ADbasic

Echtzeit-Entwicklungstool für ADwin-Systeme

Version 3.0

S/N

November 1999

Mikrosekundengenaue Echtzeitverarbeitung
- schnell und einfach programmiert

Inhaltsverzeichnis

3.1

Liebe Leserin, lieber Leser!

Dieses Handbuch zu ***ADbasic 3.0*** ist für Anwender der ***ADwin***-Systeme, die mit Hilfe von ***ADbasic*** Programme zur schnellen Echtzeitverarbeitung durch die ***ADwin***-Systeme erstellen möchten. Sie sollten über grundlegende Kenntnisse des Programmierens z. B. in BASIC verfügen, bevor Sie dieses Handbuch verwenden.

Anwender von *ADwin-Pro*-Systemen:

Die Befehle zum Ansprechen eines ***ADwin-Pro***-Systems mit ***ADbasic*** werden in speziellen INCLUDE-Dateien zur Verfügung gestellt. Lesen Sie hierzu bitte unbedingt die Dokumentation mit dem Titel „***ADwin-Pro*** : Systembeschreibung ; Programmierung in ***ADbasic***".

Anwender von *ADbasic 2.0*:

Wenn Sie ***ADbasic 2.0*** kennen, dann lesen Sie bitte zunächst Kapitel 1 (Neue Funktionen) und anschließend die Kapitel 4.3 (Speicherverwaltung) und 5.2 (Ermitteln der Ablaufzeiten mit Timerfunktionen)

...Anwender mit Programmiererfahrung aber ohne Erfahrung mit *ADbasic 2.0* empfehlen wir zusätzlich:

Kapitel 2 (Software-Installation), Kapitel 4.4 (***ADbasic***-Datentypen) und Kapitel 4.2 (Struktur eines ***ADbasic-Programm***) zu lesen.

Konventionen für dieses Handbuch

Wir verwenden die folgenden typographischen Konventionen:

Beispiel	**Anwendung**
! **Achtung:**	Steht vor einem Absatz in dem ein oder mehrere wichtige Hinweise für eine korrekte Funktion gegeben werden.
wichtig	Dieser Schriftstil dient zur Hervorhebung wichtiger Stellen in einem Text und zur Kennzeichnung von Fachbegriffen und Eigennamen.
♦ *Geben Sie ein*	Bei diesen Absätzen müssen Sie an Ihrem PC etwas eingeben, damit Sie eines unserer Beispiele ausprobieren können.
`Schreibmaschine`	Diesen Schrifttyp verwenden wir für Programmanweisungen und alle anderen Eingaben, die Sie im Editor-Fenster vornehmen können. Im Referenzteil ist bei den Beispielen der jeweils besprochene Befehl fett hervorgehoben.
`kursiv`	Setzen wir Variablen-Namen, die in einem Programmbeispiel verwendet werden.
BEFEHL	***ADbasic***-Befehle sind zur besseren Unterscheidung in Großbuchstaben gesetzt. Die Schreibweise in Ihrem Programm ist Ihnen jedoch freigestellt, ***ADbasic*** versteht auch klein geschriebene Befehle.

{BEFEHL}	Dieser Teil eines ***ADbasic***-Befehls kann, muß aber nicht verwendet werden (optional).
File ⇨ Save	Die Namen von Menüs und Untermenüs sind fett gedruckt. Der Pfeil weist auf das jeweils untergeordnete Menü.
'Delay'	Die Bezeichnungen von Eingabe- oder Auswahlfeldern in Dialogfenstern sind in einfache Anführungsstriche gesetzt.
KAPITÄLCHEN	Tastennamen wie z. B. RETURN oder CTRL.
Hinweis:	Steht vor einem Absatz mit ergänzenden Erläuterungen.

1 Neue Funktionen

Die ***ADbasic*** Version 3.0 haben wir gegenüber der ***ADbasic*** Version 2.0 um folgende Funktionen erweitert:

- Es können ***ADbasic***-Prozesse für das in den ***ADwin***-Systemen[1] ***ADwin***, ***ADwin-light*** und ***ADwin-Pro*** verwendete Prozessormodul SHARC ADSP21062 von Analog Devices[2] kompiliert werden.
- Das ***ADbasic***-Dialogfenster Parameter ist für den ADSP angepaßt.
- ***ADbasic 3.0*** ist jetzt lauffähig unter *Windows 95*, *Windows 98* und *Windows NT* aber nicht (mehr) unter Windows 3.1x

Hinweis: Für Anwender, die weiterhin mit *Windows 3.1x* und ***ADbasic*** arbeiten möchten ist auf der CD-ROM in dem Verzeichnis ADbasic ein Unterverzeichnis mit der Bezeichnung win31.

In diesem Unterverzeichnis befindet sich der ***ADbasic***-Compiler in der Version 2.0 und eine PDF-Datei (Handbuch zu ***ADbasic 2.0***).

1 Die Hardware-Systeme ***ADwin***, ***ADwin-light*** (PC-Einsteckkarten) und ***ADwin-Pro*** (19“-Zoll-Systeme) werden im folgenden als ***ADwin***-Systeme bezeichnet sofern die getroffene Aussage auf alle Hardware-Systeme zutrifft.

2 Dieses Prozessormodul wird im folgenden als ADSP bezeichnet.

2 Software-Installation

2.1 Systemvoraussetzungen

Für das Echtzeit-Entwicklungstool ***ADbasic 3.0*** wird ein Rechner benötigt, auf dem *Windows 95*, *Windows 98* oder *Windows NT* installiert ist.

Außerdem wird ein CD-ROM-Laufwerk für die Installation von ***ADbasic 3.0*** benötigt.

Hinweis: Es besteht die Möglichkeit, daß Sie sich einen Diskettensatz von ***ADbasic 3.0*** und den ***ADwin***-Treibern anfertigen, um eine Installation von Diskette durchführen zu können. Kopieren Sie dazu die Verzeichnisse Disk 1 bis Disk 2 aus dem ***ADbasic***-Verzeichnis bzw. die Verzeichnisse Disk 1 bis Disk 2 aus dem Driver-Verzeichnis der CD-ROM.

Kompilierte ***ADbasic***-Programme sind auch auf einem Rechner mit *Windows 3.1x* ausführbar, wenn auf diesem Rechner die ***ADwin***-Treiber und die Win32s-Erweiterung installiert ist.

Sinnvoll ist es, eines der gängigen Meßdaten-Auswertungsprogramme (wie z. B. *TestPoint*) zu installieren, das sich gut mit ***ADwin*** und ***ADbasic*** kombinieren läßt. Sie können aber auch Ihr eigenes Auswerteprogramm z. B. mit *Visual Basic*, *Visual C* oder *Borland Delphi* schreiben.

Die ***ADbasic***-Entwicklungsumgebung ist ein 32-Bit-Windows-Programm und läuft vollständig auf dem PC ab. Ein ***ADwin***-System wird nur zum Testen der ***ADbasic***-Programme benötigt.

Obwohl ***ADbasic*** selbst nicht viel Speicher benötigt, ist es für die Ausführung von Meßdaten-Auswerteprogrammen hilfreich, viel Arbeitsspeicher und einen schnellen Rechner zu besitzen. Bitte lesen Sie dazu die Dokumentation Ihres Anwendungsprogrammes.

2.2 Auswahl des Betriebssystems

Die Version 3.0 von ***ADbasic*** ist ein 32-Bit-Programm und benötigt deshalb *Windows 95*, *Windows 98* oder *Windows NT*.

Das Setup-Programm erkennt das auf Ihrem Rechner installierte Betriebssystem.

2.3 Installation der ADwin-Treiber

Legen Sie die mitgelieferte CD-ROM in das CD-ROM-Laufwerk Ihres Rechners. Das Setup-Programm wird automatisch gestartet[3]. Betätigen Sie den Button '***ADwin*** Driver Setup'. Nachdem Sie die Sprache und das Zielverzeichnis (es wird empfohlen das Verzeichnis C:\ADBASIC3 zu bestätigen) gewählt haben, kopiert das Setup-Programm folgende Dateien in das gewählte Verzeichnis Ihres Rechners:

ADWIN2.BTL	Treiber für die ***ADwin***-Systeme mit T225-Prozessor
ADWIN4.BTL	Treiber für die ***ADwin***-Systeme mit T400-Prozessor
ADWIN5.BTL	Treiber für die ***ADwin***-Systeme mit T450-Prozessor
ADWIN8.BTL	Treiber für die ***ADwin***-Systeme mit T805-Prozessor
ADWIN9.BTL	Treiber für die ***ADwin***-Systeme mit ADSP-Prozessor
TESTVE16.EXE	Sucht und zeigt die Version der installierten 16-Bit-***ADwin***-DLLs an

[3] Wenn das Setup-Programm nicht automatisch gestartet wird, dann führen Sie das in dem Unterverzeichnis ...\Driver\Disk1 befindliche **Programm: setup.exe aus.**

TESTVE32.EXE	Sucht und zeigt die Version der installierten 32-Bit-***ADwin***-DLLs an
ADTEST.EXE	Programm zum Testen des ***ADwin***-Systems
ADWINSET.EXE	Programm zum Anmelden der Linkadresse unter *Windows NT* (lesen Sie hierzu Kapitel 2.3.1)
ADSERVER.EXE	Programm zur Initialisierung der ***ADwin***-Netzwerkfunktion

Außerdem werden je nach verwendetem Betriebssystem automatisch folgende DLLs (Dynamic Link Libraries) in das Windows-Verzeichnis Ihres Rechners kopiert:

Windows 95	***Windows 98***	***Windows NT***
ADWIN.DLL	ADWIN.DLL	ADWIN.DLL
ADWIN32.DLL	ADWIN32.DLL	ADWIN32.DLL
ADPOINT.DLL	ADPOINT.DLL	ADPOINT.DLL
ADWIN95.DLL	ADWIN95.DLL	ADWINNT.DLL

! Achtung: Wenn Sie bereits ***ADwin***-Treiber einer früheren ***ADbasic***-Version auf Ihrem PC installiert haben, dann werden Sie von dem Setup-Programm gefragt, ob Sie vorhandene (DLL-)Dateien löschen möchten. Bestätigen Sie diese Abfragen mit „Ja“.

Nach ordnungsgemäßer Beendigung des Setup-Programms erscheint die Meldung: Das Setup wurde erfolgreich durchgeführt.

2.3.1 Hardware-Inbetriebnahme bei Windows NT

Alle ***ADwin***-Systeme arbeiten über Linkadressen, die auf der Hardware mit Hilfe von DIP-Schaltern eingestellt wird (vergleichen Sie hierzu bitte das jeweilige Hardware-Handbuch). Standardmäßig ist diese Linkadresse immer auf 150H eingestellt.

Wenn Sie auf Ihrem Rechner *Windows NT* installiert haben und eine andere als die standardmäßig eingestellte Linkadresse benutzen möchten, dann müssen Sie das in der Programmgruppe ***ADwin*** installierte Programm *AdwinSet* ausführen.

Hinweis: Bei der Ausführung von *AdwinSet* wird auf die Registrierungs-Datenbank zugegriffen. Wenn Sie Änderungen darin vornehmen möchten, dann müssen Sie Administrator-Zugriffsrechte besitzen.

In dem Dialogfeld von *AdwinSet* können Sie:

- bis zu acht Linkadressen in die Registrierungs-Datenbank eintragen,
- Linkadressen hinzufügen,
- Linkadressen markieren und ändern,
- Linkadressen löschen.

Nachdem Sie Ihre Eingaben in *AdwinSet* bestätigt haben, müssen Sie den Rechner neu starten.

Hinweis: Sie müssen die Linkadresse außerdem in ***ADbasic*** (**Options ⇨ Compiler**) bzw. in dem von Ihnen verwendeten Meßdatenauswerteprogramm angeben.

2.4 Installation des ADbasic-Compilers

Legen Sie die mitgelieferte CD-ROM in das CD-ROM-Laufwerk Ihres Rechners. Das Setup-Programm wird automatisch gestartet[4]. Betätigen Sie den Button '***ADbasic*** 3.0 Setup'. Nachdem Sie die Sprache und das Zielverzeichnis (es wird empfohlen das Verzeichnis C:\ADBASIC3 zu bestätigen) gewählt haben, kopiert das Setup-Programm folgende Dateien in das gewählte Verzeichnis Ihres Rechners:

ADBASIC.EXE	***ADbasic***-Compiler
ADBASICE.HLP	***ADbasic***-Hilfedatei (englisch)
ADBASIC.HLP	***ADbasic***-Hilfedatei (deutsch)
ADSHOW.EXE	Programm zum Anzeigen von Prozeßparametern und Variablen
...\SAMPLES\	***ADbasic*** Beispielprogrammsammlung

[4] Wenn das Setup-Programm nicht automatisch gestartet wird, dann führen Sie das in dem Unterverzeichnis ...\ADbasic\Disk1 befindliche **Programm: setup.exe aus.**

2.5 Laden des ADwin-Treibers

Der PC kann erst mit einem ***ADwin***-System kommunizieren, nachdem der ***ADwin***-Treiber auf das ***ADwin***-System geladen wurde. Der ***ADwin***-Treiber muß in jedem Fall nach jeder Unterbrechung der Spannungsversorgung des ***ADwin***-Systems geladen werden.

