ADwin-Pro

Systembeschreibung

Programmierung in *ADbasic*

Version 1.4

September 1998

INHALTSVERZEICHNIS

Einführung

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

Die Befehle zum Ansprechen des ***ADwin-Pro***-Systems aus ***ADbasic*** werden in den mitgelieferten INCLUDE-Dateien zur Verfügung gestellt. Um den Zugriff auf die Module[1] des ***ADwin-Pro***-Systems zu ermöglichen, sind - je nach gewünschter Funktion - eine oder mehrere der folgenden Dateien in Ihr ***ADbasic***-Programm einzubinden.

ADWINPRO.INC-	System-Befehle
ADWPAD.INC -	Befehle zum Ansprechen der AD-Wandler-Module
ADWPDA.INC -	Befehle zum Ansprechen der DA-Wandler-Module
ADWPDIO.INC -	Befehle zum Ansprechen der Digital-Module
ADWPEXT.INC -	Befehle zum Ansprechen der Extended-Module

Die Dateien werden mit der ***ADbasic***-Anweisung: #INCLUDE eingebunden. Realisiert werden die Befehle zur Programmierung des ***ADwin-Pro***-Systems durch die Definition von Funktionen und Prozeduren.
Diese Möglichkeit besteht für Compilerversionen von ***ADbasic 2.0*** ab 28.Oktober 1996.

Um Ihnen den Einstieg in die Programmierung des ***ADwin-Pro***-Systems so leicht wie möglich zu machen, liegt jedem System eine Diskette mit der Aufschrift: '***ADwin-Pro***-System, Programmierung in ***ADbasic***' bei. Diese Diskette beinhaltet alle INCLUDE-Dateien, das Test- und Diagnoseprogramm ADPRO, sowie den Source-Code der Programmbeispiele am Ende dieser Dokumentation. Bevor Sie mit dem System arbeiten, sollten Sie das folgende Verzeichnis auf Ihrem Rechner anlegen:

C:\ADBASIC3\INC

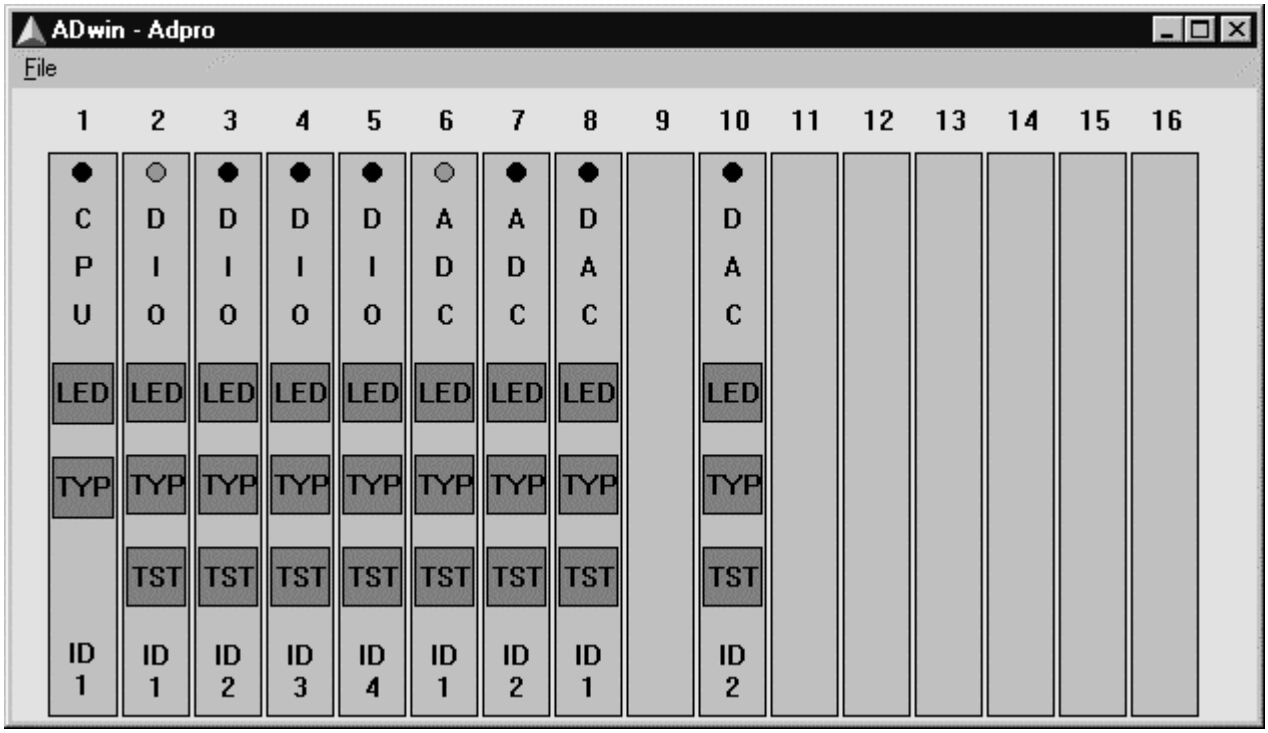
Kopieren Sie anschließend alle Dateien von der mitgelieferten Diskette in dieses Verzeichnis.

1 Anstatt des Ausdrucks „Modul" finden Sie in den Dokumentationen zu ***ADwin-Pro*** auch den Begriff „Karte". Wenn der Begriff „Karte" (z. B. Kartentyp, Kartenadresse) im Zusammenhang mit dem ***ADwin-Pro***-System verwendet wird, dann ist immer ein Modul/eine Karte für das ***ADwin-Pro***-System und keine PC-Einsteckkarte gemeint.
Ausnahme: Die PC-Einsteckkarte ***ADlink*** zum Verbinden des ***ADwin-Pro***-Systems mit dem PC.

Das Testprogramm *ADpro*

Das Test- u. Diagnoseprogramm ADpro.exe zeigt an, mit welchen Modulen Ihr ***ADwin-Pro***-System bestückt ist. Von jedem Modul wird die eingestellte Moduladresse ermittelt (ID). Außerdem kann die Funktion der einzelnen Module überprüft werden.

Bevor das Programm gestartet wird, muß der ***ADwin***-Treiber geladen werden.



Das Programm unterscheidet folgende Modulklassen:

Modulklasse	Beschreibung	Testfunktion (TST)
CPU	Prozessor-Modul	
ADC	Analog/Digital-Converter-Modul	Zeigt die aktuellen Meßwerte aller Eingänge an (Digits)
DAC	Digital/Analog-Converter-Modul	Für jeden Ausgang kann ein Ausgabewert vorgegeben werden (Digits)
DIO	Digital-Input/Output-Modul oder Zählermodul	Zeigt die aktuellen Pegel an den Eingängen an (Dezimal u. binär), dig. Ausgänge können gesetzt bzw. gelöscht werden.
EXT	Extended-Module z. B. Thermo-elementverstärker, Filter, usw.	

Nähere Informationen zu den einzelnen Modulen erscheinen, wenn der Mauszeiger auf das Feld TYP bewegt und anschließend die linke Maustaste betätigt wird.

Über das Feld LED kann die Leuchtdiode des entsprechenden Moduls ein- und ausgeschaltet werden.

Benötigt werden folgende Dateien:

adpro.exe	(Hauptprogramm)
adprot.t50	(***ADbasic***-Prozeß für T450-Prozessor)
adprot.t80	(***ADbasic***-Prozeß für T400- u. T805-Prozessor)
adprot.t90	(***ADbasic***-Prozeß für ADSP21062-Prozessor)

ADWINPRO.INC

Die Include-Datei ADWINPRO.INC beinhaltet allgemeine System-Funktionen und Prozeduren, die zum Ansprechen des ***ADwin-Pro***-Prozessor-Moduls benötigt werden. Wenn Sie diese Datei mit der ***ADbasic***-Anweisung:

```
#INCLUDE ADWINPRO.INC
```

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

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

Anmerkung: Die möglichen Werte für *KARTENTYP* sind in der Datei ADWINPRO.INC definiert und lauten:

dio	= 000H	für alle Digital Input/Output 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

EVENTENABLE

Syntax: *EVENTENABLE(KARTENADRESSE,KARTENTYP,WERT)*

KARTENADRESSE:	eingestellte Kartenadresse
KARTENTYP:	Typ der Karte
WERT:	0= externes Event sperren, 1= externes Event zulassen

Sperrt oder aktiviert den externen EVENT-Eingang zur ***ADbasic***-Programmsteuerung über ein externes Triggersignal.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC

