
    PAR_10 = 1         'set End-Flag
  ENDIF
```

5.2 Digitaler Proportional-Regler

Das Programm <PRO_DMO2.BAS> ist ein digitaler P-Regler. Der Sollwert wird mit `PAR_1` vorgegeben, die Verstärkung mit `PAR_2`.

Benötigt werden:

- 16-Bit-A/D-Modul mit Moduladresse 1 (Gruppe AD-Module).
- D/A-Modul mit Moduladresse 1 (Gruppe DA-Module).
- Eine Regelstrecke, die das Signal des D/A-Moduls aufnimmt und ein Signal an das A/D-Modul zurückgibt.

```
#INCLUDE ADWINPRO_ALL.INC 'Include file

#DEFINE offset 32768      '0V for 16 bit ADC/DAC-systems

REM cd: control deviation; av: actuating value
DIM cd, av AS LONG

INIT:
  PAR_1 = offset          'initial setpoint
  PAR_2 = 10              'initial gain
  PROCESSDELAY = 40000    'cycle-time of 1ms (T9)

EVENT:
  cd = PAR_1 - ADC16(1, 1) 'compute control deviation (cd)
  av = cd * PAR_2 + offset 'compute actuating value (av)
  DAC(1, 1, av)           'output actuating value on DAC-#1
```

5.3 Datenaustausch mit DATA-Feldern

Das Programm <PRO_DMO3.BAS> misst den analogen Eingang 1 des A/D-Moduls mit Adresse 1 und setzt nach 1000 Messungen eine Ende-Markierung, damit der PC erkennt, wann er die Messwerte abholen kann. Die Daten werden mit Hilfe des Felds `DATA_1` übertragen.

Benötigt wird ein 16-Bit-A/D-Modul mit Moduladresse 1 (Gruppe AD-Module).

```
#INCLUDE ADWINPRO_ALL.INC 'Include file

DIM DATA_1[1000] AS LONG
DIM index AS LONG

INIT:
  PAR_10 = 0
  index = 0          'reset array pointer
  PROCESSDELAY = 40000 'cycle-time of 1ms (T9)

EVENT:
  index = index + 1 'increment array pointer
  IF (index > 1000) THEN '1000 samples done?
  ' ACTIVATE_PC 'set ACTIVATE_PC flag (only necessary
                'for TestPoint)
  PAR_10 = 1 'set End-Flag
  END 'terminate process
ENDIF
DATA_1[index] = ADC16(1, 1) 'acquire sample and save in array
```


5.4 Digitaler PID-Regler

Das Programm <PRO_DMO6.BAS> ist ein digitaler PID-Regler.

Vor dem Starten des Reglers müssen die Reglereinstellungen in den globalen Variablen stehen.

Benötigt werden:

- A/D-Modul mit Moduladresse 1 (Gruppe AD-Module).
- D/A-Modul mit Moduladresse 1 (Gruppe DA-Module).
- Eine Regelstrecke, die das Signal des D/A-Moduls aufnimmt und ein Signal an das A/D-Modul zurückgibt.

Berechnungen im PC:

Die Reglerkoeffizienten, der Sollwert und die Abtastrate werden auf dem PC berechnet und in globalen Variablen an den Prozessor des ADwin-Pro-Systems übergeben. Auf dem gleichen Weg werden Informationen aus dem Programm an den PC zurück gegeben:

Einstellparameter des Reglers

F _{PAR_2}	Verstärkung des Reglers
F _{PAR_3}	Integrationszeit des Reglers
F _{PAR_4}	Differentiationszeit des Reglers
P _{AR_1}	Sollwert in Digits
P _{AR_6}	Abtastrate des Reglers in Einheiten zu 25ns

Informationen aus dem Programm

P _{AR_5}	Feld-Index (zu DATA_1) der Regelabweichung
P _{AR_9}	Mittlere Regelabweichung
P _{AR_10}	Flag: Alle Messungen sind durchgeführt
DATA_1 []	Feld, das die Regelabweichungen enthält

ADbasic-Programm:

Sowohl die Adresse des analogen Eingangsmoduls als auch die Adresse des analogen Ausgangsmoduls sind im Beispiel auf Adresse 1 eingestellt.

Beachten Sie, dass im Programm eine Zeitersparnis dadurch erreicht wird, dass während den erforderlichen Wartezeiten beim Einlesen der Regelabweichung (nach SET_MUX und START_CONV) die Berechnung und Ausgabe des Stellwerts durchgeführt wird.

Als Folge ergibt sich, dass jeweils der ausgegebene Stellwert aus der Regelabweichung des vorigen Prozessaufrufs berechnet wird.



```
#INCLUDE ADWINPRO_ALL.INC 'Include file

#DEFINE offset 32768      '0V output

DIM DATA_1[4000] AS LONG
DIM av, cd, cdo, sum AS LONG
DIM diff AS FLOAT

INIT:
  sum = 0                  'initial value of integral part
  cd = ADC(1)              'initial value of control deviation
                           '(cd) & MUX to Ch-#1
  PAR_5 = 1                'set array index
  IF (FPAR_3 < 1E3) THEN FPAR_3 = 1E3 'check min. of integration
                           'time
  IF (PAR_6 < 4E4) THEN PAR_6 = 4E4 'allow cycle-times >= 1ms
  PROCESSDELAY = PAR_6     'set cycle-time

EVENT:
  REM compute actuating value
  av = FPAR_2 * (cd + sum / FPAR_3 + diff * FPAR_4)
  START_CONV(1)            'start conversion ADC-#1
  REM while conversion is running ...
  DAC(1, av + offset)      'output actuating value at DAC-#1
  cdo = cd                  'keep control deviation in mind
  WAIT_EOC(1)              'wait until end-of-conversion of ADC
  cd = PAR_1 - READADC(1)   'compute control deviation
  FPAR_9 = FPAR_9 * 0.99 + cd * 0.01 'mean value of control
                                   'deviation
  sum = sum + cd           'calculate integral
  IF (sum > 2E6) THEN sum = 2E6 'positive limit of integral
  IF (sum < -2E6) THEN sum = -2E6 'negative limit of integral
  diff = (cd - cdo)        'calculate deviation difference
  DATA_1[PAR_5] = cd      'write control deviation in a buffer
  INC PAR_5                'increment buffer index
  IF (PAR_5 >= 4000) THEN   '4000 samples done?
  '  ACTIVATE_PC            'only for use with TestPoint
    PAR_10 = 1             'set End-flag
    PAR_5 = 1              'reset array index
  ENDIF

FINISH:
  DAC(1, offset)           'analog output #1 to 0V
```

5.5 Datenaustausch mit dem Feldbus

Das folgende Programm tauscht Daten mit dem Feldbus aus. Es setzt voraus, dass die Initialisierung der Feldbus-Schnittstelle bereits vorgenommen wurde. Benötigt wird ein Feldbus-Modul mit Moduladresse 1 (Gruppe EXT-Module). Das Programm fordert zyklisch (zeitgesteuert) den Zugriff auf das DP-RAM an, überprüft das Zugriffsrecht, tauscht die Daten aus und gibt den Zugriff wieder an die Busseite zurück. Vor dem eigentlichen Austausch der Daten überprüft das Programm, ob sich die Daten geändert haben und nur in diesem Fall werden sie ausgelesen und neu geschrieben.

Benötigt werden:

- Feldbusmodul mit Moduladresse 1 (Gruppe EXT-Module).
- Ein Gerät oder Programm am Feldbus, das Daten empfangen und senden kann.

```
#INCLUDE ADwinPRO_ALL.inc
DIM DATA_1[1000] AS LONG 'Feld für Eingangsdaten
DIM DATA_2[1000] AS LONG 'Feld für Ausgangsdaten
DIM n AS LONG

INIT:
  n = 1
  PAR_14 = 0
  FOR n = 1 TO 100          'Initialisierung der Sendedaten
    DATA_2[n] = n
  NEXT n

EVENT:
  IF (PAR_14 = 0) THEN
    INC PAR_8
    PAR_8 = PAR_8 AND 0ffh
    REQUEST_ACCESS(0,6)      'Zugriff auf Ein- und Ausgangsbereich
                              'beantragen

    PAR_14 = 1
  ENDIF
  IF (PAR_14 = 1) THEN
    PAR_1 = CHECK_ACCESS(0) 'Zugriffsrecht prüfen
    IF (PAR_1 = 6) THEN     'Wenn Zugriffsrecht erteilt...
      PAR_14 = 2
    ELSE
      REQUEST_ACCESS(0,6)   'sonst nochmals anfordern
    ENDIF
  ENDIF
  IF (PAR_14 = 2) THEN
    PAR_9 = CHANGED_DATA(0,32) 'auf neue Daten prüfen
    IF (PAR_9 <> 0) THEN       'wenn neue Daten vorhanden, dann...
      INC PAR_7
      DATA_2[1] = PAR_8
      SET_WRITE_BUFFER(0,DATA_2,0,60) 'Daten schreiben (60 Byte)
      GET_READ_BUFFER(0,DATA_1,0200h,40) 'Daten lesen (40 Byte)
    ENDIF
    REQUEST_RELEASE_ACCESS(0,6) 'Zugriffsrecht zurück geben
    PAR_14 = 3
  ENDIF
  IF (PAR_14 = 3) THEN
    PAR_1 = CHECK_ACCESS(0) 'ist Zugriff wieder beim Bus ?
    IF (PAR_1 = 0) THEN
      INC PAR_11             'Zugriffszyklen zählen
      PAR_14 = 0
    ENDIF
  ENDIF
```

**RS232:
Empfangen und senden**

5.6 Beispiele für RS232 und RS485

Die folgenden Beispiele sind vollständige Programme für das Senden und Empfangen von Daten und Strings mit RS232 oder RS485.

Benötigt wird ein RS232-Modul mit Moduladresse 1 (Gruppe EXT-Module).

Das Programm zeigt die Initialisierung der seriellen RS232-Schnittstelle im Abschnitt **INIT**: und das zyklische Lesen und Schreiben von Daten im Abschnitt **EVENT**:. Der Prozess ist zeitgesteuert.

REM Das Programm initialisiert die seriellen Schnittstellen im
REM Abschnitt Init:
REM Im Abschnitt Event: werden Daten zwischen den Schnittstellen
REM 1 & 2 des RS-Moduls ausgetauscht.
REM Mit Hilfe dieses Programms können die Schnittstellen
REM untereinander getestet werden. Dazu müssen Sie die
REM Schnittstellen vor dem Programmstart miteinander verbinden.