! Achtung: Durch das Laden des ***ADwin***-Treibers werden alle Prozesse auf dem ***ADwin***-System gelöscht und alle globalen Variablen auf den Wert 0 gesetzt.
Der Wert für das Globaldelay ist nach dem Laden des ***ADwin***-Treibes beim ADSP auf 25000 ns und bei allen anderen Prozessoren auf 1000 µs voreingestellt.

- *Starten Sie **ADbasic** indem Sie in dem Windows-Menü **Start ⇨ Programme ⇨ ADwin ⇨ ADbasic 3.0** auswählen.*

- *Überprüfen Sie, ob die Einstellungen im **ADbasic**-Menü **Options ⇨ Compiler** mit Ihrem **ADwin**-System übereinstimmen.*

- *Klicken Sie auf dieses Symbol in der Symbolleiste:*  *(alternativ: **Project ⇨ Boot ADwin**)*

Das erfolgreiche Laden des Treibers wird in der Statuszeile des ***ADbasic***-Editors durch die Meldung „ADwin is booted“ bestätigt.

3 Grundlagen und Arbeitsweise von *ADbasic*

3.1 Einführung

ADbasic ist ein Compiler, dessen Programmiersprache eng an BASIC angelehnt ist, um Ihnen einen schnellen und leichten Einstieg zu ermöglichen.

Mit ***ADbasic*** erstellen Sie Programme für Ihr ***ADwin***-System. Die mit ***ADbasic*** erstellten Programme werden ausschließlich auf diesen Systemen abgearbeitet. Ihrem System wird damit eine Echtzeitfähigkeit mit extrem kurzen Reaktionszeiten verliehen. Zusammen mit Auswerteprogrammen wie z. B. *TestPoint* der Firma Keithley bildet dies ein leistungsfähiges System zum gleichzeitigen Messen, Steuern und Regeln unter *MS Windows*.

Die Kommunikationsfunktionen zwischen PC und ***ADwin***-System werden dabei von ***ADbasic*** übernommen. Sie brauchen sich also nur noch auf die Programmierung des zyklischen Ablaufs zu konzentrieren. Da die Kommunikation bidirektional ist, können sowohl Meßergebnisse von dem ***ADwin***-System an den PC als auch Steuerparameter und Vorgaben vom PC an das ***ADwin***-System oder den Prozeß geschickt werden. Um möglichst schnell arbeiten zu können, wird Ihr ***ADbasic***-Programm nicht, wie manche BASIC-Dialekte, interpretiert und ausgeführt, sondern zuerst kompiliert und dann auf das ***ADwin***-System übertragen und vom Prozessor ausgeführt. Selbstverständlich können Sie auch Programme entwickeln, testen und kompilieren, um Sie zu einem späteren Zeitpunkt z. B. aus Ihrem Anwendungsprogramm heraus in das ***ADwin***-System zu laden.

3.1.1 *ADbasic* und *ADwin*

Mit jedem ***ADwin***-System haben Sie einen Prozeßrechner, der sich ganz der Messung, Steuerung oder Regelung widmen kann. Dieser Prozeßrechner, wurde für schnelle Taskwechsel entwickelt und optimiert. Er ist daher in der Lage, innerhalb weniger Mikrosekunden zu reagieren. Mit diesem System können Sie die langsame Reaktion und mangelnde Echtzeitfähigkeit von *MS Windows* umgehen und

trotzdem alle Vorteile nutzen, wie z. B. die grafische Oberfläche oder die leichte Bedienbarkeit.

Mit ***ADbasic*** haben Sie ein Entwicklungstool, mit dem Sie Meß-, Regel- oder Überwachungsprogramme entwickeln können. Diese Prozesse werden dann in das Standardprogramm eingebaut, das ständig auf dem ***ADwin***-System abläuft und alle Aufgaben der Kommunikation und Verwaltung erledigt. Somit laufen die ***ADbasic***-Programme *unabhängig* von Ihrem PC, der sich anderen Aufgaben widmen kann.

Mit ***ADbasic*** können Sie ausführbare Programme erzeugen, die Sie zusammen mit einem Auswerteprogramm laden und starten können. ***ADbasic*** benötigen Sie deshalb nur für die Entwicklung der Programme, nicht für deren Ausführung. Die mit ***ADbasic*** kompilierten Programme können aber ausschließlich auf einem ***ADwin***-System ausgeführt werden.

Hinweis: Die Befehle zum Ansprechen eines ***ADwin-Pro***-Systems mit ***ADbasic*** werden in speziellen INCLUDE-Dateien zur Verfügung gestellt. Lesen Sie hierzu bitte unbedingt die Dokumentation mit dem Titel „***ADwin-Pro*** : Systembeschreibung ; Programmierung in ***ADbasic***“.

3.1.2 Wie können Sie *ADbasic* einsetzen?

Mit ***ADbasic*** haben Sie ein leicht zu handhabendes Tool, um Programme zu schreiben, die dann mit hoher Geschwindigkeit von dem Prozessor eines ***ADwin***-Systems ausgeführt werden. So können Sie Echtzeitanwendungen wie digitale Regler, Sollwertüberwachungen etc. realisieren. Ein besonderer Vorteil ist dabei, daß die Reaktionszeiten auf neue Ereignisse, sogenannte Events, bei der Verwendung eines vom PC unabhängigen Prozessors extrem kurz sind. Typische Anwendungen sind:

- schnelle Meßgrößenerfassung bis zu Abtastfrequenzen von 800 kHz
- Entwicklung schneller digitaler Regler mit Abtastraten bis zu 400 kHz

- gleichzeitige Erzeugung und Messung analoger Signale, z. B. für dynamische Kennlinienmessungen
- schnelle Steuerungs- und Überwachungsaufgaben mit sicheren Antwortzeiten unter 300 ns

Sie können sogar mehrere Prozesse gleichzeitig ablaufen lassen! Um Ihnen eine Vorstellung von der Geschwindigkeit der Ausführung und dem geringen Aufwand zur Programmierung zu geben, möchten wir Ihnen das Beispiel P-Regler aus Abb. 3-1 vorstellen.

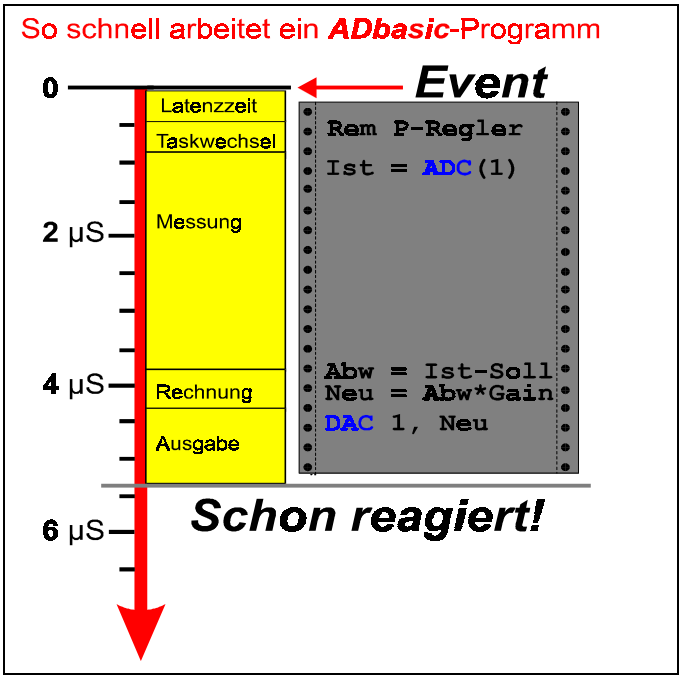


Abb. 3-1: Das Event-Konzept am Beispiel eines P-Reglers

Sie sehen, mit nur vier Zeilen Programm können Sie einen schnellen digitalen P-Regler realisieren! Wenn Sie den internen Timer so einstellen, daß alle 20 µs ein Event-Signal erzeugt wird, bekommen Sie eine Frequenz von 50 kHz, mit der dieses Programm ausgeführt wird.

4 Programmieren in *ADbasic*

In diesem Kapitel möchten wir Ihnen das Programmieren und den Aufbau eines typischen ***ADbasic***-Programms zeigen.

4.1 Starten der Entwicklungsumgebung

- *Starten Sie **ADbasic** indem Sie in dem Windows-Menü **Start ⇨ Programme ⇨ ADbasic ⇨ ADbasic 3.0** auswählen.*

Es erscheint die ***ADbasic***-Oberfläche mit den *Windows*-typischen Menüs, einer Leiste mit Schaltflächen (Buttonleiste) und dem Editor-Fenster.

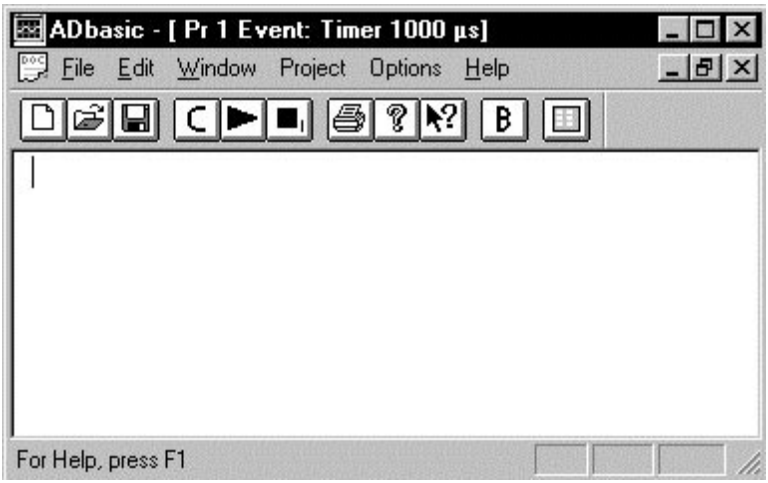


Abb. 4-1: Die ***ADbasic***-Oberfläche

- *Bitte überprüfen Sie im Menü **Options ⇨ Compiler**, ob der Prozessortyp sowie die 'Linkaddress' richtig eingestellt sind. (Wenn Sie keine Änderung an Ihrem **ADwin**-System vorgenommen haben, ist die Einstellung für 'Linkaddress' 0x150.)*

4.2 Struktur eines ADbasic-Programms

! Achtung: Im Gegensatz zu anderen BASIC-Compilern müssen Sie in jedem ***ADbasic***-Programm *zuerst* die Variablen deklarieren. Ausnahme: die globalen Variablen PAR_x und FPAR_x.

Der sich an die Variablendeklaration anschließende Programmtext kann für alle in ***ADwin***-Systemen verwendeten Prozessoren in die folgenden drei Abschnitte unterteilt werden:

```
INIT:

EVENT:

FINISH:
```

Bei ***ADwin***-Systemen mit ADSP kann das ***ADbasic***-Programm einen vierten Abschnitt enthalten. Dieser vierte Abschnitt wird unabhängig von der für den Prozeß eingestellten Priorität immmer mit niedriger Priorität ausgeführt.

Hinweis: Der Abschnitt `EVENT:` *muß* in Ihrem Programm enthalten sein. Die anderen beiden Abschnitte können wahlweise angelegt oder weggelassen werden.

Der Abschnitt LOWINIT:

Hinweis: Der Abschnitt `LOWINIT:` kann nur bei ***ADwin***-Systemen mit ADSP verwendet werden!

Dieser Abschnitt wird, wie der Abschnitt `INIT:`, nach dem Programmstart nur einmal ausgeführt. Der Abschnitt `LOWINIT`: wird unabhängig von der Priorität des Prozesses in dem er verwendet wird, immer mit niedriger Priorität ausgeführt. Dadurch unterscheidet er sich wesentlich von dem Abschnitt `INIT:`. Der Abschnitt `LOWINIT:` muß immer vor dem Abschnitt `INIT:` stehen.