INIT:
EVENTENABLE(1,dio,1)     'Externes Event von der Digital-Karte 1 zulassen

EVENT:
...
```

Hinweis:

Das Echtzeit-Entwicklungstool ***ADbasic*** bietet die Möglichkeit Ihr Programm mit einem externen Triggersignal zu synchronisieren (vgl. ***ADbasic***-Handbuch). Dazu verfügen die meisten Module über einen EVENT-Eingang. Damit ***ADbasic*** auf eine steigende Flanke an diesem Eingang reagieren kann, muß der externe EVENT mittels dieser Funktion zugelassen werden.
Nach dem Einschalten des Systems ist der externe EVENT für alle Module gesperrt.In einem System darf der EVENT-Eingang nicht für mehrere Module gleichzeitig aktiviert werden, d. h. bevor der EVENT-Eingang für ein Modul aktiviert wird, müssen alle anderen EVENT-Eingänge inaktiv geschaltet sein.

CHECKLED

Syntax: *ERGEBNIS = CHECKLED(KARTENADRESSE,KARTENTYP)*

KARTENADRESSE:	eingestellte Kartenadresse
KARTENTYP:	Typ der Karte
ERGEBNIS:	0 = LED ist aus, 1 = LED ist an

Überprüft den Status der LED auf dem Modul mit der übergebenen *KARTENADRESSE*.
‘

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC

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

Siehe auch:

SETLED

SETLED

Syntax: *SETLED(KARTENADRESSE,KARTENTYP,WERT)*

KARTENADRESSE:	eingestellte Kartenadresse
KARTENTYP:	Typ der Karte
WERT:	0/1 = ausschalten/einschalten

Schaltet die LED auf der angegebenen Karte aus oder ein.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC

INIT:
  SETLED(1,cpu,1)       'LED des Prozessor-Moduls 1 einschalten
  SETLED(1,ad,1)        'LED des A/D-Moduls mit Adresse 1 einschalten
  SETLED(1,da,1)        'LED des D/A-Moduls mit Adresse 1 einschalten
  SETLED(1,dio,1)       'LED des Digital-Moduls 1 einschalten


EVENT:
  .
  :
  :


FINISH:
  SETLED(1,cpu,0)       'LED des Prozessor-Moduls 1 ausschalten
  SETLED(1,ad,0)        'LED des A/D-Moduls mit Adresse 1 ausschalten
  SETLED(1,da,0)        'LED des D/A-Moduls mit Adresse 1 ausschalten
  SETLED(1,dio,0)       'LED des Digital-Moduls 1 ausschalten
```

Siehe auch:

CHECKLED

RESETWATCHDOGTIMER

Syntax: *RESETWATCHDOGTIMER()*

Setzt den Watchdog-Timer zurück. Bei aktivem Watchdog muß der Watchdog-Timer mindestens alle 80 ms einmal zurückgesetzt werden, da anderenfalls eine Fehlfunktion erkannt wird und das System stehen bleibt.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC

INIT:
STARTWATCHDOG()          'Watchdog aktivieren

EVENT:
RESETWATCHDOGTIMER()     'Watchdog Timer zurücksetzen
...

FINISH:
STOPWATCHDOG()           'Watchdog deaktivieren
```

Siehe auch:

STARTWATCHDOG
STOPWATCHDOG

Hinweis:

Die Watchdog-Funktion dient zur Überwachung des ***ADwin-Pro***-Systems. Diese Funktion ist nur möglich, wenn in Ihr System das Modul mit der Bezeichnung ***Pro-ADboot*** eingebaut ist. Für nähere Informationen zu diesem Modul lesen Sie bitte die Dokumentation mit dem Titel „***ADwin-Pro*** ; System- und Hardwarebeschreibung“.
Das Aktivieren des Watchdogs initialisiert einen Timer mit einem Wert von ca. 80 ms. Bei aktivem Watchdog wird dieser Timer kontinuierlich dekrementiert. Wenn der Timerstand 0 erreicht wird, nimmt das System an, daß eine Fehlfunktion vorliegt und stoppt alle Programme, wobei sämtliche Module in den Ursprungszustand (Power-On-Status) versetzt werden. Daher sollten Sie dafür sorgen, daß Ihr Programm bei aktiviertem Watchdog mindestens alle 80 ms die Funktion RESETWATCHDOGTIMER aufruft, welche den Timerstand wieder auf ca. 80 ms zurücksetzt.

STARTWATCHDOG

Syntax: *STARTWATCHDOG()*

Aktiviert den WatchDog.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC

INIT:
STARTWATCHDOG()          'Watchdog aktivieren
```

Siehe auch:

STOPWATCHDOG
RESETWATCHDOGTIMER

Hinweis:

Die Watchdog-Funktion dient zur Überwachung des ***ADwin-Pro***-Systems. Diese Funktion ist nur möglich, wenn in Ihr System das Modul mit der Bezeichnung ***Pro-ADboot*** eingebaut ist. Für nähere Informationen zu diesem Modul lesen Sie bitte die Dokumentation mit dem Titel „***ADwin-Pro*** ; System- und Hardwarebeschreibung“.
Das Aktivieren des Watchdogs initialisiert einen Timer mit einem Wert von ca. 80 ms. Bei aktivem Watchdog wird dieser Timer kontinuierlich dekrementiert. Wenn der Timerstand 0 erreicht wird, nimmt das System an, daß eine Fehlfunktion vorliegt und stoppt alle Programme, wobei sämtliche Module in den Ursprungszustand (Power on Status) versetzt werden. Daher sollten Sie dafür sorgen, daß Ihr Programm bei aktiviertem Watchdog mindestens alle 80 ms die Funktion RESETWATCHDOGTIMER aufruft, welche den Timerstand wieder auf ca. 80 ms zurücksetzt.

STOPWATCHDOG

Syntax: *STOPWATCHDOG()*

Deaktiviert den Watchdog.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC
   .
   .
   .

FINISH:
STOPWATCHDOG()           'Watchdog deaktivieren
```

Siehe auch:

STARTWATCHDOG
RESETWATCHDOGTIMER

Hinweis:

Die Watchdog-Funktion dient zur Überwachung des ***ADwin-Pro***-Systems. Diese Funktion ist nur möglich, wenn in Ihr System das Modul mit der Bezeichnung ***Pro-ADboot*** eingebaut ist. Für nähere Informationen zu diesem Modul lesen Sie bitte die Dokumentation mit dem Titel „***ADwin-Pro*** ; System- und Hardwarebeschreibung“.
Das Aktivieren des Watchdogs initialisiert einen Timer mit einem Wert von ca. 80 ms. Bei aktivem Watchdog wird dieser Timer kontinuierlich dekrementiert. Wenn der Timerstand 0 erreicht wird, nimmt das System an, daß eine Fehlfunktion vorliegt und stoppt alle Programme, wobei sämtliche Module in den Ursprungszustand (Power on Status) versetzt werden. Daher sollten Sie dafür sorgen, daß Ihr Programm bei aktiviertem Watchdog mindestens alle 80 ms die Funktion RESETWATCHDOGTIMER aufruft, welche den Timerstand wieder auf ca. 80 ms zurücksetzt.

SYNCALL

Syntax: *SYNCALL()*

Führt auf allen Modulen, bei denen SYNCENABLE aktiviert (=1) ist, synchron die folgenden Aktionen aus:

Modultyp	**Aktion**
Analog-Eingang	Wandlung starten
Analog-Ausgang	Wandlung starten
Digital-Eingang	Aktuellen Zustand der Eingänge in das Eingangs-Zwischenregister übertragen
Digital-Ausgang	Wert aus dem Ausgangs-Zwischenregister auf die dig. Ausgänge schalten
Zähler	Aktuelle Zählerstände in die Zähler-Zwischenregister übertragen

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC
#INCLUDE ADWPAD.INC

DIM i AS INTEGER
DIM DATA_1[1000], DATA_2[1000], DATA_3[1000] AS INTEGER

INIT:
SET_MUX(1,0)             'Multiplexer A/D Modul 1 auf Eingang 1 setzen
SET_MUX(2,0)             'Multiplexer A/D Modul 2 auf Eingang 1 setzen
SET_MUX(3,0)             'Multiplexer A/D Modul 3 auf Eingang 1 setzen
SYNCENABLE(1,ad,1)       'Synchronisation A/D Modul 1 aktivieren
SYNCENABLE(2,ad,1)       'Synchronisation A/D Modul 2 aktivieren
SYNCENABLE(3,ad,1)       'Synchronisation A/D Modul 3 aktivieren
i=1

EVENT:
SYNCALL()                'Wandlung für alle drei A/D Module synchron 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
```

Siehe auch:

SYNCENABLE
SYNCSTAT

SYNCENABLE

Syntax: *SYNCENABLE(KARTENADRESSE,KARTENTYP,WERT)*

KARTENADRESSE:	eingestellte Kartenadresse
KARTENTYP:	Typ der Karte
WERT:	0/1 = deaktivieren/aktivieren

Aktiviert oder deaktiviert die Synchron-Option.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC
#INCLUDE ADWPDIO.INC
#INCLUDE ADWPAD.INC

DIM i AS INTEGER
DIM DATA_1[1000], DATA_2[1000], DATA_3[1000], DATA_4[1000] AS INTEGER

INIT:
SET_MUX(1,0)                'Multiplexer A/D-Modul 1 auf Eingang 1 setzen
SYNCENABLE(1,ad,1)          'Synchronisation A/D-Modul 1 aktivieren
SYNCENABLE(1,dio,1)         'Synchronisation Digital-I/O-Modul 1 aktivieren
SYNCENABLE(2,dio,1)         'Synchronisation Zähler-Modul 2 aktivieren
i=1                         'Index initialisieren

EVENT:
SYNCALL()                   ' - Wandlung A/D-Modul 1 starten
                            ' - Aktuellen Zustand der dig. Eingänge des Digital-
                            '   I/O-Moduls 1 in das Eingangs-Zwischenregister
                            '   übernehmen bzw. Wert aus dem Ausgangs-Zwischen-
                            '   register auf die dig. Ausgänge geben
                            ' - Aktuelle Zählerstände der Zähler von Modul 2 in
                            '   die Zähler-Zwischenregister übernehmen
WAIT_EOC(1)                 'Auf des Ende der Wandlung warten
DATA_1[i]=READADC(1)        'A/D-Wandler-Modul 1 auslesen
DATA_2[i]=DIG_READLATCH1(1)'Eingangs-Zwischenregister dig. Modul 1 auslesen
DATA_3[i]=CNT_READ32(2,1)  'Zähler1-Zwischenregister Modul 2 auslesen
DATA_4[i]=CNT_READ32(2,2)  'Zähler2-Zwischenregister Modul 2 auslesen
IF (i=1000) THEN END        'Nach 1000 Durchläufen Prozess beenden
INC(i)                      'Index erhöhen
```

Siehe auch:

SYNCALL
SYNCSTAT

SYNCSTAT

Syntax: *ERGEBNIS = SYNCSTAT(KARTENADRESSE,KARTENTYP)*

KARTENADRESSE:	eingestellte Kartenadresse
KARTENTYP:	Typ der Karte
ERGEBNIS:	0 / 1 = Synchron-Option deaktiviert / aktiviert

Überprüft die Einstellung der Synchron-Option auf dem Modul mit der übergebenen *KARTENADRESSE*.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC
#INCLUDE ADWPDA.INC

DIM i AS INTEGER
DIM DATA_1[1000], DATA_2[1000] AS INTEGER

INIT:
IF (SYNCSTAT(1,da)=1) THEN  'Falls Synchron-Option deaktiviert ist
  SYNCENABLE(1,da,1)        'Für D/A Modul 1 aktivieren
  SYNCENABLE(2,da,1)        'Für D/A Modul 2 aktivieren
ENDIF
i=1

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

Siehe auch:

SYNCALL
SYNCENABLE

ADWPAD.INC

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

```
#INCLUDE ADWPAD.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.