```
#INCLUDE ADwinPRO_ALL.inc
DIM DATA_1[1000] AS LONG 'Sendedaten
DIM DATA_2[1000] AS LONG 'Empfangsdaten
DIM lauf AS LONG          'Laufvariable

INIT:
  FOR lauf = 1 TO 1000      'Initialisierung der Sendedaten
    DATA_1[lauf] = lauf AND 0FFh
  NEXT lauf
  RS_INIT(1,1,9600,0,8,1,0) 'Initialisierung Schnittstelle 1:
                           '9600 Baud;
                           'Kein Paritätsbit;
                           '8 Datenbits;
                           '2 Stoppbits;
                           'kein Handshake

  RS_INIT(1,2,9600,0,8,1,0) 'Initialisierung Schnittstelle 2
                           'wie Schnittstelle 1

  PAR_1 = 1
  PAR_4 = 1

EVENT:
  REM Einen Datensatz lesen und schreiben
  IF (PAR_1 <= 1000) THEN 'Daten senden
    PAR_2 = WRITE_FIFO(1,1,DATA_1[PAR_1])
    IF (PAR_2 = 0) THEN INC PAR_1
  ENDIF

  PAR_3 = READ_FIFO(1,2)   'Daten lesen
  IF (PAR_3 <> -1) THEN
    DATA_2[PAR_4] = PAR_3
    INC PAR_4
  ENDIF
  IF (PAR_4 > 1000) THEN END 'Alle Daten sind übertragen
```

Benötigt wird ein RS232-Modul mit Moduladresse 1 (Gruppe EXT-Module).

Viele Geräte mit RS232-Schnittstelle können mit String-Befehlen gesteuert werden. Die beiden folgenden Programme zeigen, wie man mit einem Prozess eine Zeichenfolge sendet und mit einem anderen Prozess die Zeichenfolge empfängt. Die Programme sind auf der ADwin-CD verfügbar.

Die Programme können auf dem gleichen Modul, jedoch mit verschiedenen Schnittstellen eingesetzt werden. Beachten Sie bitte die Hinweise im Programmkommentar.

Das Programm `RS232_send_string.BAS` initialisiert zuerst die Schnittstelle 1. Im Abschnitt **EVENT** sendet die Schnittstelle 1 des RS-Moduls eine

RS232:

String-Befehl senden

Zeichenfolge. Im Abschnitt **FINISH** wird das Zeichen „#“ als Ende-Markierung gesendet. Es kann durch ein beliebiges anderes Zeichen ersetzt werden.

```
' Process for RS232-communication: sending a string
' ++++++
' The program may run together with RS232_receive_string.BAS
' on the same module. If so, please follow these instructions:
' - connect the interfaces with each other
' - compile and start RS232_receive_string.BAS
' - compile and start RS232_send_string.BAS

#INCLUDE ADwinPRO_ALL.inc

REM import string library
#IF PROZESSOR = T9 THEN
    IMPORT string.li9
#ELSE
    #IF PROZESSOR = T10 THEN
        IMPORT string.lia
    #ELSE
        IMPORT string.lib
    #ENDIF
#ENDIF

#DEFINE rs_adr 1          'module address
#DEFINE rs_no 1          'interface number
#DEFINE s_endchar "#"    'end marker "#"
#DEFINE s_send DATA_1
#DEFINE str_len 50        'length of send string

DIM s_send[str_len] AS STRING 'send string
DIM s_temp[1] AS STRING      'single char
DIM sp AS LONG               'send pointer

INIT:
    REM 0.25 s
    #IF PROZESSOR = T9 THEN
        PROCESSDELAY = 10000000
    #ELSE
        #IF PROZESSOR = T10 THEN
            PROCESSDELAY = 10000000
        #ELSE
            PROCESSDELAY = 75000000
        #ENDIF
    #ENDIF
    REM A reset is allowed only once on a module!
    'RS_RESET(rs_adr)          'reset RS module
    RS_INIT(rs_adr,rs_no,9600,0,8,0,0) 'init RS interface
    sp=1                       'initialize pointer
    s_send = "This is a TESTSTRING" 'send string

EVENT:
    STRMID(s_send, sp, 1, s_temp) 'read next char of string
    PAR_11 = ASC(s_temp)          'get ascii code of char
    IF (PAR_11 = 0) THEN END      'quit when all chars are sent
    PAR_12 = WRITE_FIFO(rs_adr, rs_no, PAR_11) 'send code
    REM increase pointer, else send again
    IF (PAR_12 = 0) THEN INC sp
    REM quit when all chars are sent
    IF (sp > str_len) THEN END

FINISH:
    DO                          'send end marker "#"
        PAR_11 = ASC(s_endchar) 'get ascii code
```

```
PAR_12 = WRITE_FIFO(rs_adr, rs_no, PAR_11) 'send code  
UNTIL (PAR_12 = 0)
```

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

RS232:
String-Befehl empfangen

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
```

```
#IF PROZESSOR = T9 THEN
```

```
    IMPORT string.li9
```

```
#ELSE
```

```
    #IF PROZESSOR = T10 THEN
```

```
        IMPORT string.lia
```

```
    #ELSE
```

```
        IMPORT string.lib
```

```
    #ENDIF
```

```
#ENDIF
```

```
#DEFINE rs_adr 1          'module address
```

```
#DEFINE rs_no 2           'interface number
```

```
#DEFINE s_receive DATA_2
```

```
#DEFINE str_len 50        'max. length of received string
```

```
DIM s_receive[str_len] AS STRING 'received string
```

```
DIM s_temp[1] AS STRING 'single char
```

```
DIM s_endchar[1] AS STRING 'end marker
```

```
DIM endflag AS LONG
```

```
DIM rp AS LONG            'receive pointer
```

```
INIT:
```

```
    REM 0.25 s
```

```
    #IF PROZESSOR = T9 THEN
```

```
        PROCESSDELAY = 10000000
```

```
    #ELSE
```

```
        #IF PROZESSOR = T10 THEN
```

```
            PROCESSDELAY = 10000000
```

```
        #ELSE
```

```
            PROCESSDELAY = 75000000
```

```
        #ENDIF
```

```
    #ENDIF
```

```
    RS_RESET(rs_adr)          'reset RS module
```

```
    RS_INIT(rs_adr,rs_no,9600,0,8,0,0) 'init RS interface
```

```
    rp = 0                    'initialize receive pointer
```

```
    s_receive = ""           'initialize receive string
```

```
    s_endchar = "#"          'end marker
```

```
EVENT:
```

```
    PAR_21 = READ_FIFO(rs_adr, rs_no) 'receive status / char
```

```
    IF (PAR_21 <> -1) THEN
```

```
        CHR(PAR_21,s_temp)          'get char from ascii value
```

```
        INC rp                      'increase receive pointer
```

```
        REM end marker received or string full?
```

```
        endflag = STRCOMP(s_temp, s_endchar)
```

```
        IF ((endflag=0) OR (rp>str_len)) THEN END
```

```
        s_receive = s_receive + s_temp 'save char to string
```

```
    ENDIF
```


Benötigt wird ein RS485-Modul mit Moduladresse 1 (Gruppe EXT-Module).

In diesem Beispiel wird eine RS485-Schnittstelle als passiver Teilnehmer verwendet, der alle Daten liest, die an seinem Eingang anliegen. Wenn ein bestimmter Wert (55) empfangen wird, wird die Schnittstelle aktiv und sendet dann ihrerseits fortlaufend den Wert 44.

REM Dieses Programm setzt eine RS485-Schnittstelle mit der
REM Adresse 1 voraus.

```
#INCLUDE ADwinPRO_ALL.inc
```

```
#DEFINE rs_adr 1
```

```
DIM ret_val AS LONG
```

```
DIM val AS LONG
```

```
INIT:
```

```
    rs_adr = 1  
    RS_RESET(rs_adr)  
    RS_INIT(rs_adr,2,38400,0,8,0,3)  
    RS485_SEND(rs_adr,2,0) 'Kanal 2 empfangen
```

```
EVENT:
```

```
    val = READ_FIFO(rs_adr,2) 'Daten lesen
```

```
    IF (val = 55) THEN
```

```
        RS485_SEND(rs_adr,2,1) 'Kanal 2 senden  
        ret_val = WRITE_FIFO(rs_adr,2,44) 'Daten schreiben  
    ENDIF
```

RS485:

Empfangen und senden

5.7 Kontinuierliche Messwertwandlung

Die Module Pro II AIn-F-4/14 Rev. E und Pro II AIn-F-4/14 Rev. E ermöglichen eine sehr schnelle, kontinuierliche Messwert-Wandlung. Parallel zur Wandlung müssen jedoch die Daten gelesen (und ggf. verarbeitet) werden.

Im folgenden finden Sie Beispiele für kontinuierliche Datenwandlung.

1 Kanal wandeln

Benötigt wird ein Modul Pro II AIn-F-x/14 Rev. E mit Moduladresse 1 und ein Analogsignal am Eingang 1 des Moduls.

Das Programm <PROII-F-x-14-CONT-1CH.BAS> führt auf dem Eingangskanal 1 eine kontinuierliche Burst-Messreihe mit einer Taktrate von 25MHz aus. Der Speicherbereich für die Messwerte fasst 20000 Werte.

Während der laufenden Messreihe werden Messwerte in das Feld `DATA_1` eingelesen (ein FIFO-Feld ist nicht möglich). Der Parallelbetrieb erfordert eine Abstimmung von Wandeln und Auslesen. Hierzu wird der Speicherbereich in 4 Bereiche zu 5000 Messwerten eingeteilt, und es wird nur der Bereich ausgelesen, der gerade fertig beschrieben wurde.

Ebenso ist eine Abstimmung zwischen der Wandlungsrate und der Leserate erforderlich. Der Prozesszyklus wird mit `PROCESSDELAY = 20000` (= 15kHz) so gewählt, dass die Leserate im Mittel ein Mehrfaches der Wandlungsrate 25MHz beträgt:

$$15\text{kHz} \cdot 5000 = \text{Leserate} 75\text{MHz} > \text{Wandlungsrate } 25\text{MHz}$$



Beachten Sie bitte bei einer Veränderung des Programmbeispiels: Wenn die Bearbeitungszeit des Abschnitts `EVENT`: länger wird, beispielsweise durch eine Verarbeitung der Messwerte, genügt die Leserate vielleicht nicht mehr zum Lesen der gewandelten Messwerte. In diesem Fall gehen Messwerte verloren, weil sie schneller überschrieben als gelesen werden, und Sie müssen die Abstimmung von Wandlungsrate und Leserate neu durchführen.

```
#INCLUDE ADwinPRO_ALL.inc 'include file
#DEFINE module 1         'module no.
#DEFINE d1 DATA_1       'holds values of channel 1
#DEFINE mem_idx PAR_1    'mem position of last written value
#DEFINE max_val 20000    'no. of values
#DEFINE seg1 max_val/8    'end of segment 1
#DEFINE seg2 max_val/4    'end of segment 2
#DEFINE seg3 max_val/8*3  'end of segment 3
#DEFINE blk max_val/4     'read block size

DIM d1[max_val] AS LONG  'destination array
DIM pattern AS LONG      'bit pattern to address one module
DIM segment AS LONG      'segment that is currently written

INIT:
REM 1 channel continuous, mem for max_val values, 25 MHz
P2_BURST_INIT(module,1,0,max_val,2,010b)
pattern = SHIFT_LEFT(1,module-1)'address this module only
P2_BURST_START(pattern)
segment = 1                'start with memory segment 1
PROCESSDELAY = 20000       'cycle time 66.6 µs -> 15 kHz

EVENT:
mem_idx = P2_BURST_READ_INDEX(module) 'get current mem index
IF (segment = 1) THEN      'read 1. segment
    IF ((mem_idx > seg1) AND (mem_idx < seg3)) THEN
        REM memory index is in segments 2 or 3: read segment 1
        P2_BURST_READ_UNPACKED1(module,blk,0,DATA_1,1,3)
        segment = 2
    ENDIF
ENDIF

IF (segment = 2) THEN      'read 2. segment
    IF (mem_idx > seg2) THEN
        REM memory index is in segments 3 or 4: read segment 2
        P2_BURST_READ_UNPACKED1(module,blk,seg1,DATA_1,blk+1,3)
        segment = 3
    ENDIF
ENDIF

IF (segment = 3) THEN      'read 3. segment
    IF ((mem_idx > seg3) OR (mem_idx < seg1)) THEN
        REM memory index is in segments 4 or 1: read segment 3
        P2_BURST_READ_UNPACKED1(module,blk,seg2,DATA_1,blk*2+1,3)
        segment = 4
    ENDIF
ENDIF

IF (segment = 4) THEN      'read 4. segment
    IF (mem_idx < seg2) THEN
        REM memory index is in segments 1 or 2: read segment 4
        P2_BURST_READ_UNPACKED1(module,blk,seg3,DATA_1,blk*3+1,3)
        segment = 1
    ENDIF
ENDIF
```