Der Abschnitt INIT:

Alle Programmzeilen, die sich zwischen den Befehlen INIT: und EVENT: befinden, werden nach dem Programmstart genau einmal mit der für den Prozeß eingestellten Priorität ausgeführt. Dieser Abschnitt wird verwendet, um für Ihren Prozeß einen definierten Anfangszustand zu erzeugen, z. B. Ihre Variablen zu initialisieren oder die digitalen Ausgänge zu setzen.

Falls Sie keine Initialisierungen vornehmen wollen, können Sie den Abschnitt auch weglassen. Ihr erster Programmabschnitt nach der Variablendeklaration beginnt dann mit dem Abschnitt EVENT:.

Der Abschnitt EVENT:

Im Abschnitt EVENT: befindet sich das eigentliche Meßgrößenerfassungs- bzw. Datenverarbeitungsprogramm. Die Programmzeilen, die *zwischen* den Befehlen EVENT: und FINISH: angesiedelt sind, werden nach jedem auftretenden Event-Signal abgearbeitet. Falls Ihr Programm keinen Abschnitt FINISH: enthält, erstreckt sich der Abschnitt EVENT: von der Anweisung EVENT: bis zum Ende Ihres Programms.

Ein Event-Signal wird wahlweise vom Timer auf dem ***ADwin***-System oder von einer positiven Flanke am Event-Eingang erzeugt.

Der Abschnitt EVENT: wird so lange immer wieder abgearbeitet, bis der Prozeß vom PC aus gestoppt wird oder bei der Programmierung von Programmschleifen die vorgegebene Anzahl von Durchläufen erreicht wurde.

Der Abschnitt FINISH:

Alle Befehle, die sich hinter der Anweisung FINISH: befinden, werden genau einmal abgearbeitet, nachdem der Prozeß gestoppt wurde. Den Abschnitt FINISH: verwenden Sie, um das System in einen definierten Endzustand zu bringen. Falls Sie den Abschnitt FINISH: nicht benötigen, kann er entfallen.

! **Achtung:** Der Abschnitt FINISH: wird immer mit niedriger Priorität ausgeführt, selbst wenn Ihr Prozeß mit hoher Priorität gestartet wurde.

In allen drei (für den ADSP vier) Blöcken haben Sie die Möglichkeit, mit dem PC Daten auszutauschen. Dies kann in beiden Richtungen geschehen.

Um die Daten anzeigen und auswerten zu können, benötigen Sie ein Meßdatenauswerteprogramm wie etwa *TestPoint* oder *MATLAB*. Selbstverständlich können Sie sich auch mit *Visual Basic, Visual C, Delphi* etc. ein eigenes Meßdatenauswerteprogramm gestalten. Für jede dieser Möglichkeiten ist Treibersoftware erhältlich, die ***ADbasic*** und ***ADwin*** mit dem gewünschten Programm verbindet - wenn Sie spezielle Wünsche haben, fragen Sie bei uns an.

4.3 Speicherverwaltung

Ihre ***ADbasic***-Programme verwenden den auf den ***ADwin***-Systemen vorhandenen Speicher für den Programmcode und für die Daten.

Die Dateigröße im Editor ist nicht beschränkt. Auch die Größe der kompilierten Datei ist nur durch die Größe des Speichers des ***ADwin***-Systems begrenzt. Da die ***ADbasic***-Programme kompiliert werden, sind sie entsprechend kompakt und belegen nur wenige kByte Ihres RAM-Speichers auf dem ***ADwin***-System.

- *Klicken Sie auf dieses Symbol in der Symbolleiste:*  *, um sich den freien Speicher anzeigen zu lassen (alternativ: **Options ⇨ Parameter**)*

Hinweis: Der freie Speicher wird im Parameterfenster angezeigt. Beachten Sie aber bitte, daß das Parameterfenster für den ADSP aufgrund der Speicherstruktur anders aussieht als für die Prozessoren T225, T400, T450 und T805.

Bei Systemen mit T225-Prozessor kann es zu einer Einschränkung der Programmgröße kommen, da bei diesen Systemen nur 64 kB Speicher zur Verfügung stehen, von denen etwa die Hälfte für den ***ADwin***-Treiber (adwin2.btl) benötigt werden. Bitte beachten Sie in diesem Fall die Aufteilung des verbleibenden Speichers besonders sorgfältig.

Die ***ADwin***-Treiber für die T400- und T805-Prozessoren (adwin4.btl bzw. adwin8.btl) benötigen etwa 70 kB, so daß im allgemeinen weniger als 100 kB von dem auf dem Prozessor vorhandenen RAM belegt sind. Der Rest steht Ihnen dann für Ihre Daten zu Verfügung, bei einer Bestückung mit 1 MB also rund 900 kB.

Bei Systemen mit T450-Prozessor benötigt der Treiber (adwin5.btl) etwa 120 kB. Der verbleibende Speicher steht Ihnen für Daten und ***ADbasic***-Programme zur Verfügung.

Systeme mit ADSP verfügen über einen sehr schnellen internen Speicher von 256 kB. Dieser Speicher wird vom Prozessor in die zwei Bereiche Programmspeicher und Datenspeicher von je 128 kB aufgeteilt wird. Der erste Bereich ist für den ***ADwin***-Treiber (adwin9.btl), sowie für die mit ***ADbasic*** erstellten Programme reserviert.

Die zweiten 128 kB stehen für Daten zur Verfügung. In diesem Bereich liegen die globalen Parameter PAR_1 bis PAR_80 bzw. FPAR_1 bis FPAR_80, sowie alle lokal deklarierten, diskreten Variablen für die nicht explizit ein anderer Bereich angegeben wird.

Zum Speichern von Meßdaten oder Signalverläufen ist der ADSP in der Standardausführung zusätzlich mit 4 MB dynamischem Speicher (DRAM) bestückt, der auf maximal 32 MB erweiterbar ist. Dort legt ***ADbasic*** alle Daten aus Array-Strukturen ab, sofern bei der Deklaration nicht explizit ein anderer Speicherbereich angegeben wird. Bei der Deklaration von Array-Strukturen besteht die Möglichkeit, die Speicherart zu spezifizieren, in der die Struktur angelegt werden soll.

Da der Zugriff auf den Prozessorspeicher fünfmal so schnell wie auf den dynamischen Speicher erfolgt, ist es sinnvoll kurze Array-Strukturen, die einen sehr schnellen Zugriff erfordern, in den internen Datenspeicher des ADSPs zu legen.

Die Beispiele in Tabelle 4-1 zeigen alle möglichen Varianten der Speicherspezifizierung auf:

Speicherart	dynamischer Speicher (DRAM)
Deklaration	`DIM DATA_1[1000] AS LONG`
Speicherart	interner Datenspeicher des ADSP
Deklaration	`DIM signal[10] AS LONG AT DM_LOCAL`
Speicherart	dynamischer Speicher (DRAM)
Deklaration	`DIM DATA_1[10000] AS LONG AT DRAM_EXTERN`
Speicherart	statischer Speicher (SRAM)
Deklaration	`DIM DATA_1[1000] AS LONG AT SRAM_EXTERN`

Tabelle 4-1: Speicherarten des ADSP 21062

Hinweis: Wenn Sie keine Angabe über die Art des Speichers machen, dann legt ***ADbasic*** die Struktur automatisch in den dynamischen Speicher.

4.4 ADbasic-Datentypen

4.4.1 Schreibweise von Zahlen

ADbasic-Zahlenwerte können wahlweise in einer von vier möglichen Schreibweisen angegeben werden. In den folgenden Beispielen wird die Variable *x* jeweils auf den Dezimalwert 90 gesetzt.

Beispiele:

1. Dezimalschreibweise: `x = 90`
2. Exponentialschreibweise: `x = 9E1`

   In der Exponentialschreibweise gibt die Zahl hinter dem E die Zehnerpotenz an, mit der die Zahl vor dem E multipliziert wird.
3. Binärschreibweise: `x = 1011010B`
4. Hexadezimalschreibweise: `x = 5aH`

   Falls der HEX-Wert mit einem Buchstaben beginnt, z. B. feH, muß diesem eine Null vorangestellt werden, also 0feH.

Hinweis: Das Dezimaltrennzeichen für Fließkommazahlen ist der Punkt (.), verwenden Sie daher nicht die deutsche, sondern die englische Schreibweise für Zahlen. Die Einstellung erfolgt über das PC-Betriebssystem: **Start ⇨ Einstellungen ⇨ Systemsteuerung ⇨ Ländereinstellungen.**

4.4.2 Vordefinierte globale Variablen

Zum bidirektionalen Datentransfer zwischen parallel laufenden ***ADbasic***-Prozessen oder zwischen PC und ***ADbasic***-Prozessen stehen 80 globale Variablen sowie bis zu 200 Datensätze (Arrays) zur Verfügung. Beim T805 und beim ADSP sind zusätzlich 80 globale Fließkomma-Variablen vordefiniert.

Sie können diese skalaren Variablen in Ihren Programmen an jeder beliebigen Stelle einsetzen, *ohne* sie zu deklarieren. Die skalaren Variablen haben die Namen:

- PAR_1, PAR_2,..., PAR_80 für ganzzahlige 32-Bit-Werte (LONG)
- FPAR_1, FPAR_2, ...FPAR_80 für Fließkommawerte auf dem T450, T805 und ADSP

Beispiele:

```
PAR_5 = 700              'Parameter 5 erhält den
                         'Wert 700.
PAR_72 = ADC(1)          'Die Spannung am analogen
                         'Eingang 1 wird gemessen
                         'und im Parameter 72
                         'abgelegt.
```

! Achtung: Im Gegensatz zu den sonstigen Variablen *dürfen* die globalen skalaren Variablen PAR_x bzw. FPAR_x *nicht* deklariert werden, da sie dem ***ADbasic***-Compiler bereits bekannt sind.

Zusätzlich zu den skalaren Variablen stehen Ihnen für den Datenaustausch mit anderen Prozessen auf dem ***ADwin***-System oder dem PC, Datensätze mit dem Namen DATA zur Verfügung, die es ermöglichen, große Datenmengen auszutauschen.

Hinweis: Da Größe und Datentyp wählbar sind, müssen Sie Datensätze (DATA-Arrays) am Anfang Ihres Programms definieren.

Die Namen für DATA-Arrays sind:

```
DATA_1, DATA_2, ..., DATA_200.
```

Andere Namen sind unzulässig. Sie müssen jedoch keine fortlaufenden Nummern verwenden, auch die Deklaration von z. B. DATA_5 (ohne DATA_1 bis DATA_4) ist gültig. In Ihrem Programm werden die Datensätze vom Compiler anhand ihrer Nummer unterschieden.

Beispiel:

```
DIM DATA_5[20000] AS LONG     'Deklariert den
                              'Datensatz 5 mit
                              '20000 Elementen vom
                              'Typ LONG.
```

Die maximale Größe der Datensätze richtet sich nur nach dem verfügbaren Speicherplatz. Auf einem ***ADwin***-System mit 4 MB Speicher kann ein Datensatz mit 1 Million Long-Elementen deklariert werden.

Nachdem der Datensatz deklariert ist, können Sie auf jedes einzelne Element zugreifen. Das erste Element des Datensatzes besitzt den Index 1.

Beispiele:

```
PAR_1 = DATA_5[200]       'Der globalen Variablen 1
                          'wird der Wert des 200.
                          'Elements aus dem
                          'Datensatz 5 zugewiesen.
DATA_5[345] = 4000        'Durch diese Anweisung
                          'erhält das 345. Element
                          'aus dem Datensatz 5 den
                          'Wert 4000.
```

Sie können die Nummer des Elements auch über eine Variable übergeben:

```
nummer1 = 345
DATA_5[nummer1] = 4000    'Auch hier wird, wie im
                          'Beispiel davor, dem 345.
                          'Element des Datensatzes 5
                          'der Wert 4000 zugewiesen.
```

! **Achtung:** Die Nummer des Datensatzes darf *nicht* durch eine *Variable übergeben* werden. Die folgende Anweisung führt zu einer Fehlermeldung des ***ADbasic***-Compilers!

Beispiel:

```
index = 2
DATA_index[300] = 20
```

FALSCH !!

Anstelle von `index` muß eine konstante Zahl stehen.

4.4.3 Selbstdefinierbare Variablen

Alle Variablen, die Sie für Ihren Prozeß benötigen, müssen zu Beginn des ***ADbasic***-Programms deklariert werden. Die beiden Prozessoren T225 und T400 verarbeiten ausschließlich ganzzahlige Werte. Der T805, der T450 und der ADSP erlauben zusätzlich Fließkomma-Variablen.