Anhand der Modulliste ist erkennbar auf welche ***ADwin-Pro***-Module die jeweilige Funktion anwendbar ist.

Anmerkung: In allen Anwendungsbeispielen wird davon ausgegangen, daß auf der A/D-Karte die Adresse 1 eingestellt ist.

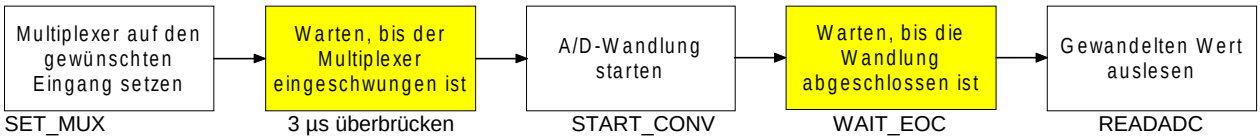
ADC

Syntax: *ERGEBNIS = ADC(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer des analogen Eingangs (1..32)
ERGEBNIS:	16 Bit Integer Wert (0..65535, die vier niederwertigsten Bits sind immer 0)

Führt eine komplette Messung auf einem 12 Bit ADC durch.

Diese Funktion besteht aus einer Sequenz von mehreren Befehlen, die im folgenden Ablaufplan schematisch dargestellt wird.



Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER

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

Siehe auch:

ADC16

Modulliste:

Pro-AIn-8/12
Pro-AIn-32/12
Pro-AO-16/8-12

Hinweis:

Diese Funktion darf nicht in zwei parallel laufenden Prozessen unterschiedlicher Priorität auf das selbe A/D-Modul angewendet werden. Ein niedrig priorisierter Prozeß kann inmitten der ADC-Sequenz von einem hoch priorisierten Prozeß unterbrochen werden. Wenn der hoch priorisierte Prozeß während dieser Unterbrechung den Multiplexer umstellt, liefert die Funktion im niedrig priorisierten Prozeß den Meßwert vom falschen Kanal. Mehrere parallel laufende hoch priorisierte Prozesse können die ADC-Funktion problemlos verwenden, da sich hoch priorisierte Prozesse nicht gegenseitig unterbrechen.

ADCF

Syntax: *ERGEBNIS = ADCF(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer des analogen Eingangs (1..8)
ERGEBNIS:	16 Bit Integer-Wert (0..65535) Die vier niederwertigsten Bits sind bei einer 12-Bit-Karte immer 0.

Führt eine komplette Messung auf einem F-ADC durch.

Diese Funktion besteht aus einer Sequenz von mehreren Befehlen, die im folgenden Ablaufplan schematisch dargestellt wird.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER

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

Modulliste:

Pro-AIn-F-4/12
Pro-AIn-F-8/12
Pro-AIn-F-4/16
Pro-AIn-F-8/16

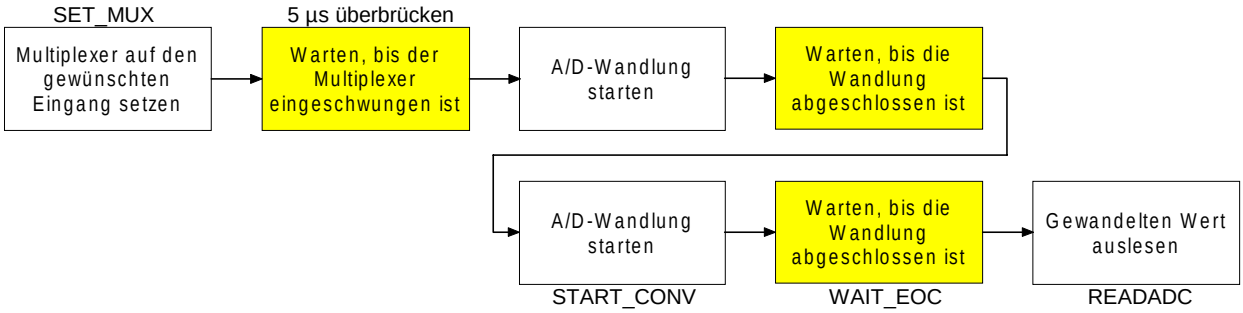
ADC16

Syntax: *ERGEBNIS = ADC16(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer des analogen Eingangs
ERGEBNIS:	16 Bit Integer-Wert (0..65535)

Führt eine komplette Messung auf einem 16-Bit ADC durch.

Diese Funktion besteht aus einer Sequenz von mehreren Befehlen, die im folgenden Ablaufplan schematisch dargestellt wird



Das Starten der A/D-Wandlung auf einem 16-Bit-Wandler, löst parallel zwei 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 an das System übertragen und steht nach dem Ende der Konvertierung zur Verfügung.

Damit die ADC16-Funktion nicht den Wert der letzten Messung, sondern den aktuellen Wert liefert, muß die Wandlung zweimal gestartet werden.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER

EVENT:
  wert = ADC16(1, 7)    'Mißt einen Wert vom analogen Eingang 7
```

Siehe auch:

ADC

Modulliste:

Pro-AIn-8/16

Hinweis:

Diese Funktion darf nicht von zwei parallel laufenden Prozessen unterschiedlicher Priorität auf das selbe A/D-Modul angewendet werden. Ein niedrig priorisierter Prozeß kann inmitten der ADC-Sequenz von einem hoch priorisierte Prozeß unterbrochen werden. Wenn der hoch priorisierte Prozeß während dieser Unterbrechung den Multiplexer umstellt, liefert die Funktion im niedrig priorisierten Prozeß den Meßwert vom falschen Kanal. Mehrere parallel laufende hochpriorisierte Prozesse können die ADC16-Funktion problemlos verwenden, da sich hoch priorisierte Prozesse nicht gegenseitig unterbrechen.

READADC

Syntax: *ERGEBNIS = READADC(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse
ERGEBNIS: 16 Bit Integer-Wert

Liest einen Wert vom A/D-Wandler.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert1 AS INTEGER     'Deklaration

EVENT:
SET_MUX(1,0)             'Multiplexer auf Eingang 1 setzen
3µs (12-Bit-ADC) bzw. 5µs (16-Bit-ADC) auf das Einschwingen des Multiplexers warten*
START_CONV(1)            'Start AD-Wandlung
WAIT_EOC(1)              'Warten auf Wandlung-Ende
wert1 = READADC(1)       'Wert vom ADC einlesen
```

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

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

```
time = READ_TIMER()            'Start AD-Wandlung
DO
UNTIL (time-READ_TIMER()>4)    '5 µs warten
```

Siehe auch:

ADC

Modulliste:

Pro-AIn-8/12
Pro-AIn-32/12
Pro-AIn-8/16
Pro-AO-16/8-12

Hinweis:

Bei einem 16-Bit-A/D-Modul wird nicht der aktuelle, sondern der Meßwert aus der vorhergehenden Messung ausgelesen (vgl. ADC16-Befehl).

READADCF

Syntax: *ERGEBNIS = READADCF(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer des ADCs, der ausgelesen werden soll
ERGEBNIS:	16 Bit Integer-Wert (0..65535) die vier niederwertigsten Bits sind bei einer 12-Bit-Karte immer 0.

Liest einen Wert vom F-A/D-Wandler.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert1 AS INTEGER     'Deklaration

EVENT:
START_CONVF(1,1)         'Start AD-Wandlung
WAIT_EOCF(1,1)           'Warten auf Wandlung-Ende
wert1 = READADCF(1,1)    'Wert vom ADC einlesen
```

Modulliste:

Pro-AIn-F-4/12
Pro-AIn-F-8/12
Pro-AIn-F-4/16
Pro-AIn-F-8/16

READADCF_32

Syntax: *ERGEBNIS = READADCF_32(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer des ersten ADCs, der ausgelesen werden soll. Es muß immer ein Wandler mit einer ungeraden Nummer sein.
ERGEBNIS:	32 Bit Integer-Wert Der Rückgabewert enthält die Meßdaten von zwei aufeinanderfolgenden A/D-Wandlern. Der ADC mit der höheren Nummer wird in die höherwertigen Bits geschrieben. Die vier niederwertigsten Bits eines jeden Wortes sind bei einer 12-Bit-Karte immer 0.

Liest zwei Werte von einer F-Karte.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert1 AS INTEGER          'Deklaration

EVENT:
START_CONVF(1,3)              'Start AD-Wandlung auf ADC1 und ADC2
WAIT_EOCF(1,3)                'Warten auf das Ende der Wandlungen
wert1 = READADCF_32(1,1)      'Wert von ADC1 und ADC2 einlesen
```

Modulliste:

Pro-AIn-F-4/12
Pro-AIn-F-8/12
Pro-AIn-F-4/16
Pro-AIn-F-8/16

READADC_SCONV

Syntax: *ERGEBNIS = READADC_SCONV(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse
ERGEBNIS: 16 Bit Integer-Wert (0..65535)