Anhang

A.1 Alphabetische Befehlsübersicht

A

ADC · 18
P2_ADC · 244
ADC16 · 20
P2_ADC24 · 245
ADCF · 22
P2_ADCF · 265
P2_ADCF24 · 266
P2_ADCF_MODE · 267
P2_ADCF_READ_LIMIT · 270
P2_ADCF_SET_LIMIT · 271
P2_ADC_READ_LIMIT · 246
P2_ADC_SET_LIMIT · 248

B

BURST_ABORT · 23
BURST_CREAD · 25
P2_BURST_CREAD_UNPACKED1 · 286
P2_BURST_CREAD_UNPACKED2 · 288
P2_BURST_CREAD_UNPACKED4 · 290
P2_BURST_CREAD_UNPACKED8 · 292
BURST_CSTART · 27
BURST_INIT · 28
P2_BURST_INIT · 294
BURST_READ · 30
P2_BURST_READ · 299
P2_BURST_READ_INDEX · 297
BURST_READ_PACKED · 32
P2_BURST_READ_UNPACKED1 · 301
P2_BURST_READ_UNPACKED2 · 303
P2_BURST_READ_UNPACKED4 · 305
P2_BURST_READ_UNPACKED8 · 307
P2_BURST_RESET · 309
BURST_START · 34
P2_BURST_START · 311
BURST_STATUS · 36
P2_BURST_STATUS · 312
P2_BURST_STOP · 314

C

CAN_MSG · 181
CHANGED_DATA · 197
CHECKLED · 4
CHECK_ACCESS · 198
P2_CHECK_LED · 227
CHECK_SHIFT_REG · 208
CNT_CLEAR · 87
CNT_ENABLE · 88
CNT_LATCH · 89
CNT_READ16 · 90
CNT_READ32 · 91
CNT_READLATCH16 · 92
CNT_READLATCH32 · 93
CNT_SETMODE · 94

CO4_CLEARENABLE · 95
CO4_GETSTATUS · 96
CO4_LATCHENABLE · 98
CO4_READ · 99
CO4_READLATCH · 100
CO4_RESETSTATUS · 101
CO4_SETMODE · 103
CO4_SET_LATCHMODE · 102
COMP_DIGIN_WORD · 141
COMP_DIGIN_WORD_DIFF · 142
COMP_FIFO_READ · 143
COMP_FIFO_SELECT · 144
COMP_READ · 145
COMP_RESET · 146
COMP_SET · 147
CPU_DIGIN (T11) · 228
CPU_DIGIN (T9, T10) · 5
CPU_DIGOUT · 229
CPU_DIG_IO_CONFIG · 230
CPU_EVENT_CONFIG · 231

D

DAC · 71
P2_DAC · 318
P2_DAC4 · 319
P2_DAC4_PACKED · 320
P2_DAC8 · 322
P2_DAC8_PACKED · 323
DIGIN_LONG_F · 114
DIGIN_WORD1 · 115
DIGIN_WORD2 · 116
DIGOUT · 117
DIGOUT_BITS_F · 119
DIGOUT_F · 121
DIGOUT_LONG_F · 122
DIGOUT_WORD1 · 123
DIGOUT_WORD2 · 124
DIGPROG1 · 125
DIGPROG2 · 126
DIG_LATCH · 105
DIG_READLATCH1 · 107
DIG_READLATCH2 · 108
DIG_WRITELATCH1 · 109
DIG_WRITELATCH2 · 111
DIG_WRITELATCH32 · 113

E

EN_INTERRUPT · 183
EN_RECEIVE · 184
EN_TRANSMIT · 185
P2_EVENT2_CONFIG · 234
EVENTENABLE · 6
P2_EVENT_CONFIG · 233
P2_EVENT_ENABLE · 232

P2_EVENT_READ · 236
EXTLCH_ENABLE · 127

F

FG_CONTROL · 72
FG_DEF · 74
FG_DELAY · 76
FG_MODE · 77
FG_READ_INDEX · 79
FG_STATUS · 81
FG_WRITE · 82

G

GET_CAN_REG · 186
GET_DIGOUT_LONG · 128
GET_DIGOUT_WORD1 · 129
GET_DIGOUT_WORD2 · 130
GET_PRO_BYTE · 199
GET_READ_BUFFER · 200
GET_RS · 209

I

INIT_CAN · 187
INIT_SLAVE · 201

L

LS_DIGPROG · 220
LS_DIG_IO · 224
LS_DIO_INIT · 218
LS_WATCHDOG_INIT · 222

M

MEDIA_RD_BLK_F · 161
MEDIA_RD_BLK_L · 157
MEDIA_RD_FILEINFO · 163
MEDIA_WR_BLK_F · 155
MEDIA_WR_BLK_L · 151

P

PT100_DIG_TO_R · 167
PT100_DIG_TO_TEMP · 166
PWM_ENABLE · 131
PWM_OUT · 132
PWM_SET · 133

R

READADC · 38
READADCF · 40
READADCF_32 · 45
READADCF_SCONV · 46
READADCF_SCONV_32 · 47
READADC_SCONV · 39
P2_READ_ADC · 249
P2_READ_ADC24 · 250
P2_READ_ADCF · 272
P2_READ_ADCF24 · 273
P2_READ_ADCF32 · 280

P2_READ_ADCF4 · 274
READ_ADCF4 · 41
P2_READ_ADCF4_24B · 275
P2_READ_ADCF4_PACKED · 278
READ_ADCF4_PACKED · 43
P2_READ_ADCF8 · 276
READ_ADCF8 · 42
P2_READ_ADCF8_24B · 277
P2_READ_ADCF8_PACKED · 279
READ_ADCF8_PACKED · 44
P2_READ_ADCF_SCONV · 281
P2_READ_ADCF_SCONV24 · 282
P2_READ_ADCF_SCONV32 · 283
P2_READ_ADC_SCONV · 251
P2_READ_ADC_SCONV24 · 252
READ_FIFO · 210
READ_MSG · 188
REQUEST_ACCESS · 204
REQUEST_RELEASE_ACCESS · 205
RESETWATCHDOGTIMER · 7
RS485_SEND · 214
RS_INIT · 211
RS_RESET · 213
RTC_GET · 150
RTC_SET · 149

S

P2_SEQ_INIT · 254
SEQ_MODE · 52
P2_SEQ_READ · 257
SEQ_READ · 54
P2_SEQ_READ24 · 258
SEQ_READ32 · 59
SEQ_READ_ONE · 55
P2_SEQ_READ_PACKED · 259
SEQ_READ_PACKED · 58
SEQ_READ_TWO · 56
SEQ_SELECT · 60
SEQ_SET_DELAY · 61
P2_SEQ_START · 260
SEQ_STATUS · 62
P2_SEQ_WAIT · 261
SETLED · 8
P2_SET_AVERAGE_FILTER · 316
SET_CAN_BAUDRATE · 190
SET_CAN_REG · 195
SET_GAIN · 49
P2_SET_LED · 237
P2_SET_MUX · 262
SET_MUX · 50
SET_PRO_BYTE · 206
SET_RS · 215
SET_WRITE_BUFFER · 207
P2_SE_DIFF · 253
SE_DIFF · 48
SH_SETMODE · 63
SSI_MODE · 134
SSI_READ · 135
SSI_SET_BITS · 136

SSI_SET_CLOCK · 137
SSI_START · 138
SSI_STATUS · 139
STARTWATCHDOG · 10
P2_START_CONV · 263
START_CONV · 64
P2_START_CONVF · 284
START_CONVF · 65
P2_START_DAC · 324
START_DAC · 84
STOPWATCHDOG · 11
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
P2_SYNC_ALL · 238
P2_SYNC_ENABLE · 240
SYNC_MODE · 66
P2_SYNC_STAT · 242

T

TCJ_DIG_TO_TEMP · 170
TCK_DIG_TO_TEMP · 171
TC_READ_B · 172
TC_READ_E · 173
TC_READ_J · 174
TC_READ_K · 175
TC_READ_N · 176
TC_READ_R · 177
TC_READ_S · 178
TC_READ_T · 179
TC_SELECT · 168
TC_SET_RATE · 180
TRANSMIT · 196

W

P2_WAIT_EOC · 264
WAIT_EOC · 68
P2_WAIT_EOCF · 285
WAIT_EOCF · 69
WRITEDAC · 85
P2_WRITE_DAC · 325
P2_WRITE_DAC32 · 330
P2_WRITE_DAC4 · 326
P2_WRITE_DAC4_PACKED · 327
P2_WRITE_DAC8 · 328
P2_WRITE_DAC8_PACKED · 329
WRITE_FIFO · 216

A.2 Befehlsübersicht nach Modulen

Sie finden die Befehlsübersichten der Module auf den folgenden Seiten:

von Modul	bis Modul	Seite
AIn-16/14-C Rev. A	AIn-32/14 Rev. A	A-5
AIn-32/16 Rev. B	AIn-8/12 Rev. B	A-6
AIn-8/14 Rev. A	AIn-8/18 Rev. E	A-7
AIn-F-4/12 Rev. A	AIn-F-8/12 Rev. A	A-8
AIn-F-8/14 Rev. B	AIn-F-8/16 Rev. A	A-9
AIn-F-8/18 Rev. E	AOut-8/16 Rev. A	A-10
AOut-8/16 Rev. B	CNT-PW4(-I)	A-11
CNT-VR2PW2	CPU-T11	A-12
CPU-T9	LS2 Rev. A	A-13
OPT-16 Rev. A	RS485-2, RS485-4	A-14
Storage Rev. A	(LP)SH-4/8(-FI)	A-15

AIn-16/14-C Rev. A

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SEQ_MODE · 52
 SEQ_READ · 54
 SEQ_READ32 · 59
 SEQ_READ_ONE · 55
 SEQ_READ_PACKED · 58
 SEQ_READ_TWO · 56
 SEQ_SELECT · 60
 SEQ_SET_DELAY · 61
 SEQ_STATUS · 62
 SETLED · 8
 SET_MUX · 50
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