Da der T450 im Gegensatz zum T805 und zum ADSP keine integrierte Floating Point Unit (FPU) besitzt, werden alle Fließkomma-Operationen für den T450 von ***ADbasic*** emuliert. Der dadurch bedingte zusätzliche Rechenaufwand sollte durch den möglichst sparsamen Einsatz von Fließkomma-Variablen, bei Verwendung des T450, minimiert werden.

Variablentyp	Prozessor			
	T400	**T805**	**T450**	**ADSP**
SHORT	-	-	-	-
INTEGER	32 Bit	32 Bit	32 Bit	32 Bit
LONG	32 Bit	32 Bit	32 Bit	32 Bit
FLOAT	-	32 Bit	32 Bit	32 Bit

Tabelle 4-2: Prozessoren und verfügbare Variablentypen

Beispiele:

```
DIM wert AS INTEGER      'Definiert die Variable
                         'wert mit dem Datentyp
                         'INTEGER

DIM wert1, wert2 AS INTEGER   'Definiert die
                              'Variablen wert1 und
                              'wert2 mit dem
                              'Datentyp INTEGER
```

Wenn Sie zwei Variablen mit Operationen verknüpfen, achtet ***ADbasic*** auf die Datentypen und konvertiert dabei die Variablen so,

daß sie miteinander verknüpfbar sind. Weitere Informationen hierzu finden sie im Kapitel „Typkonvertierung“.

Variablen können Sie nicht nur als skalare Größe, sondern auch als Array deklarieren, das heißt, Sie können Felder von Variablen erzeugen und verarbeiten. Die Anzahl der Felder im Array wird in eckigen Klammern nach dem Namen eingegeben.

Beispiel:

```
DIM wert[100] AS LONG    'Definiert ein Array der
                         'Länge 100 mit Namen wert
                         'und dem Datentyp LONG
```

! Achtung: Alle Variablen, die Sie in Ihrem ***ADbasic***-Programm verwenden, müssen *vor* dem Beginn des ersten Programmabschnitts deklariert werden.

4.4.4 Die Datenstruktur FIFO

Für Anwendungen, in denen große Datenmengen kontinuierlich übertragen werden sollen, gibt es eine Datenstruktur, die als FIFO (**F**irst **I**n, **F**irst **O**ut) organisiert ist. Alle hineingeschriebenen Werte werden in einer Warteschlange verwaltet. Sie werden vom PC oder einem anderen ***ADbasic***-Prozeß in der gleichen Reihenfolge wieder ausgelesen, in der sie geschrieben wurden. Die Größe des FIFO müssen Sie in der Deklaration festlegen. Neben der Löschfunktion besteht auch die Möglichkeit, den belegten und den noch freien Speicherplatz abzufragen.

! Achtung: Da ***ADbasic*** den FIFO intern wie einen Datensatz verwaltet, darf dieselbe Datensatznummer *nicht gleichzeitig* als Nummer eines FIFO *und* einer normalen DATA-Variablen benutzt werden.

! Achtung: Bei ***ADwin***-Systemen mit ADSP21062 darf die Datenstruktur FIFO nicht unter dem Datentyp Short deklariert werden.

Die Deklaration des FIFO ist ähnlich wie bei der DATA-Struktur:

```
DIM DATA_1[1000] AS INTEGER AS FIFO
```

Diese Anweisung deklariert ein Array mit der Datensatznummer 1 und der Länge von 1000 Integerwerten als FIFO-Ringspeicher.

Auf den FIFO können Sie zugreifen indem Sie seine Nummer angeben; der FIFO schreibt den übergebenen Wert automatisch an die richtige Stelle, bzw. liest ihn in der richtigen Reihenfolge aus.

Beispiel:

```
DATA_1 = 95                 'Schreibt in den FIFO mit
                            'der Nummer 1 den Wert 95.

PAR_7 = DATA_1              'Liest einen Wert aus dem
                            'FIFO und speichert ihn in
                            'der globalen Variablen
                            'PAR_7
```

Um sicherzustellen, daß noch Platz im FIFO ist, sollten Sie vor dem Schreiben die Funktion FIFO_EMPTY verwenden. In gleicher Weise prüft die Funktion FIFO_FULL vor dem Lesen, ob noch nicht gelesene Werte vorhanden sind.

Beispiel:

```
frei = FIFO_EMPTY(1)
IF (frei > 0) THEN
  DATA_1 = wert1
ENDIF

belegt = FIFO_FULL(1)
IF (belegt > 0) THEN
  PAR_7 = DATA_1
ENDIF
```

Mit FIFO_CLEAR (*DatensatzNr*) kann der FIFO DatensatzNr gelöscht werden.

Hinweis: Da die FIFOs beim Start nicht automatisch gelöscht werden, sollten sie mit diesem Befehl im Abschnitt INIT: des Programms gelöscht werden.

! Achtung: Wenn Sie schneller Daten in den FIFO schreiben, als Sie Daten auslesen, ist der FIFO irgendwann voll und es gehen Daten verloren.

4.4.5 Status-Variablen

Um Informationen über den Status des ***ADwin***-Systems zu erhalten, stehen die folgenden Status-Variablen zur Verfügung:

PROZESS_RUNNING

Zeigt den Status des Prozesses an. Der Wert ist 1, solange der Prozeß läuft.

GLOBALSCHLEIFE[5]

Anzahl der Durchläufe, die der Prozeß durchführen soll. Diese Status-Variable kann unter **Options ⇨ Process** in dem Feld 'Number of Loops' vorgegeben werden. Diese Variable wird sowohl durch einen internen, als auch durch einen externen Event dekrementiert.

ANZAHLSCHLEIFE[5]

Beinhaltet die Anzahl der Schleifendurchläufe, die noch auszuführen sind. Diese Variable existiert nur, wenn beim Kompilieren die Anzahl der Durchläufe größer als Null war.

[5] Die Status-Variablen GLOBALSCHLEIFE und ANZAHLSCHLEIFE stehen beim ADSP nicht zur Verfügung.

GLOBALDELAY

Zeitabstand in Mikrosekunden zwischen zwei Events. Diese Status-Variable kann unter **Options ⇨ Parameter** in dem Feld 'Delay in µs' vorgegeben werden.

NWTIME

Stand des internen Timers beim Starten des letzten Timer-Events.

! Achtung: Sie sollten diese Variablen *innerhalb* eines ***ADbasic***-Programms nur lesen, niemals mit neuen Werten überschreiben, da hierdurch das ***ADwin***-System in einen instabilen Zustand gelangen kann.

4.4.6 Typkonvertierung

Wenn Sie zwei Variablen mit Operationen verknüpfen, achtet ***ADbasic*** auf die Datentypen und konvertiert dabei die Variablen so, daß sie miteinander verknüpfbar sind.

Falls bei Berechnungen in einer Zeile ein Wert oder eine Variable vom Typ Float ist, werden in der Regel alle Werte und Variablen in dieser Zeile vor der Berechnung in Floating-Point-Zahlen konvertiert. Gegebenenfalls wird das Ergebnis am Ende der Berechnung wieder zu einer Integer-Variablen gewandelt, wobei die Nachkommastellen vernachlässigt (d. h. abgeschnitten) werden.

! Achtung: Da LONG und FLOAT Datentypen unterschiedliche Wertebereiche haben, kommt es bei der FLOAT Konvertierung sehr großer LONG- Zahlen zu Genauigkeitsverlusten. Die Abweichung kann bis zu 64 betragen (Die LONG- Zahl 2000000064 wird nach der Konvertierung in den FLOAT- Datentyp zu 2000000000).

Beispiel:

```
PAR_1 = 2000000001
PAR_2 = 2000000002

FPAR_3 = (PAR_2 - PAR_1) + 0.5
             ' FPAR_3 ist 0.5 , da PAR_1 und PAR_2
             '  vor der Subtraktion in FLOAT gewan-
             '  delt werden

PAR_9  =  PAR_2 - PAR_1
FPAR_4 =     PAR_9     + 0.5
             ' FPAR_4 ist 1.5, da die Subtraktion
             '  mit LONG- Zahlen ausgeführt wurde
```

! Achtung: Auch das Setzen von Klammern verhindert nicht die automatische Typumwandlung in FLOAT. Sollen Berechnungen in LONG durchgeführt werden, so ist hierfür eine eigene Zeile zu programmieren (siehe PAR_9 im obigen Beispiel).

Eine Ausnahme von der o. a. Regel stellen die Anweisungen `IF ... THEN` und `DO ... UNTIL` dar. Hier wird nicht die gesammte Zeile in FLOAT umgewandelt, sondern nur Teilbereiche der Zeile. Die logischen Operatoren `AND` und `OR` sowie das Wort `THEN` grenzen die einzelnen Teilbereiche voneinader ab.

Beispiel:

```
PAR_1  = 2000000001
PAR_2  = 2000000002
FPAR_2 = 5.5

IF ((PAR_1 > 2000000000) AND (FPAR_2 * 1.1 > 5.5)) THEN
    PAR_14 = 1
ENDIF
                    ' Die IF- Bedingung ist erfüllt,
                    '  weil PAR_1 nicht in FLOAT
                    '  gewandelt wird

IF (FPAR_2 * 1.1 > 5.5) THEN PAR_3 = PAR_2 - PAR_1

                    ' Ergebnis :  PAR_3 = 1
                    '  d.h. es wird mit LONG- Zahlen sub-
                    '  trahiert, nicht mit FLOAT- Zahlen
```

4.5 Mehrere ADbasic-Prozesse

Auf jedem Prozessor eines ***ADwin***-Systems können bis zu zehn ***ADbasic***-Prozesse gleichzeitig ablaufen. Eine Ausnahme stellt der Prozessor T225 dar: auf ihm können nur zwei ***ADbasic***-Prozesse gleichzeitig ablaufen. Außerdem stehen auf jedem Prozessor eine Reihe von Standard-I/O und -Regelprozessen zur Verfügung, die ebenfalls gleichzeitig ablaufen können. Zusätzlich läuft ständig der Kommunikations-Prozeß mit niedriger Priorität im Hintergrund. Dieser regelt den Datentransfer zwischen PC und ***ADwin***-System. In Abb. 4-2 sind die verschiedenen Prozesse schematisch dargestellt.

Hinweis: PID-Reglerprozesse stehen beim ADSP nicht zur Verfügung.

<table>
<tr><td colspan="7">Kommunikations-Prozeß</td></tr>
<tr><td>Zyklischer AD-Prozeß 1-4</td><td>Zyklischer DA-Prozeß 1-2</td><td>PID-Regler-Prozeß 1-4</td><td>ADbasic Prozeß 1</td><td>ADbasic Prozeß 2</td><td>ADbasic Prozeß ...</td><td>ADbasic Prozeß 10</td></tr>
<tr><td colspan="7">Globale Variablen
PAR_1, PAR_2, ..., PAR_80
FPAR_1, FPAR_2, ..., FPAR_80
DATA_1, DATA_2, ..., DATA_200</td></tr>
</table>

Abb. 4-2: ***ADwin***-Prozesse

Jedem Programm, das auf dem System ablaufen soll, müssen Sie über das Menü **Options ⇨ Process Options** eine eigene Prozeßnummer zwischen 1 und 10 zuweisen.

! Achtung: Wenn Sie zwei Prozesse mit derselben Prozeß-Nummer auf das System laden, wird der zuerst geladene Prozeß überschrieben!

4.5.1 Datenaustausch

Ebenso wie der Datenaustausch zwischen ***ADwin***-System und PC wird auch der Datenaustausch zwischen verschiedenen Prozessen auf dem ***ADwin***-System durch den Einsatz der globalen Variablen (`PAR_1 ... PAR_80`) oder der globalen Datenstrukturen (`DATA`) ermöglicht. Auf diese können sämtliche Prozesse zugreifen. Ihre Bezeichnung ist für alle Prozesse identisch.

Die globalen Variablen können auch verwendet werden, um aus einem Prozeß heraus einen anderen, gleichzeitig laufenden Prozeß zu steuern.

Beispiel:

Der ***ADbasic***-Prozeß 1 nimmt kontinuierlich Meßwerte auf und legt sie in dem als FIFO deklarierten globalen Datensatz 1 (`DATA_1`) ab. Ein zweiter gleichzeitig laufender ***ADbasic***-Prozeß fragt in regelmäßigen Abständen mit der Funktion `FIFO_FULL(1)` die Anzahl der im FIFO befindlichen Elemente ab. Wenn sich eine vorgegebene Anzahl von Elementen im FIFO befindet, beginnt der zweite ***ADbasic***-Prozeß mit der Auswertung der Meßwerte (z. B. FFT, Mittelwertbildung, Beschleunigung etc.). Der ***ADbasic***-Prozeß 1 kann währenddessen ungehindert mit der kontinuierlichen Meßwertaufnahme fortfahren. Aufgrund der FIFO-Struktur kann der zweite ***ADbasic***-Prozeß die Meßwerte in der gleichen Reihenfolge auslesen, in der sie vom Prozeß 1 abgelegt wurden. Prozeß 2 muß lediglich auf den gleichen globalen Datensatz zugreifen (hier `DATA_1`), der auch von Prozeß 1 zum Abspeichern der Meßdaten verwendet wurde.