Liest das Ergebnis einer Konvertierung auf einen ADC aus und startet eine neue Konvertierung.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM i, dummy AS INTEGER
DIM DATA_1[1000] AS INTEGER   'Deklaration

INIT:
i=1
SET_MUX(1,2)                  'Multiplexer auf Eingang 3 setzen
3µs (12-Bit ADC) bzw. 5µs (16-Bit ADC) auf das Einschwingen des Multiplexers warten*
dummy = READADC_SCONV(1)      'A/D-Wandler auslesen und starten
WAIT_EOC(1)

EVENT:
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
```

Siehe auch:

ADC

Modulliste:

Pro-AIn-8/16
Pro-AO-16/8-12

READADCF_SCONV

Syntax: *ERGEBNIS = READADCF_SCONV(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer des ADCs, der ausgelesen werden soll
ERGEBNIS:	16 Bit Integer-Wert (0..65535) die vier niederwertigsten Bits sind bei einer 12-Bit-Karte immer 0.

Liest das Ergebnis einer Konvertierung auf einem F-ADC aus und startet eine neue Konvertierung.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM i, dummy AS INTEGER
DIM DATA_1[1000] AS INTEGER         'Deklaration

INIT:
i=1
dummy = READADCF_SCONV(1,1)         'A/D-Wandler auslesen und starten

EVENT:
WAIT_EOCF(1,1)
DATA_1[i] = READADCF_SCONV(1,1)     'A/D-Wandler auslesen und starten
INC(i)                              'Index erhöhen
IF (i=1001) THEN END                'Nach 1000 Meßwerten Prozess beenden
```

Siehe auch:

ADCF

Modulliste:

Pro-AIn-F-4/12
Pro-AIn-F-8/12
Pro-AIn-F-4/16
Pro-AIn-F-8/16

READADCF_SCONV_32

Syntax: *ERGEBNIS = READADCF_SCONV_32(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer des ersten ADCs, der ausgelesen werden soll. Es muß immer ein Wandler mit einer ungeraden Nummer sein.
ERGEBNIS:	32 Bit Integer-Wert Der Rückgabewert enthält die Meßdaten von zwei aufeinanderfolgenden A/D-Wandlern. Der ADC mit der höheren Nummer wird in die höherwertigen Bits geschrieben. Die vier niederwertigsten Bits eines jeden Wortes sind bei einer 12-Bit-Karte immer 0.

Liest zwei Werte von einer F-Karte aus und startet die Konvertierungen neu.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER          'Deklaration

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


EVENT:
WAIT_EOCF(1,3)               'Warten auf das Ende der Konvertierungen
wert = READADCF_32(1,1)      'Wert vom ADC1 und ADC2 einlesen
```

Modulliste:

Pro-AIn-F-4/12
Pro-AIn-F-8/12
Pro-AIn-F-4/16
Pro-AIn-F-8/16

SET_MUX

Syntax: *SET_MUX(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: Bitmuster (s. Tabelle)

Setzen des Multiplexers und der Verstärkung.

Da die Multiplexer max. 32 Eingänge haben, werden 5 Bits verwendet. Die Verstärker haben vier Einstellungsmöglichkeiten (1-, 2-, 4-, 8-fache Verstärkung), für die weitere 2 Bits benötigt werden. Die Bits werden gesetzt, indem Sie entweder direkt die binäre Zahl angeben oder sie vorher in HEX- oder DEZ-Code umrechnen. Für HEX- und BIN-Code kennzeichnen Sie die Zahlen mit dem entsprechenden Buchstaben (H für HEX, B für BIN). Die für die Einstellungen maßgeblichen Bitkombinationen ersehen Sie aus der folgenden Tabelle.

Bit	6	5	4	3	2	1	0
Funktion	Verstärkung		Multiplexer				
	1 = 00 2 = 01 4 = 10 8 = 11 gilt nicht für 16-Bit-Module und Module mit Fast-ADC		Eingang 1: 00000 Eingang 2: 00001 Eingang 3: 00010 Eingang 4: 00011 Eingang 5: 00100 : Eingang 31: 11110 Eingang 32: 11111				

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert1 AS INTEGER     'Deklaration

EVENT:
SET_MUX(1,0)             'Multiplexer auf Eingang 1, Verstärkung 1 setzen
3µs (12-Bit ADC) bzw. 5µs (16-Bit ADC) auf das Einschwingen des Multiplexers warten
START_CONV(1)            'Start AD-Wandlung
WAIT_EOC(1)              'Warten auf Wandlung-Ende
wert1 = READADC(1)       'Wert vom ADC einlesen
```

Modulliste:

Pro-AIn-8/12
Pro-AIn-8/16
Pro-AIn-32/12
Pro-AO-16/8-12

Hinweis:

Bitte achten Sie darauf, daß der Multiplexer etwa 3 µs (12 Bit) bzw. 5 µs (16 Bit) benötigt, bis er eingeschwungen ist. Zwischen der Neueinstellung des Multiplexers und dem Konvertierungsbeginn müssen Sie deshalb eine Wartezeit von mindestens 3 µs bzw. 5 µs einhalten oder eine/mehrere Anweisung/en einfügen, deren Bearbeitungszeit 3 µs bzw. 5 µs nicht unterschreitet.

SE_DIFF

Syntax: *SE_DIFF(KARTENADRESSE,WAHL)*

KARTENADRESSE:	eingestellte Kartenadresse
WAHL:	0 = single ended (SE) ; 1 = differentiell (DIFF)

Umschalten zwischen single ended und differentiellem Betrieb.

Die analogen Eingänge des ***Pro-AIn-32/12***-Moduls können wahlweise als single ended (32 Eingänge) oder als differentielle Eingänge (16 Eingänge) betrieben werden. Nach dem Einschalten des Systems befinden sich alle Eingänge im differentiellen Modus. Falls die Eingänge single ended Signale verarbeiten sollen, muß diese Prozedur mit dem Wert 0 für *Wahl* aufgerufen werden.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC

EVENT:
SE_DIFF(1,0)              'Karte mit der Nummer 1 wird auf SE gesetzt
SE_DIFF(2,1)              'Karte mit der Nummer 2 wird auf DIFF gesetzt
```

Modulliste:

Pro-AIn-32/12

Hinweis:

Im differentiellen Betrieb werden die analogen Eingänge mit den Nummern 1-8 und 17-24 angesprochen. (vgl. ***ADwin-Pro***-Hardwarebeschreibung)

SH_SETMODE

Syntax: *SH_SETMODE(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: 0 = sample ; 1 = hold

Umschalten zwischen Sample- und Hold-Modus.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC

EVENT:
SH_SETMODE(1,0)          'Karte mit der Nummer 1 wird auf Sample-Modus gestellt
SH_SETMODE(2,1)          'Karte mit der Nummer 2 wird auf Hold-Modus gestellt
```

Modulliste:

Pro-LPSH-8-FI
Pro-LPSH-4-FI
Pro-LPSH-8

START_CONV

Syntax: *START_CONV(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse

Startet die A/D-Wandlung auf dem Modul mit der Adresse *KARTENADRESSE*.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert1 AS INTEGER     'Deklaration

EVENT:
SET_MUX(1,0)             'Multiplexer auf Eingang 1 setzen
3µs (12-Bit ADC) bzw. 5µs (16-Bit ADC) auf das Einschwingen des Multiplexers warten*
START_CONV(1)            'Start AD-Wandlung
WAIT_EOC(1)              'Warten auf Wandlung-Ende
wert1 = READADC(1)       'Wert vom ADC einlesen
```

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

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

```
time = READ_TIMER()            'Aktuellen Timerstand ermitteln
DO
UNTIL (time-READ_TIMER()>4)    '5 µs warten
```

Siehe auch:

WAIT_EOC
READADC

Modulliste:

Pro-AIn-8/12
Pro-AIn-32/12
Pro-AIn-8/16
Pro-AO-16/8-12

START_CONVF

Syntax: *START_CONVF(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer der ADCs, deren Konvertierung gestartet werden soll. Die Angabe der ADCs erfolgt bitweise, so daß die Konvertierung von mehreren Wandlern gleichzeitig gestartet werden kann. Zum Starten von A/D-Wandler 1 und 3 muß die 5 übergeben werden. (Bitweise: 00000101b)

Startet die A/D-Wandlung auf einem F-Modul von einem oder mehreren Wandlern.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER      'Deklaration

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

Siehe auch:

ADCF
READADCF

Modulliste:

Pro-AIn-F-4/12
Pro-AIn-F-8/12
Pro-AIn-F-4/16
Pro-AIn-F-8/16

WAIT_EOC

Syntax: *WAIT_EOC(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse

Warten bis die zuletzt gestartete A/D-Wandlung abgeschlossen ist.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert1 AS INTEGER
                        Deklaration

INIT:
SET_MUX(1,0)            'Multiplexer auf Eingang 1 setzen
3µs (12-Bit ADC) bzw. 5µs (16-Bit ADC) auf das Einschwingen des Multiplexers warten*

EVENT:
START_CONV(1)           'Start AD-Wandlung
WAIT_EOC(1)             'Warten auf Wandlung-Ende
wert1 = READADC(1)      'Wert vom ADC einlesen
```

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

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

```
time = READ_TIMER()             'Start AD-Wandlung
DO
UNTIL (time-READ_TIMER()>4)     '5 µs warten
```

Siehe auch:

START_CONV
READADC

Modulliste:

Pro-AIn-8/12
Pro-AIn-32/12
Pro-AIn-8/16
Pro-AO-16/8-12

WAIT_EOCF

Syntax: *WAIT_EOCF(KARTENADRESSE,ADC_NR)*

KARTENADRESSE:	eingestellte Kartenadresse
ADC_NR:	Nummer der ADCs, auf deren Konvertierungsende gewartet werden soll. Die Angabe der ADCs erfolgt bitweise, so daß das Konvertierungsende von mehreren Wandlern gleichzeitig überwacht werden kann. Zum Überwachen von A/D-Wandler 1 und 3 muß die 5 übergeben werden. (Bitweise: 00000101b)

Warten bis die A/D-Wandlungen von allen angegebenen Wandlern abgeschlossen ist.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER      'Deklaration

EVENT:
START_CONVF(1,1)         'Start AD-Wandlung
WAIT_EOCF(1,1)           'Warten auf das Ende der Konvertierungen
wert = READADCF(1,1)     'Wert vom ADC einlesen
```

Siehe auch:

START_CONVF
READADCF

Modulliste:

Pro-AIn-F-4/12
Pro-AIn-F-8/12
Pro-AIn-F-4/16
Pro-AIn-F-8/16

ADWPDA.INC

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

```
#INCLUDE ADWPDA.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.