AIn-32/12 Rev. A

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
S: SETLED · 8
 SET_MUX · 50
 SE_DIFF · 48
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

AIn-32/12 Rev. B

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SETLED · 8
 SET_MUX · 50
 SE_DIFF · 48
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

AIn-32/14 Rev. A

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SEQ_MODE · 52
 SEQ_READ · 54
 SEQ_READ32 · 59
 SEQ_READ_ONE · 55
 SEQ_READ_PACKED · 58
 SEQ_READ_TWO · 56
 SEQ_SELECT · 60
 SEQ_SET_DELAY · 61
 SEQ_STATUS · 62
 SETLED · 8
 SET_MUX · 50
 SE_DIFF · 48
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-32/16 Rev. B

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SETLED · 8
 SET_MUX · 50
 SE_DIFF · 48
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-32/16 Rev. C

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SEQ_MODE · 52
 SEQ_READ · 54
 SEQ_READ32 · 59
 SEQ_READ_ONE · 55
 SEQ_READ_PACKED · 58
 SEQ_READ_TWO · 56
 SEQ_SELECT · 60
 SEQ_SET_DELAY · 61
 SEQ_STATUS · 62
 SETLED · 8
 SET_MUX · 50
 SE_DIFF · 48
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-32/18 Rev. E

A: P2_ADC · 244
 P2_ADC24 · 245
 P2_ADC_READ_LIMIT · 246
 P2_ADC_SET_LIMIT · 248
C: P2_CHECK_LED · 227
R: P2_READ_ADC · 249
 P2_READ_ADC24 · 250
 P2_READ_ADC_SCONV · 251
 P2_READ_ADC_SCONV24 · 252
S: P2_SEQ_INIT · 254
 P2_SEQ_READ · 257
 P2_SEQ_READ24 · 258
 P2_SEQ_READ_PACKED · 259
 P2_SEQ_START · 260
 P2_SEQ_WAIT · 261
 P2_SET_LED · 237
 P2_SET_MUX · 262
 P2_SE_DIFF · 253
 P2_START_CONV · 263
 P2_SYNC_ALL · 238
W: P2_WAIT_EOC · 264

Aln-8/12 Rev. A

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
S: SETLED · 8
 SET_MUX · 50
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-8/12 Rev. B

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SETLED · 8
 SET_MUX · 50
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-8/14 Rev. A

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SEQ_MODE · 52
 SEQ_READ · 54
 SEQ_READ32 · 59
 SEQ_READ_ONE · 55
 SEQ_READ_PACKED · 58
 SEQ_READ_TWO · 56
 SEQ_SELECT · 60
 SEQ_SET_DELAY · 61
 SEQ_STATUS · 62
 SETLED · 8
 SET_MUX · 50
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-8/16 Rev. A

A: ADC16 · 20
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SETLED · 8
 SET_MUX · 50
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-8/16 Rev. B

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SETLED · 8
 SET_MUX · 50
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-8/16 Rev. C

A: ADC · 18
C: CHECKLED · 4
R: READADC · 38
 READADC_SCONV · 39
S: SEQ_MODE · 52
 SEQ_READ · 54
 SEQ_READ32 · 59
 SEQ_READ_ONE · 55
 SEQ_READ_PACKED · 58
 SEQ_READ_TWO · 56
 SEQ_SELECT · 60
 SEQ_SET_DELAY · 61
 SEQ_STATUS · 62
 SETLED · 8
 SET_MUX · 50
 START_CONV · 64
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOC · 68

Aln-8/18 Rev. E

A: P2_ADC · 244
 P2_ADC24 · 245
 P2_ADC_READ_LIMIT · 246
 P2_ADC_SET_LIMIT · 248
C: P2_CHECK_LED · 227
R: P2_READ_ADC · 249
 P2_READ_ADC24 · 250
 P2_READ_ADC_SCONV · 251
 P2_READ_ADC_SCONV24 · 252
S: P2_SEQ_INIT · 254
 P2_SEQ_READ · 257
 P2_SEQ_READ24 · 258
 P2_SEQ_READ_PACKED · 259
 P2_SEQ_START · 260
 P2_SEQ_WAIT · 261
 P2_SET_LED · 237
 P2_SET_MUX · 262
 P2_START_CONV · 263
 P2_SYNC_ALL · 238
W: P2_WAIT_EOC · 264

Aln-F-4/12 Rev. A

A: ADCF · 22
C: CHECKLED · 4
R: READADCF · 40
 READADCF_32 · 45
 READADCF_SCONV · 46
 READADCF_SCONV_32 · 47
 READ_ADCF4 · 41
 READ_ADCF4_PACKED · 43
S: SETLED · 8
 START_CONVF · 65
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOCF · 69

Aln-F-4/14 Rev. B

A: ADCF · 22
B: BURST_ABORT · 23
 BURST_CREAD · 25
 BURST_CSTART · 27
 BURST_INIT · 28
 BURST_READ · 30
 BURST_READ_PACKED · 32
 BURST_START · 34
 BURST_STATUS · 36
C: CHECKLED · 4
R: READADCF · 40
 READADCF_32 · 45
 READADCF_SCONV · 46
 READADCF_SCONV_32 · 47
 READ_ADCF4 · 41
 READ_ADCF4_PACKED · 43
S: SETLED · 8
 SET_GAIN · 49
 START_CONVF · 65
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
 SYNC_MODE · 66
W: WAIT_EOCF · 69

Aln-F-4/16 Rev. A

A: ADCF · 22
C: CHECKLED · 4
R: READADCF · 40
 READADCF_32 · 45
 READADCF_SCONV · 46
 READADCF_SCONV_32 · 47
 READ_ADCF4 · 41
 READ_ADCF4_PACKED · 43
S: SETLED · 8
 START_CONVF · 65
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOCF · 69

Aln-F-8/12 Rev. A

A: ADCF · 22
C: CHECKLED · 4
R: READADCF · 40
 READADCF_32 · 45
 READADCF_SCONV · 46
 READADCF_SCONV_32 · 47
 READ_ADCF4 · 41
 READ_ADCF4_PACKED · 43
 READ_ADCF8 · 42
 READ_ADCF8_PACKED · 44
S: SETLED · 8
 START_CONVF · 65
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16
W: WAIT_EOCF · 69

Aln-F-8/14 Rev. B

- A:** ADCF · 22
- B:** BURST_ABORT · 23
BURST_CREAD · 25
BURST_CSTART · 27
BURST_INIT · 28
BURST_READ · 30
BURST_READ_PACKED · 32
BURST_START · 34
BURST_STATUS · 36
- C:** CHECKLED · 4
- R:** READADCF · 40
READADCF_32 · 45
READADCF_SCONV · 46
READADCF_SCONV_32 · 47
READ_ADCF4 · 41
READ_ADCF4_PACKED · 43
READ_ADCF8 · 42
READ_ADCF8_PACKED · 44
- S:** SETTLED · 8
SET_GAIN · 49
START_CONVF · 65
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
SYNC_MODE · 66
- W:** WAIT_EOCF · 69

Aln-F-8/14 Rev. E

- A:** P2_ADCF_READ_LIMIT · 270
P2_ADCF_SET_LIMIT · 271
- B:** P2_BURST_CREAD_UNPACKED1 · 286
P2_BURST_CREAD_UNPACKED2 · 288
P2_BURST_CREAD_UNPACKED4 · 290
P2_BURST_CREAD_UNPACKED8 · 292
P2_BURST_INIT · 294
P2_BURST_READ · 299
P2_BURST_READ_INDEX · 297
P2_BURST_READ_UNPACKED1 · 301
P2_BURST_READ_UNPACKED2 · 303
P2_BURST_READ_UNPACKED4 · 305
P2_BURST_READ_UNPACKED8 · 307
P2_BURST_RESET · 309
P2_BURST_START · 311
P2_BURST_STATUS · 312
P2_BURST_STOP · 314
- C:** P2_CHECK_LED · 227
- E:** P2_EVENT2_CONFIG · 234
P2_EVENT_CONFIG · 233
P2_EVENT_ENABLE · 232
P2_EVENT_READ · 236
- R:** P2_READ_ADCF · 272
P2_READ_ADCF32 · 280
P2_READ_ADCF4 · 274
P2_READ_ADCF4_PACKED · 278
P2_READ_ADCF8 · 276
P2_READ_ADCF8_PACKED · 279
- S:** P2_SET_AVERAGE_FILTER · 316
P2_SET_LED · 237
P2_SYNC_ALL · 238
P2_SYNC_ENABLE · 240
P2_SYNC_STAT · 242

Aln-F-8/16 Rev. A

- A:** ADCF · 22
- C:** CHECKLED · 4
- R:** READADCF · 40
READADCF_32 · 45
READADCF_SCONV · 46
READADCF_SCONV_32 · 47
READ_ADCF4 · 41
READ_ADCF4_PACKED · 43
READ_ADCF8 · 42
READ_ADCF8_PACKED · 44
- S:** SETTLED · 8
START_CONVF · 65
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
- W:** WAIT_EOCF · 69

Aln-F-8/18 Rev. E

- A:** P2_ADCF · 265
P2_ADCF24 · 266
P2_ADCF_MODE · 267
P2_ADCF_READ_LIMIT · 270
P2_ADCF_SET_LIMIT · 271
- C:** P2_CHECK_LED · 227
- E:** P2_EVENT2_CONFIG · 234
P2_EVENT_CONFIG · 233
P2_EVENT_ENABLE · 232
P2_EVENT_READ · 236
- R:** P2_READ_ADCF · 272
P2_READ_ADCF24 · 273
P2_READ_ADCF32 · 280
P2_READ_ADCF4 · 274
P2_READ_ADCF4_24B · 275
P2_READ_ADCF4_PACKED · 278
P2_READ_ADCF8 · 276
P2_READ_ADCF8_24B · 277
P2_READ_ADCF8_PACKED · 279
P2_READ_ADCF_SCONV · 281
P2_READ_ADCF_SCONV24 · 282
P2_READ_ADCF_SCONV32 · 283
- S:** P2_SET_LED · 237
P2_START_CONV · 284
P2_SYNC_ALL · 238
P2_SYNC_ENABLE · 240
P2_SYNC_STAT · 242
- W:** P2_WAIT_EOCF · 285

AOut-16/8-12

- A:** ADC · 18
- C:** CHECKLED · 4
- D:** DAC · 71
- R:** READADC · 38
READADC_SCONV · 39
- S:** SETLED · 8
SET_MUX · 50
START_CONV · 64
START_DAC · 84
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
- W:** WAIT_EOC · 68
WRITEDAC · 85

AOut-4/16 Rev. A

- C:** CHECKLED · 4
- D:** DAC · 71
- S:** SETLED · 8
START_DAC · 84
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
- W:** WRITEDAC · 85

AOut-4/16 Rev. B

- C:** CHECKLED · 4
- D:** DAC · 71
- S:** SETLED · 8
START_DAC · 84
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
- W:** WRITEDAC · 85