! Achtung: Soll auf einen Datensatz in zwei Prozessen zugegriffen werden können, dann muß dieser Datensatz in *beiden* ***ADbasic***-Prozessen in *gleicher* Weise deklariert werden.

4.5.2 Auslastung

Wenn mehrere Prozesse gleichzeitig auf dem ***ADwin***-System ablaufen, müssen sich diese die Rechenzeit des Prozessors teilen. Die gesamte Auslastung des ***ADwin***-Systems setzt sich dann aus der Summe der durch jeden einzelnen Prozeß verursachten Auslastung zusammen. Sie können die Auslastung des Prozessors an der 'Busy'-Anzeige im Parameter-Fenster (Menü **Options ⇨ Parameter**) beobachten.

4.5.3 Mehrere hoch priorisierte Prozesse

Auf einem ***ADwin***-System können auch gleichzeitig mehrere hoch priorisierte Prozesse ablaufen. Da hoch priorisierte Prozesse nicht unterbrochen werden können, ändert sich am internen Zeitverhalten der einzelnen Prozesse nichts.

Unterschiede ergeben sich jedoch für die Zeitspanne, die zwischen dem prozeßaufrufenden Event und dem tatsächlichen Bearbeitungsbeginn vergeht: Soll ein hoch priorisierter Prozeß aufgerufen werden, während ein zweiter hoch priorisierter Prozeß gerade bearbeitet wird, dann muß der neu aufgerufene Prozeß solange warten, bis die Bearbeitung des bereits laufenden Prozesses abgeschlossen ist. Hierdurch ergeben sich Verzögerungen für den Bearbeitungsbeginn.

Hinweis: Beachten Sie bitte, daß sich die maximale Verzögerung der Ausführung eines Prozesses aus der Summe der Ausführungszeiten aller anderen gleichzeitig mit hoher Priorität auf dem ***ADwin***-System ablaufenden Prozesse ergibt. Wenn mehrere hoch priorisierte Prozesse gleichzeitig auf einem ***ADwin***-System laufen, dann sollten Sie darauf achten, daß die Ausführungszeit der einzelnen Prozesse möglichst kurz ist.

4.6 Unterprogramme und Funktionen

Um Ihr Programm besser strukturieren zu können, haben Sie in ***ADbasic*** die Möglichkeit, Unterprogramme und Funktionen zu verwenden. Die Syntax ist sehr einfach, lediglich die Begriffe `SUB ... ENDSUB` bzw. `FUNCTION ... ENDFUNCTION` umgeben die jeweiligen Prozeduren wie eine Klammer. Beim Aufruf können auch Werte von Variablen übergeben werden, nicht jedoch Variable selbst. Für den Fall, daß Sie innerhalb einer Funktion oder eines Unterprogramms Variableninhalte global verändern möchten, verwenden Sie bitte die globalen Variablen `PAR_1` bis `PAR_80`.

Sie können auch eine „Sammlung“ von Unterprogrammen oder Funktionen erstellen und in einer separaten Datei speichern. Diese Datei läßt sich dann sehr einfach mit einem `INCLUDE`-Befehl in Ihr aktuelles Programm einbinden. Der `INCLUDE`-Befehl muß allerdings ganz am Anfang Ihres Programms stehen, Funktionen und Unterprogramme können auch vor dem `INIT`-Block oder nach dem `FINISH`-Abschnitt am Ende Ihres Programms stehen.

Hinweis: Eine Erklärung und ein Beispiel der Befehle zum Aufrufen von Unterprogrammen bzw. Funktionen finden Sie in dem Kapitel Befehlsreferenz.

4.7 Auswerten und Visualisieren von Meßdaten

Um die Daten anzeigen und auswerten zu können, benötigen Sie ein Meßdatenauswerteprogramm wie etwa *TestPoint* oder *MATLAB*. Selbstverständlich können Sie sich auch ein eigenes Auswerteprogramm unter Benutzung von *Visual Basic, Visual C, Delphi, Excel* etc. gestalten. Für jede dieser Möglichkeiten ist Treibersoftware erhältlich, die ***ADbasic*** und ***ADwin*** mit dem gewünschten Programm verbindet. Das Zusammenspiel und die Kommunikation des ***ADwin***-Systems mit anderen Programmen wird in den Unterlagen zu den Treibern, die für diese Programme erhältlich sind, genau erläutert. Dort finden Sie auch Informationen darüber, wie Sie mit ***ADbasic*** erzeugte und kompilierte Programme auf ein ***ADwin***-System laden und starten können. Da das Meßprogramm auf dem ***ADwin***-System immer gleich ist, können auch mehrere verschiedene Anwendungsprogramme gleichzeitig auf das ***ADwin***-System zugreifen.

Wenn Sie spezielle Wünsche haben, fragen Sie bei uns an.

5 Optimieren des Zeitverhaltens

5.1 Priorität und Zeitverhalten

Die Prozessoren der ***ADwin***-Systeme verfügen über eine eigene Prozeßverwaltung. Für die Abarbeitung der einzelnen Prozesse werden zwei Prioritätsebenen unterschieden. Jedem Prozeß kann die Prioritätsstufe High oder Low zugewiesen werden.

Diese Fähigkeit der Prozessoren ist auch unter ***ADbasic*** nutzbar. Sie können die Priorität jedes einzelnen Prozesses im Menü **Options ⇨ Process Options** einstellen.

Priorität High

Bei den Prozessoren T225, T400, T450 und T805 ist für *einen* hoch priorisierten Prozeß sichergestellt, daß vom Aufruf dieses Prozesses bis zu dessen Bearbeitungsbeginn maximal 2,5 µs vergehen. Bei dem ADSP21062 vergehen bei dieser Art des Prozesses maximal 300 ns. Die Befehle des hoch priorisierten Prozesses werden komplett abgearbeitet, bevor der Prozessor des ***ADwin***-Systems wieder für andere Prozesse zur Verfügung steht. Dadurch ist ein konstantes und exakt vorhersagbares Zeitverhalten garantiert. Der hoch priorisierte Prozeß ermöglicht somit sichere Reaktions- und Antwortzeiten im Mikrosekundenbereich.

Der Abstand (Delay) zwischen zwei vom internen Timer des ***ADwin***-Systems erzeugten Events kann für die ***ADwin***-Systeme mit T225-, T400-, T450- oder T805-Prozessor bei einem hoch priorisierten Prozeß mit einer Auflösung von 1 µs vorgegeben werden. Bei ***ADwin***-Systemen mit ADSP21062 ist die Auflösung 25 ns.

Beispiel:

Wird bei einem ***ADwin***-System mit T805-Prozessor der Wert für ‘Delay’ auf den Wert 100 gesetzt, dann liegt zwischen zwei aufeinanderfolgenden, vom internen Timer gesteuerten Aufrufen desselben Prozesses eine Zeitspanne von 100 µs.

Da ein hoch priorisierter Prozeß nicht unterbrochen werden kann, ist darauf zu achten, daß die Bearbeitungszeit des Prozesses selbst die Delay-Zeit (in dem vorangegangenen Beispiel 100 µs) deutlich

unterschreitet. Anderenfalls steht dem Prozessor des ***ADwin***-Systems keine Zeit mehr zur Verfügung, um andere gleichzeitig laufende Prozesse, wie z. B. den Kommunikations-Prozeß, zu bedienen.

! Achtung: Die Ausführungszeit von hoch priorisierten Prozessen sollte so gering wie möglich gehalten werden. Zeitintensive Schleifen oder Berechnungen, deren Ergebnis nicht sofort weiterverarbeitet wird, sollten immer mit niedriger Priorität ablaufen.

Der Abschnitt FINISH: eines jeden ***ADbasic***-Prozesses wird immer mit niedriger Priorität ausgeführt. Wenn ein Prozeß mit hoher Priorität gestartet wurde, wird die Priorität beim Erreichen des Abschnitts FINISH: automatisch herabgesetzt.

Priorität Low

Ein Prozeß, der mit niedriger Priorität abläuft, kann jederzeit von einem hoch priorisierten Prozeß unterbrochen werden. Somit können gleichzeitig mehrere Prozesse mit niedriger Priorität auf dem ***ADwin***-System laufen, ohne daß dadurch das Zeitverhalten eines hoch priorisierten Prozesses beeinflußt wird.

Allerdings kann das ***ADwin***-System nur dann niedrig priorisierte Prozesse bearbeiten, wenn ihre Rechenleistung nicht gleichzeitig von einem hoch priorisierten Prozeß vollständig beansprucht wird.

! Achtung: Zeitkritische Meßprozesse können von anderen, gleichzeitig auf einem ***ADwin***-System laufenden Prozessen nicht gestört werden, wenn der zeitkritische Meßprozeß mit hoher und alle anderen Prozesse mit niedriger Priorität ablaufen.

Der Abstand (Delay) zwischen zwei vom internen Timer des ***ADwin***-Systems erzeugten Events kann für die ***ADwin***-Systeme mit T225-, T400-, T450- oder T805-Prozessor bei einem niedrig priorisierten

Prozeß mit einer Auflösung von 64 µs vorgegeben werden. Bei ***ADwin***-Systemen mit ADSP21062 ist die Auflösung 100 µs.

Beispiel:

Wird bei einem ***ADwin***-System mit ADSP21062 der Wert für 'Delay' auf den Wert 5 gesetzt, dann liegt zwischen zwei aufeinanderfolgenden, vom internen Timer gesteuerten Aufrufen desselben Prozesses eine Zeitspanne von 500 µs.

! Achtung: Bei der Umsetzung von bestehenden ***ADbasic***-Prozessen für den ADSP21062 muß die unterschiedliche Timerauflösung beachtet werden: Um bei einem hochpriorisierten timergesteuerten Prozess für den ADSP das gleiche Delay zu erhalten wie z. B. für den T805, muß die Delay-Einstellung des T805 mit dem Faktor 40 multipliziert werden.

Delay T225, T400, T450, T805 [1µs]	Delay ADSP21062 [25ns]
1000	40000
500	20000
125	5000

Tabelle 5-1: Beispiele für die Anpassung des Delays bei der Umsetzung von bestehenden *ADbasic*-Prozessen für den ADSP21062.

5.2 Ermitteln der Ablaufzeiten mit Timerfunktionen

Der Prozessor des ***ADwin***-Systems verfügt über zwei interne Timer, die als Zähler realisiert sind.

Der erste Timer wird bei ***ADwin***-Systemen mit T225-, T400-, T450- oder T805-Prozessor jede Mikrosekunde um den Wert 1 erhöht und wird in hoch priorisierten Prozessen ausgelesen. Der zweite Timer wird nur alle 64 µs um den Wert 1 erhöht und bei niedrig priorisierten Prozessen ausgelesen.

Bei ***ADwin***-Systemen mit ADSP21062 wird der erste Timer alle 25 ns um den Wert 1 erhöht und in hoch priorisierten Prozessen ausgelesen. Der zweite Timer wird nur alle 100 µs um den Wert 1 erhöht und bei niedrig priorisierten Prozessen ausgelesen.

Nach dem Einschalten der Spannungsversorgung für das ***ADwin***-System werden beide Zähler auf den Wert 0 gesetzt. Anschließend werden sie in den oben angegebenen Zeittakten kontinuierlich inkrementiert.

Mit der ***ADbasic***-Funktion `READ_TIMER()` kann der aktuelle Zählerstand eines Timers ermittelt werden.

Hinweis: Bitte beachten Sie, daß beim T225 die Timerregister nur 16 Bit breit sind. Weitere Informationen hierzu finden Sie in dem Kapitel Befehlsreferenz unter dem Befehl `READ_TIMER`.

Hinweis: Alle ***ADbasic***-Funktionen, die auf den Prozessortimer zugreifen (`READ_TIMER()`, `NWTIME`, ...), liefern bei Verwendung des ADSP den Timerstand in 25 ns zurück.