Anhand der Modulliste ist erkennbar auf welche ***ADwin-Pro***-Module die jeweilige Funktion anwendbar ist.

Anmerkung: In allen Anwendungsbeispielen wird davon ausgegangen, daß auf der D/A-Karte die Adresse 1 eingestellt ist.

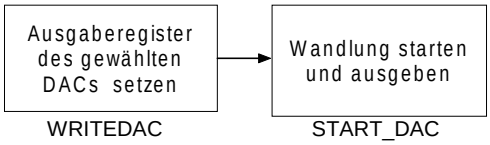
DAC

Syntax: *DAC(KARTENADRESSE,DAC_NR,WERT)*

KARTENADRESSE:	eingestellte Kartenadresse
DAC_NR:	Nummer des Ausgangs
WERT:	Auszugebender Wert (0..65535)

Ausgabe eines Wertes auf einem DAC.

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



Anwendungsbeispiel:

```
REM Digitaler P-Regler
#INCLUDE ADWPAD.INC
#INCLUDE ADWPDA.INC
DIM sw, aw AS INTEGER    'Deklaration
DIM v, stell AS INTEGER  'Deklaration


EVENT:
sw = PAR_1               'Sollwert
v = PAR_2                'Verstärkung
aw = sw - ADC(1,1)       'Regelabweichung berechnen
stell = aw * v           'Stellgröße berechnen
DAC(1,1,stell)           'Ausgabe der Stellgröße
```

WRITEDAC

Syntax: *WRITEDAC(KARTENADRESSE,DAC_NR,WERT)*

KARTENADRESSE:	eingestellte Kartenadresse
DAC_NR:	Nummer des Ausgangs
WERT:	Auszugebender Wert (0..4095 bei 12-Bit-DAC, 0..65535 bei 16-Bit-DAC)

Schreibt den übergebenen *Wert* in das Ausgaberegister des DACs mit der Nummer *DAC_NR*. Der Wert wird erst durch den Aufruf der Prozedur START_DAC ausgegeben.

Anwendungsbeispiel:

```
REM Simultane Ausgabe von vier verschiedenen Signalverlaeufen
REM auf den Ausgaengen 1, 2, 3 und 4 eines D/A-Moduls
REM Die Signalverläufe sind in vier DATAs abgelegt und können
REM vor dem Programmstart vom PC uebergeben werden

#INCLUDE ADWPDA.INC
DIM i AS INTEGER         'Deklaration
DIM DATA_1[1000], DATA_2[1000], DATA_3[1000], DATA_4[1000] AS INTEGER

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 DACs starten
INC(i)
IF (i>1000) THEN i=1
```

Siehe auch:

START_DAC
DAC

START_DAC

Syntax: *START_DAC(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse

Startet die Wandlung bzw. Ausgabe aller DACs des D/A-Moduls mit der übergebenen *Kartenadresse.*

Anwendungsbeispiel:

```
REM Simultane Ausgabe von zwei verschiedenen Signalverlaeufen
REM auf den Ausgaengen 1 und 2 eines D/A-Moduls
#INCLUDE ADWPDA.INC
DIM i INTEGER           '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 DACs starten
INC(i)
IF (i=65535) THEN i=0
```

Siehe auch:

WRITEDAC
DAC

ADWPDIO.INC

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

```
#INCLUDE ADWPDIO.INC
```

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

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

Anhand der Modulliste ist erkennbar auf welche ***ADwin-Pro***-Module die jeweilige Funktion anwendbar ist.

Anmerkung: In allen Anwendungsbeispielen wird davon ausgegangen, daß auf der Digital-Karte die Adresse 1 eingestellt ist

CNT_CLEAR

Syntax: *CNT_CLEAR(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: Bitmuster (1=Zähler zurücksetzen))

Setzt den Zählerstand der/des spezifizierten Zähler/s auf den Wert 0.

Anwendungsbeispiel: **! In dieser Form nur für *Pro-CNT-VR4*-Modul gültig !**

```
#INCLUDE ADWPDIO.INC

INIT:
CNT_SETMODE(1,01111B)              'Zähler 1-4 auf Vierfachflankenauswertung
CNT_CLEAR(1,01111B)                'Zählerstand der Zähler 1-4 auf 0 setzen
CNT_ENABLE(1,01111B)               'Zähler 1-4 aktivieren
```

Hinweis für *Pro-CNT-VR4*-Module:
Bei ***Pro-CNT-VR4***-Modulen ist jedes Bit einem Zähler zugeordnet:
Bit 1 für Zähler 1, Bit 2 für Zähler 2, Bit 3 für Zähler 3, Bit 4 für Zähler 4.

Hinweis für *Pro-CNT-16/16*- und *Pro-CNT-8/32*-Module:
Bei ***Pro-CNT-16/16***-Modulen ist ein Bit vier Zählern zugeordnet; d. h. mit einem Bit werden vier Zähler gleichzeitig zurückgesetzt:
Bit 1 für die Zähler 1, 5, 9 und 13
Bit 2 für die Zähler 2, 6, 10 und 14
Bit 3 für die Zähler 3, 7, 11 und 15
Bit 4 für die Zähler 4, 8, 12 und 16

Bei ***Pro-CNT-8/32***-Modulen ist ein Bit zwei Zählern zugeordnet; d. h. mit einem Bit werden zwei Zähler gleichzeitig zurückgesetzt:
Bit 1 für die Zähler 1 und 5
Bit 2 für die Zähler 2 und 6
Bit 3 für die Zähler 3 und 7
Bit 4 für die Zähler 4 und 8

Modulliste:

Pro-CNT-16/16
Pro-CNT-8/32
Pro-CNT-VR4

CNT_ENABLE

Syntax: *CNT_ENABLE(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: Bitmuster (0=inaktiv, 1=aktiv)

Aktiviert oder deaktiviert die Zähler. Eine 0 deaktiviert den/die entsprechenden Zähler, eine 1 aktiviert den/die Zähler.

Anwendungsbeispiel: **! In dieser Form nur für *Pro-CNT-VR4*-Modul gültig !**

```
#INCLUDE ADWPDIO.INC

INIT:
CNT_SETMODE(1,01000B)          'Zähler 4 auf Vierfachflankenauswertung, alle
                               'anderen Zähler auf Takt/Richtungsauswertung
CNT_CLEAR(1,01000B)            'Zählerstand des Zählers 4 auf 0 setzen
CNT_ENABLE(1,01000B)           'Zähler 4 aktivieren, alle anderen deaktivieren
```

Hinweis für *Pro-CNT-VR4*-Module:

Bei ***Pro-CNT-VR4***-Modulen ist jedes Bit einem Zähler zugeordnet:
Bit 1 für Zähler 1, Bit 2 für Zähler 2, Bit 3 für Zähler 3, Bit 4 für Zähler 4.

Hinweis für *Pro-CNT-16/16*- und *Pro-CNT-8/32*-Module:

Bei ***Pro-CNT-16/16***-Modulen ist ein Bit vier Zählern zugeordnet; d. h. mit einem Bit werden vier Zähler gleichzeitig aktiviert/deaktiviert:
Bit 1 für die Zähler 1, 5, 9 und 13
Bit 2 für die Zähler 2, 6, 10 und 14
Bit 3 für die Zähler 3, 7, 11 und 15
Bit 4 für die Zähler 4, 8, 12 und 16

Bei ***Pro-CNT-8/32***-Modulen ist ein Bit zwei Zählern zugeordnet; d. h. mit einem Bit werden zwei Zähler gleichzeitig aktiviert/deaktiviert:
Bit 1 für die Zähler 1 und 5
Bit 2 für die Zähler 2 und 6
Bit 3 für die Zähler 3 und 7
Bit 4 für die Zähler 4 und 8

Modulliste:

Pro-CNT-16/16
Pro-CNT-8/32
Pro-CNT-VR4

CNT_LATCH

Syntax: *CNT_LATCH(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: Bitmuster (1=Zählerstand in das Latch übernehmen)

Übernimmt den aktuellen Zählerstand der Zähler bei denen das entsprechende Bit im Wert *WAHL* auf 1 steht in die Zwischenregister.