AOut-4/16 Rev. C

- C:** CHECKLED · 4
- D:** DAC · 71
- F:** FG_CONTROL (nur AOut-4/16-M2) · 72
FG_DEF (nur AOut-4/16-M2) · 74
FG_DELAY (nur AOut-4/16-M2) · 76
FG_Mode (nur AOut-4/16-M2) · 77
FG_READ_INDEX (nur AOut-4/16-M2) · 79
FG_STATUS (nur AOut-4/16-M2) · 81
FG_WRITE (nur AOut-4/16-M2) · 82
- S:** SETLED · 8
START_DAC · 84
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
- W:** WRITEDAC · 85

AOut-4/16 Rev. E

- C:** P2_CHECK_LED · 227
- D:** P2_DAC · 318
P2_DAC4 · 319
P2_DAC4_PACKED · 320
- E:** P2_EVENT_CONFIG · 233
P2_EVENT_ENABLE · 232
P2_EVENT_READ · 236
- S:** P2_SET_LED · 237
P2_START_DAC · 324
P2_SYNC_ALL · 238
P2_SYNC_ENABLE · 240
P2_SYNC_STAT · 242
- W:** P2_WRITE_DAC · 325
P2_WRITE_DAC32 · 330
P2_WRITE_DAC4 · 326
P2_WRITE_DAC4_PACKED · 327

AOut-8/16 Rev. A

- C:** CHECKLED · 4
- D:** DAC · 71
- S:** SETLED · 8
START_DAC · 84
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
- W:** WRITEDAC · 85

AOut-8/16 Rev. B

C: CHECKLED · 4
D: DAC · 71
S: SETLED · 8
START_DAC · 84
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
W: WRITEDAC · 85

AOut-8/16 Rev. C

C: CHECKLED · 4
D: DAC · 71
S: SETLED · 8
START_DAC · 84
SYNCSTAT · 16
W: WRITEDAC · 85

AOut-8/16 Rev. E

C: P2_CHECK_LED · 227
D: P2_DAC · 318
P2_DAC4 · 319
P2_DAC4_PACKED · 320
P2_DAC8 · 322
P2_DAC8_PACKED · 323
E: P2_EVENT_CONFIG · 233
P2_EVENT_ENABLE · 232
P2_EVENT_READ · 236
S: P2_SET_LED · 237
P2_START_DAC · 324
P2_SYNC_ALL · 238
P2_SYNC_ENABLE · 240
P2_SYNC_STAT · 242
W: P2_WRITE_DAC · 325
P2_WRITE_DAC32 · 330
P2_WRITE_DAC4 · 326
P2_WRITE_DAC4_PACKED · 327
P2_WRITE_DAC8 · 328
P2_WRITE_DAC8_PACKED · 329

CAN-1, CAN-2

C: CAN_MSG · 181
CHECKLED · 4
E: EN_INTERRUPT · 183
EN_RECEIVE · 184
EN_TRANSMIT · 185
G: GET_CAN_REG · 186
I: INIT_CAN · 187
R: READ_MSG · 188
S: SETLED · 8
SET_CAN_BAUDRATE · 190
SET_CAN_REG · 195
T: TRANSMIT · 196

CNT-16/16(-I)

C: CHECKLED · 4
CNT_CLEAR · 87
CNT_ENABLE · 88
CNT_LATCH · 89
CNT_READ16 · 90
CNT_READLATCH16 · 92
E: EVENTENABLE · 6
S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

CNT-16/32(-I)

C: CHECKLED · 4
CNT_CLEAR · 87
CNT_ENABLE · 88
CNT_LATCH · 89
CO4_READ · 99
CO4_READLATCH · 100
E: EVENTENABLE · 6
S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

CNT-8/32(-I)

C: CHECKLED · 4
CNT_CLEAR · 87
CNT_ENABLE · 88
CNT_LATCH · 89
CNT_READ32 · 91
CNT_READLATCH32 · 93
E: EVENTENABLE · 6
S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

CNT-PW4(-I)

C: CHECKLED · 4
CNT_CLEAR · 87
CNT_ENABLE · 88
CNT_LATCH · 89
CNT_READ32 · 91
CNT_READLATCH32 · 93
E: EVENTENABLE · 6
S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

CNT-VR2PW2

C: CNT_CLEAR · 87
 CNT_ENABLE · 88
 CNT_LATCH · 89
 CNT_READ32 · 91
 CNT_READLATCH32 · 93
 CNT_SETMODE · 94

CNT-VR4L(-I)

C: CHECKLED · 4
 CNT_CLEAR · 87
 CNT_ENABLE · 88
 CNT_LATCH · 89
 CNT_READ32 · 91
 CNT_READLATCH32 · 93
 CNT_SETMODE · 94

E: EVENTENABLE · 6
 EXTLCH_ENABLE · 127

S: SETLED · 8
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16

CNT-VR4(-I)

C: CHECKLED · 4
 CNT_CLEAR · 87
 CNT_ENABLE · 88
 CNT_LATCH · 89
 CNT_READ32 · 91
 CNT_READLATCH32 · 93
 CNT_SETMODE · 94

E: EVENTENABLE · 6

S: SETLED · 8
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16

CO4

C: CHECKLED · 4
 CNT_CLEAR · 87
 CNT_ENABLE · 88
 CNT_LATCH · 89
 CO4_CLEARENABLE · 95
 CO4_GETSTATUS · 96
 CO4_LATCHENABLE · 98
 CO4_READ · 99
 CO4_READLATCH · 100
 CO4_RESETSTATUS · 101
 CO4_SETMODE · 103
 CO4_SET_LATCHMODE · 102

E: EVENTENABLE · 6

S: SETLED · 8
 SSI_MODE · 134
 SSI_READ · 135
 SSI_SET_BITS · 136
 SSI_SET_CLOCK · 137
 SSI_START · 138
 SSI_STATUS · 139
 SYNCALL · 12
 SYNCENABLE · 14
 SYNCSTAT · 16

COMP16 Rev. A

C: CHECKLED · 4
 COMP_DIGIN_WORD · 141
 COMP_DIGIN_WORD_DIFF · 142
 COMP_FIFO_READ · 143
 COMP_FIFO_SELECT · 144
 COMP_READ · 145
 COMP_RESET · 146
 COMP_SET · 147

S: SETLED · 8

CPU-T10

C: CHECKLED · 4
 CPU_DIGIN · 5

R: RESETWATCHDOGTIMER · 7

S: SETLED · 8
 STARTWATCHDOG · 10
 STOPWATCHDOG · 11

CPU-T11

C: CPU_DIGIN · 228
 CPU_DIGOUT · 229
 CPU_DIG_IO_CONFIG · 230
 CPU_EVENT_CONFIG · 231
 P2_CHECK_LED · 227

R: RESETWATCHDOGTIMER · 7

S: P2_SET_LED · 237
 STARTWATCHDOG · 10
 STOPWATCHDOG · 11

CPU-T9

- C: CHECKLED · 4
CPU_DIGIN (Bestelloption des Moduls) · 5
- R: RESETWATCHDOGTIMER · 7
- S: SETLED · 8
STARTWATCHDOG · 10
STOPWATCHDOG · 11

DIO-32

- C: CHECKLED · 4
- D: DIGIN_WORD1 · 115
DIGIN_WORD2 · 116
DIGOUT · 117
DIGOUT_WORD1 · 123
DIGOUT_WORD2 · 124
DIGPROG1 · 125
DIGPROG2 · 126
DIG_LATCH · 105
DIG_READLATCH1 · 107
DIG_READLATCH2 · 108
DIG_WRITELATCH1 · 109
DIG_WRITELATCH2 · 111
DIG_WRITELATCH32 · 113
- E: EVENTENABLE · 6
- G: GET_DIGOUT_LONG · 128
GET_DIGOUT_WORD1 · 129
GET_DIGOUT_WORD2 · 130
- S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

DIO-32 Rev. B

- C: CHECKLED · 4
- D: DIGIN_LONG_F · 114
DIGIN_WORD1 · 115
DIGIN_WORD2 · 116
DIGOUT · 117
DIGOUT_BITS_F · 119
DIGOUT_F · 121
DIGOUT_LONG_F · 122
DIGOUT_WORD1 · 123
DIGOUT_WORD2 · 124
DIGPROG1 · 125
DIGPROG2 · 126
DIG_LATCH · 105
DIG_READLATCH1 · 107
DIG_READLATCH2 · 108
DIG_WRITELATCH1 · 109
DIG_WRITELATCH2 · 111
DIG_WRITELATCH32 · 113
- E: EVENTENABLE · 6
- G: GET_DIGOUT_WORD1 · 129
GET_DIGOUT_WORD2 · 130
- S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

Inter-SL

- C: CHANGED_DATA · 197
CHECKLED · 4
CHECK_ACCESS · 198
- G: GET_PRO_BYTE · 199
GET_READ_BUFFER · 200
- I: INIT_SLAVE · 201
- R: REQUEST_ACCESS · 204
REQUEST_RELEASE_ACCESS · 205
- S: SETLED · 8
SET_PRO_BYTE · 206
SET_WRITE_BUFFER · 207

LS-2 Rev. A

- L: LS_DIGPROG · 220
LS_DIG_IO · 224
LS_DIO_INIT · 218
LS_WATCHDOG_INIT · 222

LS2 Rev. A

- C: CHECKLED · 4
- S: SETLED · 8

OPT-16 Rev. A

- C: CHECKLED · 4
- D: DIGIN_WORD1 · 115
DIG_LATCH · 105
DIG_READLATCH1 · 107
- E: EVENTENABLE · 6
- S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

Profi-SL

- C: CHANGED_DATA · 197
CHECKLED · 4
CHECK_ACCESS · 198
- G: GET_PRO_BYTE · 199
GET_READ_BUFFER · 200
- I: INIT_SLAVE · 201
- R: REQUEST_ACCESS · 204
REQUEST_RELEASE_ACCESS · 205
- S: SETLED · 8
SET_PRO_BYTE · 206
SET_WRITE_BUFFER · 207

PT100-4, PT100-8

- C: CHECKLED · 4
- P: PT100_DIG_TO_R · 167
PT100_DIG_TO_TEMP · 166
- S: SETLED · 8
- T: TC_SELECT · 168

PWM-4(-I)

- C: CHECKLED · 4
- E: EVENTENABLE · 6
- P: PWM_ENABLE · 131
PWM_OUT · 132
PWM_SET · 133
- S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

REL-16

- C: CHECKLED · 4
- D: DIGOUT · 117
DIGOUT_WORD1 · 123
DIG_LATCH · 105
DIG_WRITELATCH1 · 109
- E: EVENTENABLE · 6
- G: GET_DIGOUT_WORD1 · 129
- S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

RS232-2, RS232-4

- C: CHECKLED · 4
CHECK_SHIFT_REG · 208
- G: GET_RS · 209
- R: READ_FIFO · 210
RS_INIT · 211
RS_RESET · 213
- S: SETLED · 8
SET_RS · 215
- W: WRITE_FIFO · 216

RS422-2, RS422-4

- C: CHECK_SHIFT_REG · 208
- G: GET_RS · 209
- R: READ_FIFO · 210
RS_INIT · 211
RS_RESET · 213
- S: SET_RS · 215
- W: WRITE_FIFO · 216