Mit `READ_TIMER()` kann z. B. die Zeitdifferenz zwischen zwei extern getriggerten Events leicht ermittelt werden. In Verbindung mit dem Parameter `NWTIME` läßt sich auch die Latenzzeit ermitteln, also die Zeit zwischen Aufruf und dem Beginn des Prozesses. Der Parameter `NWTIME` enthält den Timerstand, bei dem der aktuelle Prozeß aufgerufen werden sollte. Liest man mit dem ersten Befehl des Prozesses den aktuellen Timerstand aus und vergleicht ihn mit dem Wert des Parameters `NWTIME`, so erhält man die Latenzzeit. Das folgende Beispiel zeigt die Vorgehensweise.

Beispiel:

```
DIM zeit, latenz, max_latenz AS INTEGER

EVENT:
zeit = READ_TIMER()
latenz = NWTIME - zeit
IF (latenz > max_latenz) THEN
  max_latenz = latenz
ENDIF
```

5.3 Schnellere Meßfunktion

Mit dem ADC-Befehl wird *eine* A/D-Wandlung für *einen* Kanal mit bestimmter Verstärkung vorgenommen. Der Befehl ist sehr einfach gehalten, um Ihnen die Anwendung zu erleichtern. Allerdings wird dabei nicht ausgenutzt, daß sich zwei ADCs auf der ***ADwin***-Karte befinden, mit denen *gleichzeitig zwei verschiedene* Kanäle konvertiert werden können.

Hinweis: Auf der ***ADwin-light*** PC-Einsteckkarte befindet sich nur *ein* ADC.

Um auf der ***ADwin***-Karte beide ADCs simultan zu benutzen, ersetzen Sie den ADC-Befehl durch die im folgenden Beispiel angegebenen Befehle. Mit diesen Befehlen haben Sie die Möglichkeit, die Geschwindigkeit des Prozesses zu optimieren.

Beispiel:

```
SET_MUX(0)                  'Setzen der Multiplexer
                            'auf die Kanäle 1 der ADCs
    Warten auf das Einschwingen der Multiplexer
START_CONV(3)               'Konvertierung beider ADCs
                            'starten
WAIT_EOC(3)                 'Ende der Konvertierung
                            'abwarten
ad1 = READADC(1)            'Auslesen von ADC1
ad2 = READADC(2)            'Auslesen von ADC2
```

Hinweis: Die in dem Beispiel grau hinterlegten Flächen sind Wartezeiten, die nötig sind, um das Einschwingen des Multiplexers und die Konvertierung der ADCs abzuwarten. Beispielsweise besteht die Möglichkeit, direkt nach dem Befehl START_CONV(3) die Multiplexer für die nächste Messung umzustellen. Zum Beispiel braucht beim ***ADwin-GOLD*** das Einschwingen des Multiplexers 6,5 µs (bei 16 Bit Genauigkeit). Für die Konvertierung werden weitere 7 µs benötigt - zusammen also 13,5 µs, die Sie anderweitig nutzen können. Weitere Zeiten für andere ***ADwin-***Systeme (und andere Genauigkeiten) entnehmen Sie bitte dem entsprechenden Hardware-Handbuch oder den Ausführungen zu den Befehlen ADC und ADC12 in der Befehlsreferenz.

! Achtung: Die notwendigen Zeiten für das Einschwingen des Multiplexers und für die Konvertierung der ADCs müssen unbedingt eingehalten werden, da die A/D-Wandlung sonst nicht funktioniert und *falsche Ergebnisse* liefert.

6 Menüs und Dialogfenster

Wenn Sie ***ADbasic*** aus Ihrer *MS-Windows*-Oberfläche gestartet haben, erscheint die Entwicklungsoberfläche von ***ADbasic***, die Sie in Abb. 6-1 sehen können. Die ***ADbasic***-Oberfläche ist wie alle anderen *MS-Windows*-Oberflächen aufgebaut, um Ihnen die Bedienung zu erleichtern.

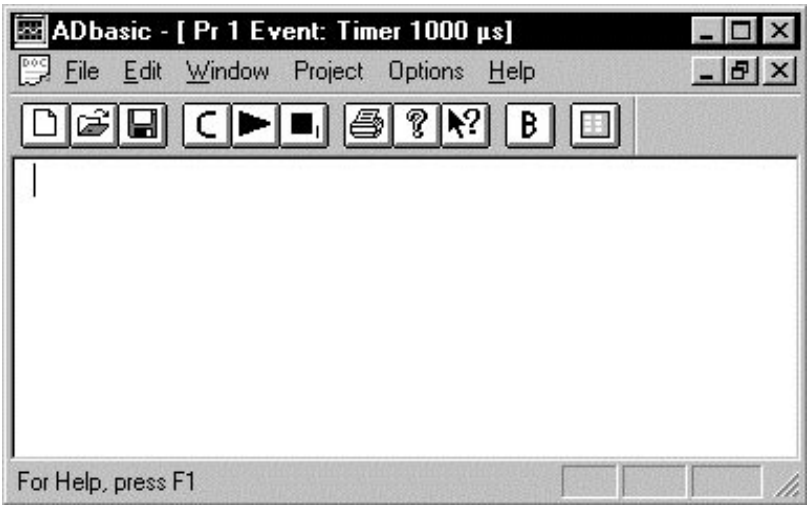


Abb. 6-1: Die *ADbasic*-Entwicklungsoberfläche

Die ***ADbasic***-Oberfläche besteht aus der Menüleiste, der Toolbar (auch als Symbolleiste oder Werkzeugkiste bezeichnet) und dem Editor-Fenster.

Aus der Menüleiste wählen Sie das gewünschte Menü aus, indem Sie entweder mit dem Maus-Zeiger auf das entsprechende Feld gehen, und es durch das Drücken der linken Maustaste aktivieren, oder indem Sie die Tastenkombination [ALT] + [ANFANGSBUCHSTABE] des entsprechenden Menüs eingeben.

Mit der Toolbar haben Sie die Möglichkeit, auf häufig benutzte Befehle schnell zuzugreifen. Jede Schaltfläche entspricht einem Befehl in der Menüleiste.

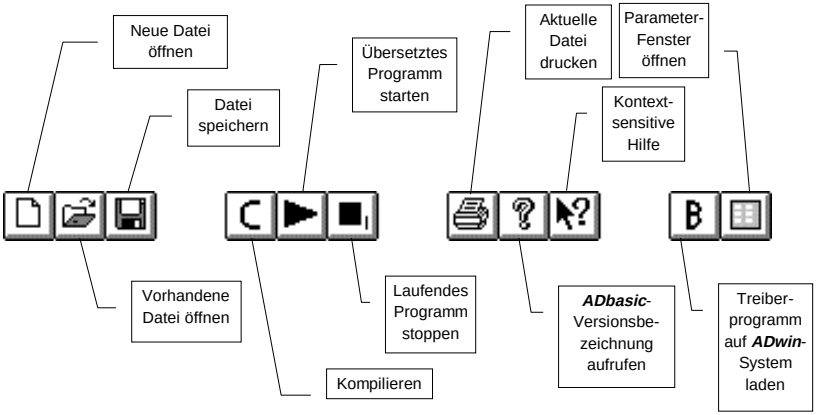


Abb. 6-2: Die Toolbar

6.1 Das Menü File

Mit dem Menü **File** können Sie wie bei *Windows* üblich Dateien laden und speichern, sowie neue Dateien (und damit Editor-Fenster) erzeugen. Sie können beliebig viele Editor-Fenster erzeugen, aber nur maximal zehn Programme/Prozesse gleichzeitig auf Ihr ***ADwin***-System laden.

In diesem Menü sind auch die Druckfunktionen (Drucken, Druckvorschau und Druckereinrichtung) enthalten.

In diesem Menü wird auch eine Liste der zuletzt geöffneten Dateien angezeigt. Die Größe der Liste ist auf vier Dateien begrenzt.

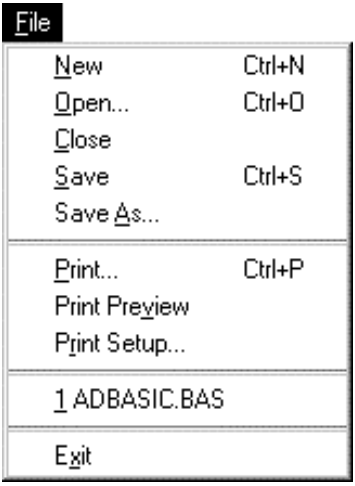


Abb. 6-3: Das Menü File

6.2 Das Menü Edit

Das Menü **Edit** entspricht den *Windows*-Konventionen. Es enthält auch eine Such-Funktion (Find) sowie eine Ersetzen-Funktion (Replace).

Hinweis: Die Rückgängig-Funktion (Undo) ist nur sinnvoll für die Kopier- (Copy), Ausschneide- (Cut) oder Einfüge- (Paste) Funktion einsetzbar.

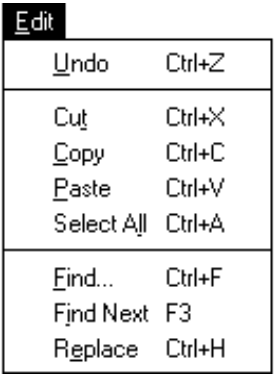


Abb. 6-4: Das Menü Edit

6.3 Das Menü Window

Mit dem Menü **Window** können Sie zwischen verschiedenen Editor-Fenstern umschalten bzw. diese am Bildschirm arrangieren.

Außerdem haben Sie die Möglichkeit die Toolbar (Symbolleiste) und die Status Bar (Statusleiste) ein- und auszuschalten.

Zusätzlich finden Sie hier Informationen über die geöffneten Programme: Name, Art des Events und Delay-Wert.

Beispiel:

In Abb. 6-5 sind zwei Programme geöffnet. Der Name des aktiven Programms ist ADBASIC.BAS; die Prozeßnummer des aktiven Programms ist 1; der Event wird von einem internen Timer erzeugt; der Abstand zwischen zwei Events beträgt 1000 µs.

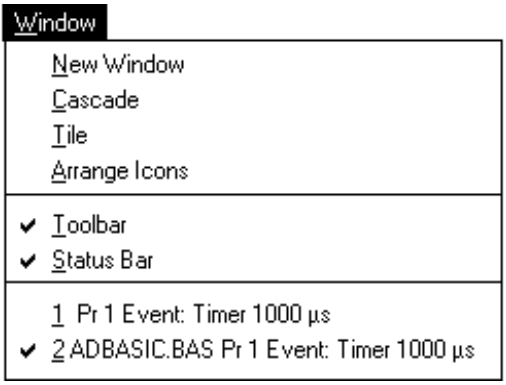


Abb. 6-5: Das Menü Window

6.4 Das Menü Project

Mit dem Menü **Project** übersetzen, starten oder stoppen Sie Ihre Programme.

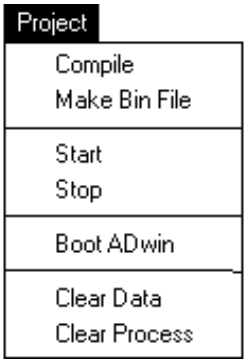


Abb. 6-6: Das Menü Project

Compile übersetzt Ihr aktuelles Programm und überträgt es zur Ausführung auf den Prozessor des ***ADwin***-Systems. Falls Sie im Menü **Options ⇨ Compiler** 'Autostart' auf 'Yes' gesetzt haben, wird Ihr Prozeß sofort nach der Übertragung gestartet. Im anderen Fall können Sie ihn mit dem Menüpunkt 'Start' ausführen lassen.

Verwenden Sie den Menüpunkt 'Stop', um den aktuellen Prozeß (aktives Fenster) zu stoppen.

Make Bin File speichert den aktuellen Prozeß in kompilierter Form (als Binärdateien) ab. Die Dateinamenerweiterung wird dabei vom Programm automatisch vergegeben. Der Buchstabe „T" ist dabei immer Bestandteil der Dateinamenerweiterung. Für ***ADwin***-Systeme mit dem Prozessor ADSP21062 folgt auf das „T" eine „9" . Für ***ADwin***-Systeme mit dem Prozessor T 805 folgt eine „8", eine „5" für den T450, eine „4" für den T400 und eine 2 für den T225. Die zweite Ziffer zeigt die Prozeßnummer, die Sie im Menü **Options ⇨ Process Options** eingestellt haben, an.

Beispiel:

Die Dateinamenerweiterung *.T81 bedeutet, daß der Prozessor T805 benutzt wird und es sich um den Prozeß mit der Prozeßnummer 1 handelt.

Wenn Sie den entsprechenden ***ADwin***-Treiber auf Ihr ***ADwin***-System geladen haben, dann können Sie mit ***ADbasic*** erzeugte Binärdateien auch ohne die ***ADbasic***-Entwicklungsumgebung auf Ihr ***ADwin***-System laden und starten. Lesen Sie dazu bitte in der von uns zur Verfügung gestellten Treiberdokumentation zu Ihrem Programm zur Visualisierung der Meßwerte nach.