Anwendungsbeispiel: **! In dieser Form nur für *Pro-CNT-VR4*-Modul gültig !**

```
#INCLUDE ADWPDIO.INC
DIM wert AS INTEGER

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

Hinweis für *Pro-CNT-VR4*-Module:

Bei ***Pro-CNT-VR4***-Modulen ist jedes Bit einem Zähler zugeordnet:
Bit 1 für Zähler 1, Bit 2 für Zähler 2, Bit 3 für Zähler 3, Bit 4 für Zähler 4.

Hinweis für *Pro-CNT-16/16*- und *Pro-CNT-8/32*-Module:

Bei ***Pro-CNT-16/16***-Modulen ist ein Bit vier Zählern zugeordnet; d. h. mit einem Bit werden vier Zähler gleichzeitig zurückgesetzt:
Bit 1 für die Zähler 1, 5, 9 und 13
Bit 2 für die Zähler 2, 6, 10 und 14
Bit 3 für die Zähler 3, 7, 11 und 15
Bit 4 für die Zähler 4, 8, 12 und 16

Bei ***Pro-CNT-8/32***-Modulen ist ein Bit zwei Zählern zugeordnet; d. h. mit einem Bit werden zwei Zähler gleichzeitig zurückgesetzt:
Bit 1 für die Zähler 1 und 5
Bit 2 für die Zähler 2 und 6
Bit 3 für die Zähler 3 und 7
Bit 4 für die Zähler 4 und 8

Siehe auch:

CNT_READLATCH16
CNT_READLATCH32

Modulliste:

Pro-CNT-16/16
Pro-CNT-8/32
Pro-CNT-VR4

CNT_SETMODE

Syntax: *CNT_SETMODE(KARTENADRESSE,WAHL)*

KARTENADRESSE:	eingestellte Kartenadresse
WAHL:	Bitmuster (0=Vierfachflankenauswertung, 1=Takt- u. Richtungseingang)

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

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC

INIT:
CNT_SETMODE(1,12)       'Zähler 3 und 4 auf Takt-/Richtungsauswertung, alle
                        'anderen Zähler auf Vierfachflankenauswertung
CNT_CLEAR(1,12)         'Zählerstand der Zähler 3 und 4 auf 0 setzen
CNT_ENABLE(1,12)        'Zähler 3 und 4 aktivieren, alle anderen deaktivieren
```

Modulliste:

Pro-CNT-VR4

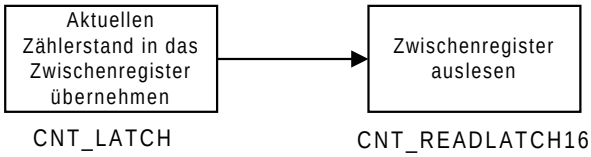
CNT_READ16

Syntax: *ERGEBNIS=CNT_READ16(KARTENADRESSE,NUMMER)*

KARTENADRESSE:	eingestellte Kartenadresse
NUMMER:	Zählernummer
ERGEBNIS:	Aktueller Zählerstand (16-Bit Integer-Wert)

Ermittelt den aktuellen Zählerstand des spezifizierten 16-Bit-Zählers.

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



Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.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
```

Siehe auch:

CNT_READ32

Modulliste:

Pro-CNT-16/16

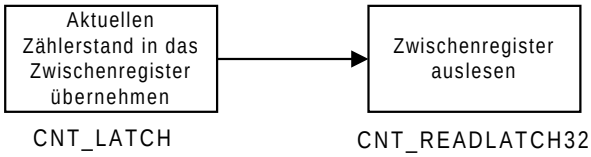
CNT_READ32

Syntax: *ERGEBNIS = CNT_READ32(KARTENADRESSE,NUMMER)*

KARTENADRESSE:	eingestellte Kartenadresse
NUMMER:	Zählernummer
ERGEBNIS:	Aktueller Zählerstand (32-Bit Integer-Wert)

Ermittelt den aktuellen Zählerstand des spezifizierten 32-Bit-Zählers.

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



Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.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
```

Siehe auch:

CNT_READ16

Modulliste:

Pro-CNT-8/32
Pro-CNT-VR4

CNT_READLATCH16

Syntax: *ERGEBNIS = CNT_READLATCH16(KARTENADRESSE,NUMMER)*

KARTENADRESSE:	eingestellte Kartenadresse
NUMMER:	Zählernummer
ERGEBNIS:	16 Bit Integer-Wert

Liefert den Wert aus dem Zwischenregister eines 16-Bit-Zählers. Um den aktuellen Zählerstand zu erhalten, muß dieser vor dem Aufruf der Funktion: CNT_READLATCH16 mit der Prozedur CNT_LATCH in das Zwischenregister übernommen werden.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
DIM wert AS INTEGER

INIT:
wert = ADC(1,1)                                    'Meßwert aufnehmen
IF (wert>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
```

Siehe auch:

CNT_LATCH

CNT_READLATCH32

Syntax: *ERGEBNIS = CNT_READLATCH32(KARTENADRESSE)*

KARTENADRESSE:	eingestellte Kartenadresse
NUMMER:	Zählernummer
ERGEBNIS:	32 Bit Integer-Wert

Liefert den Wert aus dem Zwischenregister eines 32-Bit-Zählers. Um den aktuellen Zählerstand zu erhalten, muß dieser vor dem Aufruf der Funktion: CNT_READLATCH32 mit der Prozedur CNT_LATCH in das Zwischenregister übernommen werden.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
DIM wert AS INTEGER

INIT:
wert = ADC(1,1)                                   'Meßwert aufnehmen
IF (wert>49151) THEN CNT_LATCH(1,6)               'Zähler 2 u. 3 latchen
PAR_1 = CNT_READLATCH32(1,1) - CNT_READLATCH32(1,2)  'Differenz Zähler 2 u. 3
```

Siehe auch:

CNT_LATCH

DIGOUT

Syntax: *DIGOUT(KARTENADRESSE,AUSGANG,WERT)*

KARTENADRESSE:	eingestellte Kartenadresse
AUSGANG:	Nummer des Ausgangs der angesprochen werden soll (0 bis 31)
WERT:	Neuer Zustand für den gewählten Ausgang (0/1 = low/high)

Einen einzelnen Ausgang der Digital-Karte setzen oder rücksetzen. Alle übrigen Ausgänge bleiben unverändert.

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

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER

EVENT:
wert = ADC(1,1)          'Meßwerterfassung
IF (wert < 100) THEN     'Anfang Kontrollstruktur
  DIGOUT(1,2,0)          'Dig. Ausgang 2 zurücksetzen
ENDIF                    'Ende Kontrollstruktur
```

Siehe auch:

DIGOUT_WORD1 (benötigt weniger Ausführungszeit als DIGOUT, außerdem können mehrere Kanäle gleichzeitig umgesetzt werden)

Modulliste:

Pro-DIO-32
Pro-REL-16
Pro-TRA-16

Hinweis:

Diese Prozedur darf nicht in zwei parallel laufenden Prozessen unterschiedlicher Priorität auf das selbe Modul angewendet werden. Ein niedrig priorisierter Prozeß kann inmitten der CLEAR_DIGOUT Sequenz von einem hoch priorisierte Prozeß unterbrochen werden. Wenn der hoch priorisierte Prozeß während dieser Unterbrechung die Stellung der digitalen Ausgänge ändert, geht diese Änderung beim Rückschreiben der manipulierten Variable verloren. Mehrere parallel laufende hochpriorisierte Prozesse können die DIGOUT Prozedur problemlos verwenden, da sich hoch priorisierte Prozesse nicht gegenseitig unterbrechen.

DIGIN_WORD1

Syntax: *ERGEBNIS = DIGIN_WORD1(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse
ERGEBNIS: 16 Bit Integer-Wert

Liefert den Pegel an den digitalen Eingänge 0-15, zusammengefaßt in einem 16-Bit-Wert.

Die Funktion gibt einen 16-Bit-Wert zurück. Jedem digitalen Eingang ist ein Bit des Rückgabewertes zugeordnet. Wenn am Eingang ein High-Pegel anliegt, dann wird das zugehörige Bit auf 1 gesetzt.

Berechnungsbeispiel:
Die Funktion *DIGIN_WORD1()* liefert den Dezimalwert 14. Anhand des Bitmusters von dezimal 14 können Sie erkennen, welche digitalen Eingänge gesetzt sind. In diesem Beispiel sind die Eingänge 1, 2 und 3 gesetzt, da die Summe der zugeordneten Bitwerte den Dezimalwert 14 ergibt.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
#INCLUDE ADWPAD.INC
DIM DATA_1[10000] AS INTEGER AS FIFO

EVENT:
If (DIGIN_WORD1(4) AND 3 = 3) THEN       'Abfrage,ob die Eingänge 0 und 1
                                         'gesetzt sind
  DATA_1 = ADC(1,1)     'Messwerterfassung
ENDIF
```

Siehe auch:

DIGOUT_WORD1

Modulliste:

Pro-DIO-32
Pro-OPT-16

DIGIN_WORD2

Syntax: *ERGEBNIS = DIGIN_WORD2(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse
ERGEBNIS: 16-Bit-Integer-Wert

Liefert den Pegel an den digitalen Eingänge 16-31, zusammengefaßt in einem 16-Bit-Wert.

Die Funktion gibt einen 16-Bit-Wert zurück. Jedem digitalen Eingang ist ein Bit des Rückgabewertes zugeordnet. Wenn am Eingang ein High-Pegel anliegt, dann wird das zugehörige Bit auf 1 gesetzt.