RS485-2, RS485-4

- C: CHECKLED · 4
CHECK_SHIFT_REG · 208
- G: GET_RS · 209
- R: READ_FIFO · 210
RS485_SEND · 214
RS_INIT · 211
RS_RESET · 213
- S: SETLED · 8
SET_RS · 215
- W: WRITE_FIFO · 216

Storage Rev. A

- C: CHECKLED · 4
- M: MEDIA_RD_BLK_F · 161
MEDIA_RD_BLK_L · 157
MEDIA_RD_FILEINFO · 163
MEDIA_WR_BLK_F · 155
MEDIA_WR_BLK_L · 151
- R: RTC_GET · 150
RTC_SET · 149
- S: SETLED · 8

TC-16

- C: CHECKLED · 4
- S: SETLED · 8
- T: TCJ_DIG_TO_TEMP · 170
TCK_DIG_TO_TEMP · 171
TC_SELECT · 168

TC-4

- C: CHECKLED · 4
- S: SETLED · 8
- T: TCJ_DIG_TO_TEMP · 170
TCK_DIG_TO_TEMP · 171
TC_SELECT · 168

TC-8

- C: CHECKLED · 4
- S: SETLED · 8
- T: TCJ_DIG_TO_TEMP · 170
TCK_DIG_TO_TEMP · 171
TC_SELECT · 168

TC-8-ISO

- C: CHECKLED · 4
- S: SETLED · 8
- T: TC_READ_B · 172
TC_READ_E · 173
TC_READ_J · 174
TC_READ_K · 175
TC_READ_N · 176
TC_READ_R · 177
TC_READ_S · 178
TC_READ_T · 179
TC_SET_RATE · 180

TRA-16 Rev. A

- C: CHECKLED · 4
- D: DIGOUT · 117
DIGOUT_WORD1 · 123
DIG_LATCH · 105
DIG_WRITELATCH1 · 109
- E: EVENTENABLE · 6
- G: GET_DIGOUT_WORD1 · 129
- S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

TRA-16 Rev. B

- C: CHECKLED · 4
- D: DIGOUT · 117
DIGOUT_BITS_F · 119
DIGOUT_F · 121
DIGOUT_WORD1 · 123
DIG_LATCH · 105
DIG_WRITELATCH1 · 109
- E: EVENTENABLE · 6
- G: GET_DIGOUT_WORD1 · 129
- S: SETLED · 8
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16

(LP)SH-4/8(-FI)

- A: ADC · 18
ADC16 · 20
- C: CHECKLED · 4
- R: READADC · 38
READADC_SCONV · 39
- S: SETLED · 8
SET_MUX · 50
SH_SETMODE · 63
START_CONV · 64
SYNCALL · 12
SYNCENABLE · 14
SYNCSTAT · 16
- W: WAIT_EOC · 68

A.3 Thematische Befehlsübersicht

Die Befehle sind in die folgenden Themengruppen aufgeteilt. Innerhalb der Themengruppen sind die Befehle alphabetisch sortiert.

- Analoge Ausgänge: Seite [A-17](#)
- Analoge Eingänge: Seite [A-18](#)
- Digitale Ein-/Ausgänge: Seite [A-20](#)
- Kommunikationsschnittstelle: Seite [A-21](#)
- Komparator: Seite [A-22](#)
- Speichermedium: Seite [A-22](#)
- System: Seite [A-23](#)
- Temperatur-Eingänge: Seite [A-23](#)
- Zähler: Seite [A-24](#)

Analoge Ausgänge

P2_DAC	gibt auf einem Kanal des angegebenen Moduls eine (analoge) Spannung aus, die dem angegebenen Digitalwert entspricht.
DAC	gibt auf einem Kanal des angegebenen Moduls eine (analoge) Spannung aus, die dem angegebenen Digitalwert entspricht.
P2_DAC4	gibt 4 Digitalwerte aus einem Feld auf die DAC 1...4 des angegebenen Moduls als (analoge) Spannung aus.
P2_DAC4_PACKED	gibt 4 gepackte Digitalwerte aus einem Feld auf die DAC 1...4 des angegebenen Moduls als (analoge) Spannung aus.
P2_DAC8	gibt 8 Digitalwerte aus einem Feld auf die DAC 1...8 des angegebenen Moduls als (analoge) Spannung aus.
P2_DAC8_PACKED	gibt die Digitalwerte aus einem Feld auf den DAC 1...8 des angegebenen Moduls als (analoge) Spannung aus.
FG_CONTROL	startet oder stoppt den Funktionsgenerator (d.h. die Werteausgabe) auf den gewählten Ausgabekanälen des angegebenen Moduls.
FG_DEF	definiert für einen Ausgabekanal auf dem angegebenen Modul die Startadresse und die Größe des internen Zwischenspeichers für den Funktionsgenerator-Modus.
FG_DELAY	stellt auf dem angegebenen Modul die Ausgaberate des Funktionsgenerators ein.
FG_MODE	aktiviert oder deaktiviert auf dem angegebenen Modul den Funktionsgenerator-Modus.
FG_READ_INDEX	gibt auf dem angegebenen Modul den Positionszeiger eines bestimmten Funktionsgenerators zurück.
FG_STATUS	gibt den Status aller Funktionsgeneratoren auf dem angegebenen Modul zurück.
FG_WRITE	überträgt eine gerade Zahl von Daten eines Felds an eine bestimmte Adresse im internen Zwischenspeicher des angegebenen Moduls.
P2_START_DAC	startet die Wandlung bzw. Ausgabe aller DAC des angegebenen D/A-Moduls.
START_DAC	startet die Wandlung bzw. Ausgabe aller DAC des angegebenen D/A-Moduls.
WRITEDAC	schreibt einen Digitalwert in das Ausgaberegister eines bestimmten DAC des angegebenen Moduls.
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.

gegebenen Moduls.

Analoge Eingänge

P2_ADC	führt eine komplette Messung auf einem ADC des angegebenen Moduls durch. Der Rückgabewert hat 16 Bit Auflösung.
ADC	führt eine komplette Messung auf dem 12 Bit-, 14 Bit- oder 16 Bit-ADC des angegebenen Moduls durch.
ADC16	führt eine komplette Messung auf einem 16 Bit ADC durch. Die Angaben gelten ausschließlich für das Modul Pro-AIn-8/16 Rev. A.
P2_ADC24	führt eine komplette Messung auf einem ADC des angegebenen Moduls durch. Der Rückgabewert hat 24 Bit Auflösung.
P2_ADCF	führt eine komplette Messung auf einem Fast-ADC durch. Der Rückgabewert hat 16 Bit Auflösung.
ADCF	führt eine komplette Messung auf einem Fast-ADC durch.
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_SET_LIMIT	setzt den oberen und unteren Grenzwert für einen F-ADC des angegebenen Moduls.
P2_ADC_READ_LIMIT	liest die Flags für Grenzwertüber- und unterschreitungen auf 16 ADC des angegebenen Moduls aus.
P2_ADC_SET_LIMIT	setzt den oberen und unteren Grenzwert für einen ADC des angegebenen Moduls.
BURST_ABORT	bricht eine laufende Burst-Messreihe auf dem angegebenen Modul ab und gibt die Anzahl der bereits durchgeführten Messungen oder der gespeicherten Messwerte zurück.
BURST_CREAD	kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines bestimmten Kanals in ein Feld. Die Anzahl der zu kopierenden Messwerte muss angegeben werden.
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.
BURST_CSTART	startet eine kontinuierliche Burst-Messreihe (Modus „Continuous“) auf dem angegebenen Modul.
P2_BURST_INIT	legt die Kennwerte für eine Burst-Messreihe auf dem angegebenen Modul fest.
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.
BURST_READ	kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines Kanals in ein bestimmtes Feld.
P2_BURST_READ_INDEX	gibt die Adresse im Modulspeicher zurück, an der zuletzt Messwerte abgelegt wurden.
BURST_READ_PACKED	kopiert die auf dem angegebenen Modul gespeicherten Messwerte eines Kanals in ein bestimmtes Feld, jedoch komprimiert und schnell.
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	P2_BURST_START startet eine Burst-Messreihe auf allen angegebenen Modulen gleichzei-

	tig.
P2_BURST_START	startet eine Burst-Messreihe auf allen angegebenen Modulen gleichzeitig.
BURST_START	startet (unabhängig vom Prozessor) eine Burst-Messreihe auf dem angegebenen Modul.
P2_BURST_STATUS	ermittelt die Anzahl der noch durchzuführenden Burst-Messungen auf dem angegebenen Modul.
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.
READADC	liest das Ergebnis einer Wandlung aus dem ADC-Register des angegebenen Moduls aus.
READADCF	liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus.
READADCF_32	liest die Wandlungsergebnisse aus 2 aufeinander folgenden F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.
READADCF_SCONV	liest das Wandlungsergebnis aus einem F-ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung.
READADCF_SCONV_32	liest die Wandlungsergebnisse aus 2 F-ADC des angegebenen Moduls aus und gibt sie gemeinsam in einem 32 Bit-Wert zurück.
READADC_SCONV	liest das Wandlungsergebnis aus dem ADC des angegebenen Moduls aus und startet sofort eine neue Konvertierung.
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. Das Ergebnis hat 24 Bit Auflösung.
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.
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.
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.
P2_READ_ADCF8	liest die Wandlungsergebnisse aus allen 8 F-ADC des angegebenen Moduls aus.
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.
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.
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_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.
SEQ_MODE	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.
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.
SEQ_READ32	kopiert alle 32 Messwerte (je 16 Bit) von dem angegebenen Modul in ein Ziel-Feld.
SEQ_READ_ONE	liest einen bestimmten Messwert (16 Bit) einer Messgruppe auf dem angegebenen Modul aus.
SEQ_READ_PACKED	
SEQ_READ_PACKED	kopiert eine gerade Anzahl an Messwerten (16 Bit) paarweise von dem angegebenen Modul in ein Ziel-Feld.
SEQ_READ_TWO	liest auf einmal 2 aufeinanderfolgende Messwerte (je 16 Bit) einer Messgruppe auf dem angegebenen Modul aus und gibt diese in einem 32 Bit-Wert zurück.
SEQ_SELECT	legt fest, welche Kanäle zu der Messgruppe gehören, die die Ablaufsteuerung auf dem angegebenen Modul wandeln soll.
SEQ_SET_DELAY	legt die Einschwingzeit (Wartezeit zwischen 2 Messungen) der Ablaufsteuerung auf dem angegebenen Modul fest.
P2_SEQ_START	P2_SEQ_WAIT wartet, bis die Ablaufsteuerung auf dem angegebenen Modul alle Kanäle der Messgruppen gewandelt und gespeichert hat.
SEQ_STATUS	ermittelt, wieviele Kanäle der Messgruppe die Ablaufsteuerung auf dem angegebenen Modul bereits gewandelt und gespeichert hat.
P2_SEQ_WAIT	wartet, bis die Ablaufsteuerung auf dem angegebenen Modul alle Kanäle der Messgruppen gewandelt und gespeichert hat.
P2_SET_AVERAGE_FILTER	legt fest, ob und aus wievielen Messwerten das angegebene Modul einen gleitenden Mittelwert berechnet.
SET_GAIN	setzt für einen Kanal des angegebenen Moduls die Betriebsart fest und damit auch den Verstärkungsfaktor und den Messbereich.
P2_SET_MUX	stellt den Multiplexer des angegebenen Moduls auf einen bestimmten Eingang und eine bestimmte Verstärkung ein.
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.
SE_DIFF	stellt die Betriebsart single ended oder differentiell für alle Analog-Eingänge auf dem angegebenen Modul ein.
SH_SETMODE	stellt den Modus der Sample- und Hold-Stufen auf dem angegebenen Modul ein.
P2_START_CONV	startet die Wandlung auf dem angegebenen Modul.
START_CONV	startet die A/D-Wandlung auf dem angegebenen Modul.
P2_START_CONV_F	startet die Wandlung auf einem oder mehreren F-ADC des angegebenen Moduls.
START_CONV_F	startet die Wandlung auf einem oder mehreren F-ADC des angegebenen Moduls.
SYNC_MODE	bestimmt auf dem angegebenen Modul die Art der Synchronisation mit anderen Modulen, insbesondere für Burst-Messreihen.
P2_WAIT_EOC	wartet, bis die Wandlung auf dem angegebenen Modul abgeschlossen ist.
WAIT_EOC	wartet, bis die zuletzt gestartete A/D-Wandlung abgeschlossen ist.
P2_WAIT_EOC_F	wartet, bis die Wandlung auf allen angegebenen F-ADC des Moduls abgeschlossen ist.
WAIT_EOC_F	wartet, bis die Wandlung auf allen angegebenen F-ADC des Moduls abgeschlossen ist.