Boot ADwin lädt das Treiberprogramm erneut auf das ***ADwin***-System. Dadurch werden alle laufenden Prozesse gelöscht und alle globalen Variablen auf den Wert 0 gesetzt. Der Wert für das Globaldelay ist nach dem Laden des ***ADwin***-Treibes beim ADSP auf 25000 ns und bei allen anderen Prozessoren auf 1000 µs voreingestellt.

! Achtung: Wenn Sie zwei Prozesse mit gleicher Prozeß-Nummer (Menü **Options ⇨ Process Options**) auf Ihr ***ADwin***-System laden, wird der zuerst geladene Prozeß ohne Sicherheitsabfrage überschrieben.

6.5 Das Menü Options

Im Menü **Options** haben Sie die Möglichkeit, Parameter einzustellen, um Art und Ausführung Ihres Programms zu beeinflussen.

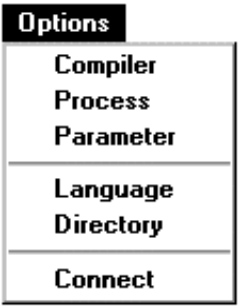


Abb. 6-7: Das Menü Options

Die Einstellungen im Compiler-, Parameter-, Language-, Directory-, und Connect-Fenster wirken auf *alle* Programmfenster.

Die Einstellungen im Process-Fenster beziehen sich *nur* auf das Programm in dem Editor-Fenster, das vor dem Aufruf des jeweiligen Menüpunktes aktiv war. Haben Sie mehrere Editor-Fenster geöffnet, kontrollieren Sie bitte vor dem Aufruf, ob das Fenster des Programms, zu dem Sie Einstellungen vornehmen wollen, aktiv im Vordergrund ist.

6.5.1 Die Compiler-Optionen

Options ⇨ Compiler

Das mit diesem Menüpunkt aufgerufene Dialogfenster dient dazu, dem Compiler Informationen über den gewünschten Zielprozessor und andere Parameter Ihres ***ADwin***-Systems zu geben. Die hier eingestellten Werte gelten für alle ***ADbasic***-Prozesse.

Abb. 6-8: Das Dialogfenster Compiler Options

'Prozessor':
Stellen Sie den Prozessor ein, den Sie in Ihrem ***ADwin***-System haben. Wenn Sie sich nicht sicher sind, welchen Prozessor Sie verwenden, dann lesen Sie bitte im Handbuch zu Ihrer ***ADwin***-Karte, Ihrer ***ADwin-Box*** bzw. Ihrem ***ADwin-Pro***-System nach.

Hinweis: Häufig werden nur die Kurzbezeichnungen der in den ***ADwin***-Systemen verwendeten Prozessoren angegeben. Tabelle 6-1 zeigt eine Gegenüberstellung der Kurz- und der Vollbezeichnungen.

Prozessor-Kurzbezeichnung	ADSP	T8	T5	T4	T2
Prozessor	ADSP 21062	T805	T450	T400	T225

Tabelle 6-1: Prozessorbezeichnungen

Da Sie Programme auch für einen anderen als den installierten Prozessor kompilieren können, wird der jeweils zuletzt eingestellte Wert beibehalten. Erst bei der Kompilierung wird die aktuelle Konfiguration überprüft. Stimmt diese nicht mit den Einstellungen

überein, können Sie zwar kompilieren, nicht jedoch das Programm auf den Prozessor des ***ADwin***-Systems laden.

'Autostart':

Hier können Sie einstellen, ob Ihr Programm nach dem Kompilieren sofort gestartet werden soll oder erst über die Start-Taste.

'Debug Mode':

Hier können Sie vorgeben, daß in das Progamm zusätzliche Sicherheitsabfragen eingebaut werden, um die folgenden Laufzeitfehler zu erkennen und im 'Status'-Feld des Parameter-Fensters anzuzeigen.

- Division durch Null
- Wurzel aus Werten kleiner Null (negative Werte)
- Zugriffe auf nicht definierte Data-Elemente
- Zugriffe auf nicht definierte Array-Elemente
- Angabe zu kurzer Wiederholraten

Hinweis: Das Einschalten des Debug-Modes kostet Rechenzeit und verlängert die Programmausführungszeit um ca. 20 %. Sie sollten daher diese Option nur während der Programmentwicklung nutzen.

'Linkaddress':

Wählen Sie hier die Linkadresse (Basisadresse für den Linkadapter) aus, die auf dem ***ADwin***-System mit den DIP-Schaltern eingestellt ist. Die Werkseinstellung ist 336 (150H). Falls Sie ohne Prozessor entwickeln möchten, wählen Sie hier 'NONE' aus.

'Memory':

Geben Sie die Größe des Speichers an, mit dem Ihr ***ADwin***-System ausgerüstet ist[6].

[6] Die Speichergröße wird beim ADSP automatisch ermittelt

6.5.2 Die Prozeß-Optionen

Options ⇨ Process

Das mit diesem Menüpunkt aufgerufene Dialogfenster dient zur Festlegung der prozeßspezifischen Parameter. Die hier eingestellten Werte gelten für das aktive Programmfenster und müssen vor dem Kompilieren sowie dem Laden des Prozesses vorgenommen werden.

Je nach verwendetem Prozessor wird das in Abb. 6-9 oder in Abb. 6-10 gezeigte Dialogfenster geöffnet

6.5.2.1 Dialogfenster Process Options für die Prozessoren T225, T400, T450 und T805

Abb. 6-9: Das Dialogfenster Process Options

‘Event’:

Hier können Sie einstellen, wie die Events erzeugt werden, die Ihr Programm starten. Sie können entweder den internen ‘Timer’ des Prozessors Ihres ***ADwin***-Systems dazu benutzen, oder Sie benutzen ein Signal (positive Flanke) am Event-Eingang an der Rückseite der ***ADwin***-Karte und stellen in dem Dialogfenster Process Options auf ‘Extern’.

Hinweis: Wie Sie bei einem ***ADwin-Pro***-System einen externen Event-Eingang nutzen können, lesen Sie bitte in der Dokumentation mit dem Titel „***ADwin-Pro*** : Systembeschreibung ; Programmierung in ***ADbasic***“ unter dem `EVENTENABLE`-Befehl nach.

In der Einstellung ‘None’ wird Ihr Programm unabhängig von einem Event sofort gestartet und nach der Ausführung des letzten Befehls wieder von neuem ausgeführt. Insbesondere bei einem hoch priorisierten Prozeß müssen Sie in der Einstellung ‘None’ dafür sorgen, daß der Prozeß auch Rechenzeit für andere Aufgaben (Kommunikation mit dem PC/ mit anderen über einen Link angeschlossenen Prozessoren) bereitstellt.

Hinweis: Nähere Informationen hierzu unter dem Befehl `LINKIN`.

‘Process’:

Dies ist die Nummer Ihres Prozesses. Wenn Sie mehrere Prozesse gleichzeitig auf einem ***ADwin***-System ablaufen lassen möchten, müssen Sie hier jedem Prozeß eine eigene Nummer zuweisen.

Hinweis: Prozeß1 läuft bei T805-Prozessoren im internen Speicher und ist daher schneller.

‘Number of Loops’:

Falls gewünscht, können Sie hier die Anzahl der Programmaufrufe durch Events Ihres Programms einstellen. Ist die eingestellte Anzahl erreicht, stoppt das Programm. Diese maximale Anzahl der Durchläufe können Sie für jeden Start des Programms erneut festlegen, ohne das Programm neu kompilieren zu müssen.

Wenn Sie den Wert 0 eintragen, wird das Programm solange wiederholt, bis...

- ...Sie den Prozeß mit einem END-Befehl im ***ADbasic***-Programm,
- mit einem STOP_PROCESS-Befehl vom PC oder von einem anderen ***ADbasic***-Prozeß aus oder
- mit der Stopp-Taste von ***ADbasic*** explizit stoppen.

'Version':

Hier können Sie einen Integer-Wert als Versionsnummer eingeben. Diese Versionsnummer wird bei einem kompilierten Programm (nur) in einem Text-Editor angezeigt.

'Priority':

Wenn Sie mehrere Prozesse gleichzeitig auf einem ***ADwin***-System ablaufen lassen möchten, können Sie hier die Prioritäten für die Abarbeitung des aktiven Prozesses setzen.

Weitere Informationen zu diesem Thema finden Sie in Kapitel 5 (Optimieren des Zeitverhaltens).

'Control long Delays for Stop':

Wenn Sie timergesteuerte Prozesse verwenden, die sehr selten aufgerufen werden, d. h. einen Delay-Wert von mehr als 5 Millisekunden haben, sollten Sie diese Option einschalten, um die Prozesse schneller stoppen zu können.

'Optimize':

Der wahlweise einschaltbare Optimizer kann die Programmausführungszeit um maximal 20 % verkürzen.

6.5.2.2 Dialogfenster Process Options für den Prozessor ADSP21062

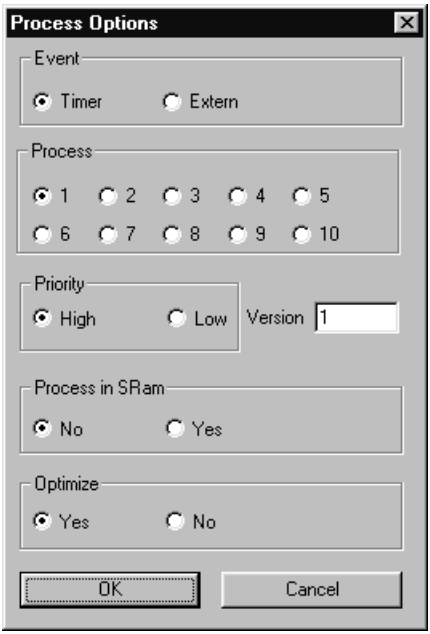


Abb. 6-10: Das Dialogfenster Process Options für den ADSP21062

‘Event’:
Hier können Sie einstellen, wie die Events erzeugt werden, die Ihr Programm starten. Sie können entweder den internen ‘Timer’ des Prozessors Ihres ***ADwin***-Systems dazu benutzen, oder Sie benutzen ein Signal (positive Flanke) am Event-Eingang an der Rückseite der ***ADwin***-Karte und stellen in dem Dialogfenster Process Options auf ‘Extern’.

Hinweis: Prozesse, die für den ADSP geschrieben sind und mit einem externen Event gesteuert werden sollen, müssen immer mit hoher Priorität ablaufen.

Hinweis: Wie Sie bei einem ***ADwin-Pro***-System einen externen Event-Eingang nutzen können, lesen Sie bitte in der Dokumentation mit dem Titel „***ADwin-Pro*** : Systembeschreibung ; Programmierung in ***ADbasic***“ unter dem EVENTENABLE-Befehl nach.

‘Process’:

Dies ist die Nummer Ihres Prozesses. Wenn Sie mehrere Prozesse gleichzeitig auf einem ***ADwin***-System ablaufen lassen möchten, müssen Sie hier jedem Prozeß eine eigene Nummer zuweisen.

‘Version’:

Hier können Sie einen Integer-Wert als Versionsnummer eingeben. Diese Versionsnummer wird bei einem kompilierten Programm (nur) in einem Text-Editor angezeigt.

‘Priority’:

Wenn Sie mehrere Prozesse gleichzeitig auf einem ***ADwin***-System ablaufen lassen möchten, können Sie hier die Prioritäten für die Abarbeitung des aktiven Prozesses setzen.

Weitere Informationen zu diesem Thema finden Sie in Kapitel 5 (Optimieren des Zeitverhaltens).

‘Process in SRam’:

Schalten Sie diese Option ein, wenn Ihr ***ADwin***-System über einen ADSP21062 mit SRAM verfügt, und Sie möchten, daß der aktive Prozeß in den SRAM geladen, und dadurch seine Ausführungszeit verkürzt wird.

‘Optimize’:

Der wahlweise einschaltbare Optimizer kann die Programm-ausführungszeit um etwa 2 % verkürzen.

6.5.3 Das Dialogfenster Parameter

Options ⇨ Parameter

Das Dialogfenster Parameter können Sie nur aufrufen, wenn Sie die ***ADwin***-Treiber auf Ihren PC geladen haben, und Ihr ***ADwin***-System gebootet haben. Es ermöglicht Ihnen einen schnellen Überblick über die Werte von einigen der globalen Variablen sowie über die Auslastung des Prozessors und den freien Speicher auf dem ***ADwin***-System. Sie können es vor allem zum Debuggen, zum Verifizieren oder Steuern Ihres Programms verwenden.