Berechnungsbeispiel:
Die Funktion *DIGIN_WORD2()* liefert den Dezimalwert 14. Anhand des Bitmusters von dezimal 14 können Sie erkennen, welche digitalen Eingänge gesetzt sind. In diesem Beispiel sind die Eingänge 17, 18 und 19 gesetzt, da die Summe der zugeordneten Bitwerte den Dezimalwert 14 ergibt.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
#INCLUDE ADWPAD.INC
DIM DATA_1[10000] AS INTEGER AS FIFO

EVENT:
If(DIGIN_WORD2(4) AND 3 = 3)THEN    'Falls die Eingänge 16 und 17 gesetzt sind
  DATA_1 = ADC(1,1)                 'Messwerterfassung
ENDIF
```

Siehe auch:

DIGOUT_WORD2

Modulliste:

Pro-DIO-32

DIGOUT_WORD1

Syntax: *DIGOUT_WORD1(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: Bitmuster

Setzt gleichzeitig die digitalen Ausgänge 0-15 der ***ADwin***-Karte auf den durch *WAHL* vorgegebenen Wert. *WAHL* ist ein 16-Bit-Wert, wobei jedem digitalen Ausgang ein Bit dieses Wertes zugeordnet ist.

Berechnungsbeispiel:
Die Ausgänge 0, 2 und 14 sollen gesetzt werden. Sie erhalten das Bitmuster 100000000000101, welches dem Dezimalwert 16389 entspricht und schreiben: DIGOUT_WORD1(16389).
Alle Ausgänge, deren zugeordnetes Bit im Wert *WAHL* nicht gesetzt ist, werden gelöscht! Im obigen Beispiel sind dies die Ausgänge 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 und 15.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER

EVENT:
wert = ADC(1,1)          'Messwerterfassung
IF (wert > 3000) THEN    'Anfang Kontrollstruktur
  DIGOUT_WORD1(4,7)      'Ausgänge 0, 1 u. 2 'setzen, alle anderen
                         'Ausgänge werden zurückgesetzt!
ENDIF                    'Ende Kontrollstruktur
```

Siehe auch:

DIGIN_WORD1

Modulliste:

Pro-DIO-32
Pro-REL-16
Pro-TRA-16

DIGOUT_WORD2

Syntax: *DIGOUT_WORD2(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: Bitmuster

Setzt gleichzeitig die digitalen Ausgänge 16-31 der ***ADwin***-Karte auf den durch *WAHL* vorgegebenen Wert. *WAHL* ist ein 16-Bit-Wert, wobei jedem digitalen Ausgang ein Bit dieses Wertes zugeordnet ist.

Berechnungsbeispiel:
Die Ausgänge 16, 18 und 30 sollen gesetzt werden. Sie erhalten das Bitmuster 100000000000101, welches dem Dezimalwert 16389 entspricht und schreiben: DIGOUT_WORD2(16389).
Alle Ausgänge, deren zugeordnetes Bit im Wert *WAHL* nicht gesetzt ist, werden gelöscht! Im obigen Beispiel sind dies die Ausgänge 15, 17, 19, 29 und 31.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER

EVENT:
wert = ADC(1,1)          'Meßwerterfassung
IF (wert > 3000) THEN    'Anfang Kontrollstruktur
  DIGOUT_WORD2(4,7)      'Ausgänge 16, 17 u. 18 setzen, alle anderen
                         'Ausgänge werden zurückgesetzt!
ENDIF                    'Ende Kontrollstruktur
```

Siehe auch:

DIGIN_WORD2

Modulliste:

Pro-DIO-32

DIG_READLATCH1

Syntax: *DIG_READLATCH1(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse

Liefert die unteren 16-Bit (Bit0 - Bit15) aus dem Zwischenregister für die digitalen Eingänge eines Digital-I/O-Moduls.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
#INCLUDE ADWINPRO.INC
DIM wert AS INTEGER

INIT:
SYNCENABLE(1,dio,1)      'Synchronisation Digital-Modul 1 freigeben
SYNCENABLE(2,dio,1)      'Synchronisation Digital-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 (Bit0-Bit15)
PAR_2= DIG_READLATCH1(2) 'Zwischenregister Modul 2 auslesen (Bit0-Bit15)
```

Siehe auch:

DIGIN_WORD1

Modulliste:

Pro-DIO-32
Pro-OPT-16

Hinweis:

Der aktuelle Zustand der digitalen Eingänge wird bei folgenden Befehlen in das Zwischenregister übernommen:

- *DIGIN_WORD1*
- *DIGIN_WORD2*
- *SYNCALL* (falls für das Modul aktiviert).

DIG_READLATCH2

Syntax: *DIG_READLATCH2(KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse

Liefert die oberen 16-Bit (Bit16 - Bit31) aus dem Zwischenregister für die digitalen Eingänge eines Digital-I/O-Moduls.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
#INCLUDE ADWINPRO.INC
DIM wert AS INTEGER

INIT:
SYNCENABLE(1,dio,1)      'Synchronisation Digital-Modul 1 freigeben
SYNCENABLE(2,dio,1)      'Synchronisation 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 (Bit16-Bit31)
PAR_2= DIG_READLATCH2(2) 'Zwischenregister Modul 2 auslesen (Bit16-Bit31)
```

Siehe auch:

DIGIN_WORD2

Modulliste:

Pro-DIO-32
Pro-OPT-16

Hinweis:

Der aktuelle Zustand der digitalen Eingänge wird bei folgenden Befehlen in das Zwischenregister übernommen:

- *DIGIN_WORD1*
- *DIGIN_WORD2*
- *SYNCALL* (falls für das Modul aktiviert).

DIG_WRITELATCH1

Syntax: *DIG_WRITELATCH1(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: Bitmuster

Schreibt den Wert: *WAHL* in die unteren 16-Bit (Bit0 - Bit15) des Zwischenregisters für die digitalen Ausgänge eines Digital-I/O-Moduls.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC
#INCLUDE ADWPDIO.INC
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER

INIT:
SYNCENABLE(1,dio,1)      'Synchronisation Digital-Modul 1 freigeben
SYNCENABLE(2,dio,1)      'Synchronisation 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:
wert = ADC(1,1)          'Meßwerterfassung
IF (wert > 3000) THEN    'Anfang Kontrollstruktur
  SYNCALL()              'Werte aus den Zwischenregistern ausgeben
ENDIF                    'Ende Kontrollstruktur
```

Siehe auch:

DIGOUT_WORD1

Modulliste:

Pro-DIO-32
Pro-REL-16
Pro-TRA-16

Hinweis:

Der Wert des Zwischenregisters für die digitalen Ausgänge wird durch folgende Befehle gesetzt:

- *DIGOUT*
- *DIGOUT_WORD1*
- *DIGOUT_WORD2*
- *DIG_WRITELATCH1*
- *DIG_WRITELATCH2.*

DIG_WRITELATCH2

Syntax: *DIG_WRITELATCH2(KARTENADRESSE,WAHL)*

KARTENADRESSE: eingestellte Kartenadresse
WAHL: Bitmuster

Schreibt den Wert: *WAHL* in die oberen 16-Bit (Bit16 - Bit31) des Zwischenregisters für die digitalen Ausgänge eines Digital-I/O-Moduls.

Anwendungsbeispiel:

```
#INCLUDE ADWINPRO.INC
#INCLUDE ADWPDIO.INC
#INCLUDE ADWPAD.INC
DIM wert AS INTEGER

INIT:
SYNCENABLE(1,dio,1)      'Synchronisation Digital-Modul 1 freigeben
SYNCENABLE(2,dio,1)      'Synchronisation Digital-Modul 2 freigeben
DIG_WRITELATCH2(1,1)     'Bit16 im Ausgangs-Zwischenregister setzen
DIG_WRITELATCH2(2,16)    'Bit20 im Ausgangs-Zwischenregister setzen

EVENT:
wert = ADC(1,1)          'Meßwerterfassung
IF (wert > 3000) THEN    'Anfang Kontrollstruktur
  SYNCALL()              'Werte aus den Zwischenregistern ausgeben
ENDIF                    'Ende Kontrollstruktur
```

Siehe auch:

DIGOUT_WORD2

Modulliste:

Pro-DIO-32
Pro-REL-16
Pro-TRA-16

Hinweis:

Der Wert des Zwischenregisters für die digitalen Ausgänge wird durch folgende Befehle gesetzt:

- *DIGOUT*
- *DIGOUT_WORD1*
- *DIGOUT_WORD2*
- *DIG_WRITELATCH1*
- *DIG_WRITELATCH2.*

DIGPROG1

Syntax: *DIGPROG1(KARTENADRESSE,WERT)*

KARTENADRESSE:	eingestellte Kartenadresse
WERT:	Bitmuster: 1 entspricht Ausgang

Initialisierung der unteren 16 Kanäle (0-15) eines ***Pro-DIO-32***-Moduls.

Jeder einzelne Kanal des ***Pro-DIO-32***-Moduls kann wahlweise als Ein- oder Ausgang betrieben werden. Nach dem Einschalten des Systems sind alle Kanäle als Eingänge konfiguriert. Um einen Kanal als Ausgang zu betreiben muß die entsprechende Bitposition von *WERT* auf 1 gesetzt werden.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC

INIT:
DIGPROG1(1, 11111B)     'Konfiguriert Kanal 0-4 des
                        'Pro-DIO-32-Moduls 1 als Ausgänge und
                        'Kanal 5-15 als Eingänge
```

Siehe auch:

DIGPROG2

Modulliste:

Pro-DIO-32

DIGPROG2

Syntax: *DIGPROG2(KARTENADRESSE,WERT)*

KARTENADRESSE:	eingestellte Kartenadresse
WERT:	Bitmuster: 1 entspricht Ausgang

Initialisierung der oberen 16 Kanäle (16-31) eines ***Pro-DIO-32***-Moduls.

Jeder einzelne Kanal des ***Pro-DIO-32***-Moduls kann wahlweise als Ein- oder Ausgang betrieben werden. Nach dem Einschalten des Systems sind alle Kanäle als Eingänge konfiguriert. Um einen Kanal als Ausgang zu betreiben muß die entsprechende Bitposition von *WERT* auf 1 gesetzt werden.

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC

INIT:
DIGPROG2(1, 11111B)     'Konfiguriert Kanal 16-20 des Pro-DIO-32-Moduls 1
                        'als Ausgänge und Kanal 21-31 als Eingänge
```

Siehe auch:

DIGPROG1

Modulliste:

Pro-DIO-32

GET_DIGOUT_WORD1

Syntax: *ERGEBNIS = GET_DIGOUT_WORD1 (KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse
ERGEBNIS: 16-Bit-Integer-Wert

Rücklesen der Ausgänge einer digitalen Karte (Bit 0-15).