Digitale Ein-/Ausgänge

DIGIN_LONG_F	gibt den Zustand der Eingänge (Bits 31...00) des angegebenen Moduls als Bitmuster zurück.
DIGIN_WORD1	gibt den Zustand der Eingänge 0...15 des angegebenen Moduls als Bitmuster zurück.
DIGIN_WORD2	gibt den Zustand der Eingänge 16...31 des angegebenen Moduls als Bitmuster zurück.
DIGOUT	setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.
DIGOUT_BITS_F	setzt die spezifizierten Ausgänge auf dem angegebenen Modul auf den Pegel High oder den

	Pegel Low.
DIGOUT_F	setzt einen einzelnen Ausgang des angegebenen Digital-Moduls auf den Pegel High oder Low. Alle übrigen Ausgänge bleiben unverändert.
DIGOUT_LONG_F	setzt oder löscht durch den übergebenen 32 Bit-Wert alle Ausgänge des angegebenen Moduls.
DIGOUT_WORD1	setzt gleichzeitig die digitalen Ausgänge 0...15 des angegebenen Moduls auf einen bestimmten Pegel.
DIGOUT_WORD2	setzt gleichzeitig die digitalen Ausgänge 16...31 des angegebenen Moduls auf einen bestimmten Pegel.
DIGPROG1	programmiert die digitalen Kanäle 0...15 des angegebenen Moduls als Ein- oder Ausgang.
DIGPROG2	programmiert die Kanäle 16...31 des angegebenen Moduls als Ein- oder Ausgang.
DIG_LATCH	überträgt auf dem angegebenen Modul Digital-Informationen von den Eingängen in die Eingangs-Latches und/oder von den Ausgangs-Latches zu den Ausgängen.
DIG_READLATCH1	liefert die unteren 16 Bit (Bit 0...Bit 15) aus dem Zwischenregister für die digitalen Eingänge des angegebenen Moduls.
DIG_READLATCH2	liefert die oberen 16 Bit (Bit 16...Bit 31) aus dem Zwischenregister für die digitalen Eingänge des angegebenen Moduls.
DIG_WRITELATCH1	schreibt einen Wert in die unteren 16 Bit (Bit 0...15) des Zwischenregisters für die digitalen Ausgänge des angegebenen Moduls.
DIG_WRITELATCH2	schreibt einen Wert in die oberen 16 Bit (Bit 16...31) des Zwischenregisters für die digitalen Ausgänge des angegebenen Moduls.
DIG_WRITELATCH32	schreibt einen 32 Bit-Wert in das Langwort (Bits 31...00) des Latches auf dem angegebenen Modul.
GET_DIGOUT_LONG	gibt den Inhalt des Ausgangs-Latches (Register für digitale Ausgänge) auf dem angegebenen Modul zurück.
GET_DIGOUT_WORD1	gibt das untere Wort (Bit 0...15) des Ausgangs-Latches (Register für digitale Ausgänge) des angegebenen Moduls zurück.
GET_DIGOUT_WORD2	gibt das obere Wort (Bit 16...31) des Ausgangs-Latches (Register für digitale Ausgänge) des angegebenen Moduls zurück.
PWM_ENABLE	gibt alle internen Zähler frei oder stoppt sie. Die Zähler werden über die Nummer des zugehörigen PWM-Ausgangs gewählt.
PWM_OUT	setzt einen bestimmten PWM-Ausgabekanal des angegebenen Moduls auf den Pegel High oder Low.
PWM_SET	setzt die Voreinstellungen eines bestimmten PWM-Ausgabekanals auf dem angegebenen Modul.

Kommunikationsschnittstelle

CAN

CAN_MSG	ist ein eindimensionales Feld bestehend aus 9 Elementen, in dem die Message-Objekte (Nachrichten) des CAN-Busses beim Senden und Empfangen gespeichert sind oder werden.
EN_INTERRUPT	konfiguriert ein Message-Objekt des angegebenen Moduls, dass es bei Eintreffen einer Nachricht einen Event-Signal (Interrupt) erzeugt.
EN_RECEIVE	gibt ein Message-Objekt für den Empfang von Nachrichten auf dem angegebenen Modul frei.
EN_TRANSMIT	gibt ein Message-Objekt für das Senden von Nachrichten auf dem angegebenen Modul frei.
GET_CAN_REG	gibt den Inhalt eines bestimmten Registers auf einem CAN-Controller des angegebenen Moduls zurück.
INIT_CAN	initialisiert einen der CAN-Controller auf dem angegebenen Modul und bringt ihn in einen definierten Anfangszustand.
READ_MSG	gibt zurück, ob eine neue Nachricht in einem Message-Objekt eines der CAN-Controller auf dem angegebenen Modul empfangen wurde.
SET_CAN_BAUDRATE	stellt die angegebene Baudrate auf einem der Controller auf dem angegebenen Modul ein und gibt zurück, ob dies erfolgreich geschehen ist.
SET_CAN_REG	schreibt einen Wert in ein Register des gewählten CAN-Controllers auf dem angegebenen Modul.
TRANSMIT	liest die Daten aus dem Feld CAN_MSG und sendet die Daten als Nachricht.

Feldbus

CHANGED_DATA	überprüft, ob sich auf dem angegebenen Modul seit dem letzten Zugriff der Applikation auf das DP-RAM die Daten im Ausgangsbereich geändert haben.
CHECK_ACCESS	gibt zurück, auf welche Bereiche des DP-RAM im angegebenen Modul die Applikation das Zugriffsrecht hat.
GET_PRO_BYTE	gibt ein Byte aus einer bestimmten Speicheradresse des DP-RAM des Feldbus-Moduls zurück.
GET_READ_BUFFER	kopiert einen definierten Datenblock aus dem Speicherbereich des DP-RAM in das angegebene Zielfeld.
INIT_SLAVE	initialisiert den Feldbus-Slave und ist nur nach einem Einschalten (power up) möglich.
REQUEST_ACCESS	beantragt das Zugriffsrecht auf das DP-RAM des Slaves für die Applikation.
REQUEST_RELEASE_ACCESS	beantragt, das Zugriffsrecht auf das DP-RAM des Slaves an den Feldbus zurückzugeben.
SET_PRO_BYTE	schreibt einen Wert in eine Speicherstelle des DP-RAM des Feldbus-Moduls.
SET_WRITE_BUFFER	kopiert die Daten eines Felds in einen definierten Speicherbereich des DP-RAM.

LS-Bus

LS_DIGPROG	programmiert die digitalen Kanäle 1...32 eines Moduls vom Typ HSM-24V am LS-Bus über eine Schnittstelle des Pro-Moduls in Gruppen zu 8 als Ein- oder Ausgang.
LS_DIG_IO	setzt alle Digital-Ausgänge des Moduls HSM-24V am LS-Bus über eine Schnittstelle des Pro-Moduls auf den Pegel High oder Low und gibt den Zustand aller Kanäle als Bitmuster zurück.
LS_DIO_INIT	initialisiert ein Modul vom Typ HSM-24V am LS-Bus über eine Schnittstelle des Pro-Moduls und gibt den Fehlerstatus zurück.
LS_WATCHDOG_INIT	aktiviert oder deaktiviert den Watchdog-Zähler eines Moduls am LS-Bus über eine Schnittstelle des Pro-Moduls.

RSxxx

CHECK_SHIFT_REG	gibt zurück, ob alle Daten gesendet sind, die in den Sende-FIFO des Kanals (auf dem angegebenen Modul) geschrieben wurden.
GET_RS	liest den Inhalt eines bestimmten Controller-Registers auf dem angegebenen Modul aus.
READ_FIFO	liest einen Wert aus dem Eingangs-FIFO eines bestimmten Kanals auf dem angegebenen Modul.
RS485_SEND	legt die Übertragungsrichtung für einen bestimmten Kanal des angegebenen Moduls fest.
RS_INIT	initialisiert einen bestimmten Kanal auf dem angegebenen Modul.
RS_RESET	führt auf dem angegebenen Modul einen Hardware-Reset durch und löscht dabei die Einstellungen für alle Kanäle.
SET_RS	schreibt einen Wert in ein bestimmtes Register des angegebenen Moduls.
WRITE_FIFO	schreibt einen Wert in den Sende-FIFO eines bestimmten Kanals auf dem angegebenen Modul.

Komparator

COMP_DIGIN_WORD	gibt den aktuellen Status des Schwellenwert-Vergleichs für alle Kanäle auf dem angegebenen Modul zurück.
COMP_DIGIN_WORD_DIFF	gibt den aktuellen Status des Schwellenwert-Vergleichs auf dem angegebenen Modul zurück.
COMP_FIFO_READ	liest die letzten 2 x 1024 Messwerte eines Kanalpaars aus dem internen FIFO-Speicher des angegebenen Moduls und überträgt sie in 2 Felder.
COMP_FIFO_SELECT	legt das Kanalpaar fest, dessen Daten im internen FIFO-Speicher des angegebenen Moduls gespeichert werden.
COMP_READ	gibt den aktuellen, minimalen oder maximalen Messwert eines bestimmten Kanals auf dem angegebenen Modul zurück.
COMP_RESET	setzt auf dem angegebenen Modul die Messung des Minimal- und Maximalwerts für die ausgewählten Kanäle gleichzeitig zurück.
COMP_SET	setzt den unteren und den oberen Schwellenwert für einen bestimmten Kanal des angegebenen Moduls fest.

Speichermedium

<code>MEDIA_RD_BLK_F</code>	kopiert eine Anzahl an Datenblöcken mit Float-Werten aus einer der zehn Dateien auf dem Datenträger des angegebenen Moduls in ein Feld.
<code>MEDIA_RD_BLK_L</code>	kopiert eine Anzahl an Datenblöcken mit Long-Werten aus einer der zehn Dateien auf dem Datenträger des angegebenen Moduls in ein Feld.
<code>MEDIA_RD_FILEINFO</code>	initialisiert die Mediumverwaltung (Glue-Logik) auf dem angegebenen Modul und gibt die Datei-Informationen (Start- und Endsektor) in einem Feld zurück.
<code>MEDIA_WR_BLK_F</code>	kopiert eine Anzahl an Float-Datenblöcken aus einem Feld in eine der zehn Dateien auf dem Datenträger des angegebenen Moduls.
<code>MEDIA_WR_BLK_L</code>	kopiert eine Anzahl an Long-Datenblöcken aus einem Feld in eine der zehn Dateien auf dem Datenträger des angegebenen Moduls.
<code>RTC_GET</code>	gibt das Datum und die Zeit von der Echtzeituhr des angegebenen Moduls zurück.
<code>RTC_SET</code>	setzt das Datum und die Zeit auf der Echtzeituhr des angegebenen Moduls. Ungültige Werte werden nicht akzeptiert.

System

<code>CHECKLED</code>	gibt den Status der LED (oben auf der Frontplatte) auf dem angegebenen Modul zurück.
<code>P2_CHECK_LED</code>	gibt den Status der LED (oben auf der Frontplatte) auf dem angegebenen Modul zurück.
<code>CPU_DIGIN (T11)</code>	Nur Prozessor T11. <code>CPU_DIGIN</code> gibt zurück, ob seit dem letzten Befehlsaufruf eine Flanke an einem DIG I/O-Eingang des Prozessormoduls aufgetreten ist.
<code>CPU_DIGIN (T9, T10)</code>	Nur Prozessoren T9, T10. <code>CPU_DIGIN</code> gibt zurück, ob seit dem letzten Aufruf der Anweisung eine fallende Flanke am Eingang Digin 0 des Prozessormoduls aufgetreten ist.
<code>CPU_DIGOUT</code>	setzt einen DIG I/O-Ausgang des Prozessormoduls auf den angegebenen TTL-Pegel.
<code>CPU_DIG_IO_CONFIG</code>	konfiguriert alle DIG I/O-Kanäle des Prozessormoduls.
<code>CPU_EVENT_CONFIG</code>	konfiguriert den EVENT IN-Kanal des Prozessormoduls.
<code>P2_EVENT2_CONFIG</code>	konfiguriert die Vorverarbeitung der Event-Signale auf dem angegebenen Modul.
<code>EVENTENABLE</code>	sperrt oder aktiviert den externen Event-Eingang auf dem angegebenen Modul.
<code>P2_EVENT_CONFIG</code>	konfiguriert den externen Event-Eingang des angegebenen Moduls.
<code>P2_EVENT_ENABLE</code>	sperrt oder aktiviert den externen Event-Eingang auf dem angegebenen Modul.
<code>P2_EVENT_READ</code>	gibt den aktuellen TTL-Pegel an den Event-Eingängen des angegebenen Moduls zurück.
<code>RESETWATCHDOGTIMER</code>	setzt den Watchdog-Zähler des CPU-Moduls zurück auf den Startwert. Der Zähler bleibt aktiv.
<code>SETLED</code>	schaltet die LED (oben auf der Frontplatte) auf dem angegebenen Modul ein oder aus.
<code>P2_SET_LED</code>	schaltet die LED (oben auf der Frontplatte) auf dem angegebenen Modul ein oder aus.
<code>STARTWATCHDOG</code>	aktiviert den Watchdog-Zähler des CPU-Moduls und setzt ihn auf den Startwert.
<code>STOPWATCHDOG</code>	deaktiviert den Watchdog-Zähler des CPU-Moduls.
<code>SYNCALL</code>	startet eine bestimmte Aktion synchron auf allen Modulen, die mit SyncEnable aktiviert wurden.
<code>SYNCENABLE</code>	aktiviert oder deaktiviert auf dem angegebenen Modul die Synchron-Option.
<code>SYNCSTAT</code>	gibt die Einstellung der Synchron-Option auf dem angegebenen Modul zurück.
<code>P2_SYNC_ALL</code>	startet auf den angegebenen Modulen synchron eine bestimmte Aktion.
<code>P2_SYNC_ENABLE</code>	aktiviert oder deaktiviert die gewählten Ein- oder Ausgänge auf dem angegebenen Modul für die Synchron-Option.
<code>P2_SYNC_STAT</code>	gibt die Einstellung der Synchron-Option auf dem angegebenen Modul zurück.

Temperatur-Eingänge

<code>PT100_DIG_TO_R</code>	berechnet aus dem Digitalwert eines PT100-Fühlers den zugehörigen Widerstand in Ohm.
<code>PT100_DIG_TO_TEMP</code>	berechnet aus dem Digitalwert eines PT100-Fühlers die zugehörige Temperatur in Celsius oder Fahrenheit.
<code>TCJ_DIG_TO_TEMP</code>	berechnet aus einem Digitalwert eines Thermoelements vom Typ J die zugehörige Temperatur in Celsius oder Fahrenheit.
<code>TCK_DIG_TO_TEMP</code>	berechnet aus einem Digitalwert eines Thermoelements vom Typ K die zugehörige Temperatur in Celsius oder Fahrenheit.
<code>TC_READ_B</code>	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf

	dem Modul zurück.
TC_READ_E	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.
TC_READ_J	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.
TC_READ_K	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.
TC_READ_N	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.
TC_READ_R	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.
TC_READ_S	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.
TC_READ_T	gibt die Thermospannung (μV) oder die Temperatur ($^{\circ}\text{C}$ / $^{\circ}\text{F}$) eines bestimmten Kanals auf dem Modul zurück.
TC_SELECT	schaltet den angegebenen Thermoelement-Kanal über Multiplexer auf den analogen Ausgang des Moduls.
TC_SET_RATE	stellt die Abtastrate für das angegebene Modul ein.

Zähler

CNT_CLEAR	setzt den Zählerstand eines oder mehrerer Zähler auf dem angegebenen Modul auf den Wert 0 (Null).
CNT_ENABLE	aktiviert oder deaktiviert einen oder mehrere Zähler auf dem angegebenen Modul.
CNT_LATCH	übernimmt den aktuellen Zählerstand eines oder mehrerer Zähler auf dem angegebenen Modul in die jeweiligen Zwischenregister (= latches).
CNT_READ16	gibt den aktuellen Zählerstand eines 16 Bit-Zählers auf dem angegebenen Modul zurück.
CNT_READ32	gibt den aktuellen Zählerstand eines 32 Bit-Zählers auf dem angegebenen Modul zurück.
CNT_READLATCH16	gibt den Wert aus dem Zwischenregister eines 16 Bit-Zählers auf dem angegebenen Modul zurück.
CNT_READLATCH32	gibt den Wert aus dem Zwischenregister eines 32 Bit-Zählers auf dem angegebenen Modul zurück.
CNT_SETMODE	stellt die Betriebsart aller Zähler auf dem angegebenen Modul ein, Vierfach-Flankenauswertung oder Takt- und Richtungseingang.
CO4_CLEARENABLE	schaltet den externen Eingang CLR eines oder mehrerer Zähler auf dem angegebenen Modul frei.
CO4_GETSTATUS	gibt den Status der Eingangssignale eines Zählers auf dem angegebenen Modul in einem Bitmuster zurück.
CO4_LATCHENABLE	schaltet den externen Eingang LATCH eines oder mehrerer Zähler auf dem angegebenen Modul frei.
CO4_READ	gibt den aktuellen Zählerstand eines Zählers des angegebenen Moduls zurück.
CO4_READLATCH	gibt den Wert aus dem Zwischenregister (Latch) eines Zählers des angegebenen Moduls zurück.
CO4_RESETSTATUS	löscht das Statusregister eines oder mehrerer Zähler auf dem angegebenen Modul.
CO4_SETMODE	stellt den Zählmodus eines Zählers auf dem angegebenen Modul ein.
CO4_SET_LATCHMODE	bestimmt den Modus der Latch-Eingänge für alle Zähler des angegebenen Moduls.
EXTLCH_ENABLE	gibt alle Latch-Eingänge auf dem angegebenen Modul entweder frei oder sperrt sie. Die Latch-Eingänge werden über die Nummer des zugehörigen Zählers ausgewählt.
SSI_MODE	stellt auf dem angegebenen Modul den Modus aller SSI-Decoder ein, entweder „single shot“ (einzelne lesen) und „continuous“ (kontinuierlich lesen).
SSI_READ	gibt den zuletzt gespeicherten Zählerstand eines bestimmten SSI-Zählers auf dem angegebenen Modul zurück.
SSI_SET_BITS	stellt für einen bestimmten SSI-Zähler auf dem angegebenen Modul die Anzahl der zu Bits ein, die einen vollständigen Encoder-Wert bilden.
SSI_SET_CLOCK	stellt die Taktrate (ca. 40kHz bis 1MHz) auf dem angegebenen Modul ein, mit der der Encoder getaktet wird.
SSI_START	startet auf dem angegebenen Modul das Auslesen eines oder beider SSI-Encoder (nur im

SSI_STATUS

Modus single shot).

liefert für einen bestimmten Decoder den aktuellen Lese-Status auf dem angegebenen Modul zurück.

A.4 Abkürzungsverzeichnis

A/D	Analog to Digital	h / Hex	Hexadezimalzahl
ADC	Analog to Digital Converter	I/O	Input / Output
ADSP	Analog Devices Signal Processor	IC	Integrated Circuit
b	Binärzahl	InAmp	Instrumentation Amplifier
CLK	Clock	INL	Integral Non-Linearity
CLR	CleaR	IRQ	Interrupt ReQuest
CMOS	Complementary Metal Oxide Semiconductor	kB	kilo-Byte (= 1024 Byte)
CMRR	Common Mode Rejection Ratio	kByte	siehe kB
D/A	Digital to Analog	LED	Light Emitting Diode
DAC	Digital to Analog Converter	LSB	Least Significant Bit
DIL	Dual InLine	MB	Mega-Byte (= 1024kB)
DIO	Digital Input / Output	MByte	siehe MB
DIR	DIRection	MSB	Most Significant Bit
DMA	Direct Memory Access	MUX	MUltipleXer
DMM	Digital Multi-Meter	OpAmp	Operational Amplifier
DNL	Differential Non-Linearity	PC	Personal Computer
DRAM	Dynamic Random Access Memory	PCB	Printed Circuit Board
DSP	Digital Signal Processor	PGA	Programmable Gain Amplifier
EOC	End Of Conversion	S&H	Sample & Hold
EMV	Elektro-Magnetische Verträglichkeit	SRAM	Static Random Access Memory
ESD	Electro Static Discharge	TCP/IP	Transport Control Protocol / Internet Protocol
FPGA	Field Programmable Gate Array	TTL	Transistor-Transistor Logic
FSR	Full Scale Range	V _{cc}	Voltage collector-collector
GND	GrouND	V _{ee}	Voltage emitter-emitter
		V/R	Vor-/Rückwärts
Hersteller			
AD	Analog Devices	TI	Texas Instruments
BB	Burr-Brown		
LT	Linear Technology		