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
DIM wert AS INTEGER

INIT:
wert = GET_DIGOUT_WORD1(1)          'aktuellen Wert der dig. Ausgänge lesen
wert = wert AND 0FFFEH              'letztes Bit (Bit 0) auf 0 setzen
DIGOUT_WORD1(wert)                  'geänderten Wert zurückschreiben
```

Siehe auch:

DIGIN_WORD1

Modulliste:

Pro-DIO-32
Pro-REL-16
Pro-TRA-16

GET_DIGOUT_WORD2

Syntax: *ERGEBNIS = GET_DIGOUT_WORD2 (KARTENADRESSE)*

KARTENADRESSE: eingestellte Kartenadresse
ERGEBNIS: 16-Bit-Integer-Wert

Rücklesen der Ausgänge einer digitalen Karte (Bit 16-31).

Anwendungsbeispiel:

```
#INCLUDE ADWPDIO.INC
DIM wert AS INTEGER

INIT:
wert = GET_DIGOUT_WORD2(1)          'aktuellen Wert der dig. Ausgänge lesen
wert = wert AND 0FFFDH              'vorletztes Bit (Bit 17) auf 0 setzen
DIGOUT_WORD2(wert)                  'geänderten Wert zurückschreiben
```

Siehe auch:

DIGIN_WORD2

Modulliste:

Pro-DIO-32

ADWPEXT.INC

Die Include-Datei ADWPEXT.INC beinhaltet alle Funktionen und Prozeduren, die zum Ansprechen der ***ADwin-Pro***-Extended-Module benötigt werden. In dieser Gruppe befinden sich u. a. Thermoelement-, Filter-, PT100-, und Schnittstellen-Module. Wenn Sie diese Datei mit der ***ADbasic***-Anweisung:

```
#INCLUDE ADWPEXT.INC
```

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

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

Anhand der Modulliste ist erkennbar auf welche ***ADwin-Pro***-Module die jeweilige Funktion anwendbar ist.

Anmerkung: In allen Anwendungsbeispielen wird davon ausgegangen, daß auf der Extended-Karte die Adresse 1 eingestellt ist.

TC_SELECT

Syntax: *TC_SELECT(KARTENADRESSE, KANAL)*

KARTENADRESSE: eingestellte Kartenadresse
KANAL: Kanalnummer (1 - 4(TC-4), - 8(TC-8) oder - 16(TC-16))

Schaltet den angegebenen Thermoelement-Kanal über Multiplexer auf den analogen Ausgang des Moduls.

Hinweis: Der Multiplexer benötigt ca. 3 µs Einschwingzeit. Dies Zeit muß durch entsprechende Programmierung überbrückt werden.

Anwendungsbeispiel:

```
#INCLUDE ADWPAD.INC
#INCLUDE ADWPEXT.INC
DIM wert, i AS INTEGER

INIT:
wert = 0                          'Variablen-Initialisierung
i = 1                             'Variablen-Initialisierung

EVENT:
TC_SELECT(1,3)                    'Thermoelement-Kanal 3 wählen
wert = wert + ADC(1,1)            'Wert messen
IF (i=10) THEN
  PAR_1=wert/10                   'Mittelwert aus 10 Messungen
  wert=0
  i=1
ENDIF
i=i+1
```

Hinweis:

Im Beispiel wird davon ausgegangen, daß der analoge Ausgang des Thermoelement-Moduls mit dem analogen Eingang1 eines A/D-Moduls mit Kartenadresse 1 verbunden ist.
Zur Umrechnung der gemessenen Spannung in die Einheit °C, wird die Umrechnungstabelle des Thermoelement-Verstärkers AD594(Typ J) bzw. AD595(Typ K) benötigt. Sie finden diese Tabellen in der Windows-Hilfe mit der Bezeichnung **THERMO.HLP**

Modulliste:

Pro-TC-4
Pro-TC-8
Pro-TC-16
Pro-PT100-4
Pro-PT100-8

Programmbeispiele

Online-Auswertung von Meßwerten

Das Programm BAS_DMO1.BAS sucht den Maximal- und Minimalwert aus 1000 Messungen von ADC1 und schreibt das Ergebnis in die Variablen PAR_1 und PAR_2.
Das Programm geht davon aus, daß ein 16-Bit-A/D-Modul verwendet wird, auf dem die Kartenadresse 1 eingestellt ist.

```
#include adwpad.inc                 'Include-Datei für das ADwin-Pro-System


dim i1, iw, max, min as integer

INIT:
  i1 = 1
  max = 0
  min = 65535

EVENT:
  iw = ADC16(1,1)                   'Messung durchführen
  if (iw>max) then max = iw
  if (iw<min) then min = iw
  i1 = i1+1
  if (i1>1000) then
    i1 = 1
    par_1 = min                     'Minimalwert in Parameter1 speichern
    par_2 = max                     'Maximalwert in Parameter2 speichern
    max = 0
    min = 65535
    activate_pc                     'Nach 1000 Messungen den PC informieren
    endif
```

Digitaler Proportional-Regler

Das Programm BAS_DMO2.BAS ist ein digitaler P-Regler. Der Sollwert wird mit PAR_1 vorgegeben, die Verstärkung mit PAR_2.
Das Programm geht davon aus, daß ein 16-Bit-A/D-Modul verwendet wird, auf dem die Kartenadresse 1 eingestellt ist. Auf dem D/A-Modul sollte ebenfalls die Adresse 1 eingestellt sein.

```
#include adwpad.inc                ' Include-Dateien für das ADwin-Pro-System
#include adwpda.inc

dim iw,abweichung, stell as long
dim Sollwert, Verstärkung as long


EVENT:

Sollwert    = Par_1 * 16
Verstärkung = Par_2

Rem Istwert messen und Regelabweichung berechnen :
abweichung = Sollwert - ADC16(1,1)

Rem Stellwert berechnen :
stell = stell + (abweichung * Verstärkung/100)

Rem neuen Stellwert ausgeben :
DAC(1, 1, stell)
```

Datenaustausch mit DATA

Das Programm BAS_DMO3.BAS mißt den analogen Eingang 1 des A/D-Moduls mit Adresse 1 und informiert nach 1000 Messungen den PC, der dann die Meßwerte abholen kann. Die Daten werden mit Hilfe eines DATA-Arrays übertragen.

```
#include adwpad.inc                'Include-Datei für das ADwin-Pro-System

dim data[1][2000] as long
dim index

INIT:
index = 0

EVENT:
index = index+1


REM Nach 1000 Messungen PC informieren und das Programm beenden
if (index>1000) then
  ACTIVATE_PC
  END
ENDIF

REM Messwert aufnehmen
ih = ADC16(1,1)
data[1][index] = ih
```

Digitaler PID-Regler

Das Programm BAS_DMO6 ist ein digitaler PID-Regler. Damit das Programm auch auf ***ADwin-Pro***-Prozessormodulen mit T400- und T450-Prozessoren benutzt werden kann, arbeitet der Regler nur mit Integer-Variablen. Die Reglerkoeffizienten werden auf dem PC berechnet, mit 1000 multipliziert und als Integer-Parameter an den Prozessor des ***ADwin-Pro***-Systems uebergeben.
Die Adressen des analogen Eingangs- und des analogen Ausgangsmoduls sind im Beispiel beide auf Adresse 1 eingestellt.

Par_1 = Sollwert in Digits
Par_2 = q0 * 1000 = k *(1+t0/(2*ti)+td/t0) *1000
Par_3 = q1 * 1000 = -k * (1+2*td/t0-t0/(2*ti)) * 1000
Par_4 = q2 * 1000 = k * td/t0 *1000
Par_5 = Bufferindex fuer die Regelabweichung

```
#include adwpad.inc                 'Include-Dateien für das ADwin-Pro-System
#include adwpda.inc
dim iw,abweichung as long
dim Sollwert, Verstärkung as long
dim ek, ek1, ek2, uk, uk1, i as long
dim q0,q1,q2 as long
dim data_1[600] AS INTEGER

INIT:
sollwert = par_1 *16
q0=par_2
q1=par_3
q2=par_4
par_5 = 0

EVENT:
set_mux(1,0)              'Multiplexer setzen
uk1 = uk + q0*ek          'Stellwert berechnen
start_conv(1)             'Messung starten
dac (1,1,(uk1/1000))      'Stellwert ausgeben
                          'Da alle Parameter mit 1000 multiplziert wurden,
                          'muß er durch 1000 dividiert werden
ek2=ek1
ek1=ek
uk = uk1+q1*ek1+q2*ek2
wait_eoc(1)               'Warten bis der ADC ausgelesen werden kann

REM Bei 16-Bit A/D-Modulen muß die Messung nochmal gestartet werden, da
REM ... der A/D-Wandler den Wert der letzten Messung liefert
  start_conv(1)             'Messung starten
  wait_eoc(1)               'Warten bis der ADC ausgelesen werden kann
REM Bei 12-Bit A/D-Modulen entfällt die 2. Messung

ek = sollwert-readadc(1) 'Auslesen und Regelabweichung berechnen

if (par_5<500) then         'Regelabweichung in einen Buffer schreiben
  data[1][par_5] = ek
  par_5=par_5+1
  if (par_5=500) then activate_pc
endif

if (par_9=1) then         'Falls die Parameter geaendert wurden,
  sollwert = par_1 *16    'neue Parameter uebernehmen
  q0=par_2
  q1=par_3
  q2=par_4
  par_9=0
  par_5=0
  endif
```

Befehlsübersicht (alphabetisch)

W