Je nach verwendetem Prozessor wird das in Abb. 6-11 oder in Abb. 6-12 gezeigte Dialogfenster geöffnet.

6.5.3.1 Dialogfenster Parameter für die Prozessoren T225, T400, T450 und T805

Die erste Zeile zeigt die Auslastung des Prozessors in dem ***ADwin***-System in Prozent (Busy) und den freien Speicher (Free Memory) des ***ADwin***-Systems in Bytes.

Falls Sie das Programm im Debug-Mode kompiliert haben, werden in der Anzeige 'Status' eventuelle Laufzeitfehler angezeigt.

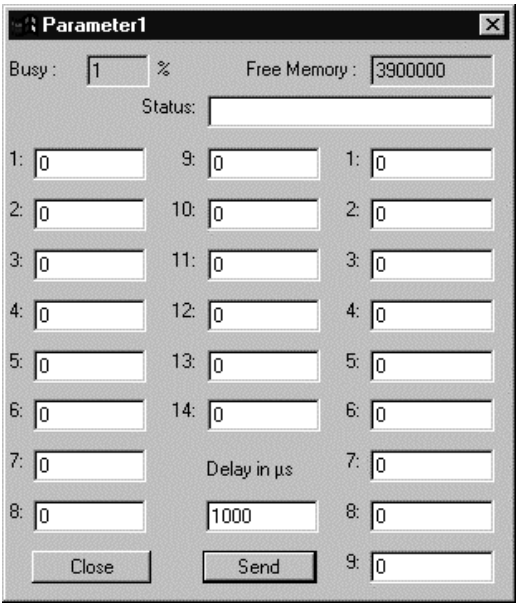


Abb. 6-11: Das Dialogfenster Parameter für die Prozessoren T225, T400, T450 und T805

Das Dialogfenster zeigt 14 globale Integer-Variablen (PAR_1 bis PAR_14) und – falls Sie den Prozessor T805 in Ihrem ***ADwin***-System haben - 9 globale Float-Variablen (FPAR_1 bis FPAR_9) an.

Alle Variablen können verändert werden, auch wenn das Programm bereits läuft. Durch Anklicken der Taste 'Send' werden alle geänderten Variablen auf das ***ADwin***-System übertragen.

Für den bei Aufruf dieses Fensters aktiven Prozeß wird auch der zeitliche Abstand (Delay) zwischen zwei von den internen Timern erzeugten Events angezeigt.

Die Einheit für den Delay-Wert beträgt für einen hoch priorisierten Prozeß 1 µs und für einen niedrig priorisierten Prozeß 64 µs (wie im Fenster jeweils angezeigt). Für die Berechnung des zeitlichen Abstands bei niedrig priorisierten Prozessen müssen Sie die angezeigte Zahl deshalb mit 64 µs multiplizieren: Eine Anzeige von 7 entspricht einer Gesamtzeit von 7 * 64 µs = 448 µs. Bei einem Prozeß mit hoher Priorität ergibt sich direkt eine Anzeige in Mikrosekunden.

Hinweis: Die Anzeige des korrekten Delay-Wertes in Abhängigkeit von der eingestellten Priorität erhalten Sie für das aktive Fenster in der Titelleiste.

Der Delay-Wert kann über dieses Fenster oder von den meisten Anwendungsprogrammen aus (*TestPoint*, *MATLAB* etc.) verändert werden.

6.5.3.2 Dialogfenster Parameter für den Prozessor ADSP21062

Die erste Zeile zeigt den freien Speicher (Free Memory) des ADSP in Bytes.

Feldbezeichnung	Speicherart
PM	interner Programmspeicher
DM	interner Datenspeicher
DX	externer Datenspeicher (DRAM)

Die zweite Zeile zeigt die Auslastung des ADSP in Prozent (Busy). Falls Sie das Programm im Debug-Mode kompiliert haben, werden in der Anzeige 'Status' eventuelle Laufzeitfehler angezeigt.

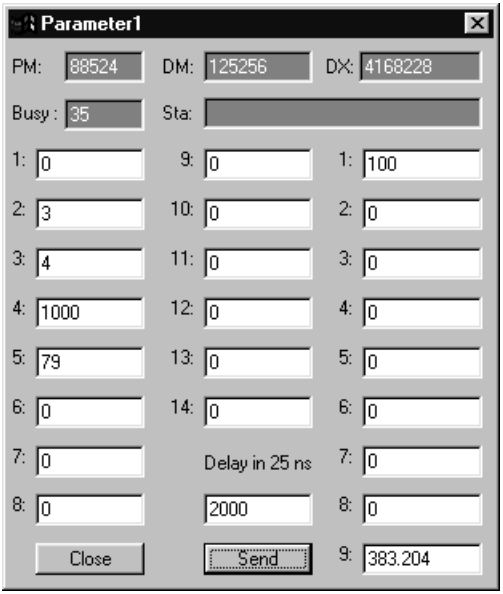


Abb. 6-12: Das Dialogfenster Parameter für den Prozessor ADSP21062

Das Dialogfenster zeigt 14 globale Integer-Variablen (PAR_1 bis PAR_14) und 9 globale Float-Variablen (FPAR_1 bis FPAR_9) an.

Alle Variablen können verändert werden, auch wenn das Programm bereits läuft. Durch Anklicken der Taste 'Send' werden alle geänderten Variablen auf das ***ADwin***-System übertragen.

Für den bei Aufruf dieses Fensters aktiven Prozeß wird auch der zeitliche Abstand (Delay) zwischen zwei von den internen Timern erzeugten Events angezeigt.

Die Einheit für den Delay-Wert beträgt für einen hoch priorisierten Prozeß 25 ns und für einen niedrig priorisierten Prozeß 100 µs. Für die Berechnung des zeitlichen Abstands bei niedrig priorisierten Prozessen müssen Sie die angezeigte Zahl deshalb mit 25 ns multiplizieren: Eine Anzeige von 7 entspricht einer Gesamtzeit von 7 * 25 ns = 175 µs.

Hinweis: Die Anzeige des korrekten Delay-Wertes in Abhängigkeit von der eingestellten Priorität erhalten Sie für das aktive Fenster in der Titelleiste.

Der Delay-Wert kann über dieses Fenster oder von den meisten Anwendungsprogrammen aus (*TestPoint*, *MATLAB* etc.) verändert werden.

6.5.4 Das Language-Fenster

Options ⇨ Language

Hier können Sie wählen, in welcher Landessprache die Fehlermeldungen des Compilers ausgegeben werden. Zur Wahl stehen Englisch oder Deutsch.

6.5.5 Die Directory-Spezifizierung

Options ⇨ Directory

In diesem Dialogfenster wird festgelegt aus welchem Verzeichnis der Compiler die Treiberdatei (*.BTL) zum Booten des ***ADwin***-Systems nehmen soll und in welchem Verzeichnis der Compiler nach Include-Dateien suchen soll, falls bei der ***ADbasic***-#INCLUDE-Anweisung nicht explizit ein Verzeichnis angegeben wird.

6.5.6 Das Dialogfenster Connect

Options ⇨ Connect

Mit ***ADbasic*** können Sie über ein vorhandenes Netzwerk (z. B. LAN, ISDN, Internet, ...) auf ein ***ADwin***-System zugreifen, das mit einem beliebigen Netzwerkrechner verbunden ist. Dazu muß auf dem Rechner der mit dem ***ADwin***-System verbunden ist zunächst das Programm ***ADserver*** gestartet werden. Anschließend können Sie mit der Taste 'Connect' die Verbindung zwischen ***ADbasic*** und dem Netzwerkrechner, der mit dem ***ADwin***-System verbunden ist, aufbauen. Sobald die Verbindung besteht, laufen alle Aktionen von ***ADbasic*** über das Netzwerk zu dem Rechner der mit dem ***ADwin***-System verbunden ist.

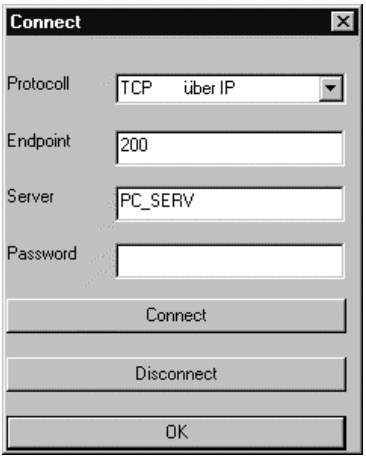


Abb. 6-11: Das Dialogfenster Connect

'Protocol':

Hier können Sie das Netzwerk-Protokoll einstellen. Das Protokoll muß auf Ihrem Rechner ordnungsgemäß installiert sein. Diese Einstellung muß mit der Einstellung im Programm ***ADserver*** identisch sein.

'Endpoint':

Endpunkt für die Netzwerk-Kommunikation. Diese Einstellung muß mit der Einstellung im Programm ***ADserver*** identisch sein.

‘Server’:

Name bzw. Adresse des Rechners, zu dem die Verbindung aufgebaut werden soll.

‘Password’:

Falls im Programm ***ADserver*** ein Paßwort angegeben wurde, muß hier das gleiche Paßwort eingetragen werden. Bei der Überprüfung des Paßwortes wird auf Groß-/Kleinschreibung geachtet.

6.5.7 Das Menü Help

Mit diesem Menü rufen Sie die *Windows*-Hilfefunktion zu ***ADbasic*** auf.

Abb. 6-12: Das Menü Help

7 Befehlsreferenz

Der folgende Abschnitt beinhaltet eine alphabetische Auflistung aller ***ADbasic***-Befehle. Der aktuell behandelte Befehl ist dabei jeweils durch Fettdruck gekennzeichnet. Zur Verdeutlichung ist für jeden Befehl ein kurzes Anwendungsbeispiel angegeben.

Hinweis: Nicht alle Befehle sind immer verfügbar. Beachten Sie bitte entsprechende Hinweise.

Für zeitintensive Befehle ist zusätzlich die Ausführungszeit in Abhängigkeit vom Prozessortyp dargestellt.

Auf der letzten Seite dieser Dokumentation finden Sie eine ausklappbare Kurzübersicht aller Befehle mit Seitenzahlangabe.

Addition +

Syntax:

```
Wert3 = Wert1 + Wert2
```

Anwendungsbeispiel:

```
wert3 = 9 + 4            'Ergebnis: wert3 = 13
```

Subtraktion -

Syntax:

```
Wert3 = Wert1 - Wert2
```

Anwendungsbeispiel:

```
wert3 = 9 - 4            'Ergebnis: wert3 = 5
```

Multiplikation *

Syntax:

```
Wert3 = Wert1 * Wert2
```

Anwendungsbeispiel:

```
wert3 = 9 * 4            'Ergebnis: wert3 = 36
```

Division /

Syntax:

```
Wert3 = Wert1 / Wert2
```

Anwendungsbeispiel:

```
wert3 = 36 / 9           'Ergebnis: wert3 = 4
```

Potenz ^

Syntax:

```
Wert3 = Wert1 ^ Wert2
```

Zeitverhalten:

			
FLOAT: 3,65µs	INT.: 450µs LONG: 450µs	INT.: 110µs LONG: 110µs FLOAT: 110µs	INT.: 430µs LONG: 430µs FLOAT: 430µs

Der Zeitbedarf steigt mit wachsendem Exponenten an.

Anwendungsbeispiel:

```
wert3 = 4 ^ 3              'Ergebnis: wert3 = 64
```

Vergleich <=>

Syntax:

Wert1 **>** *Wert2*

Beschreibung:

Die Vergleichs-Operatoren dienen zum Vergleich zweier Werte oder Ausdrücke. Das Ergebnis ist entweder wahr (1) oder falsch (0) und kann z. B. von der IF- oder der DO .. UNTIL-Anweisung ausgewertet werden.

Operator	Bedeutung
<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich
=	gleich
<>	ungleich

Anwendungsbeispiel:

```
DIM wert1, wert2 AS INTEGER

EVENT:
wert1 = -5
if (wert1 < 0) then wert1 = 0
REM Ergebnis: wert1 = 0
```

ABS

Syntax:

```
Wert2 = ABS(Wert1)
```

Beschreibung:

Die Funktion ABS liefert den Betrag einer Integer/Long-Variablen.

Zeitverhalten:

LONG: 0,1µs LONG: 3,4 LONG: 2,9µs LONG: 25µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS INTEGER

EVENT:
wert1 = -5
wert2 = ABS(wert1)       'Ergebnis: wert2 = 5
```

ABSF

Syntax:

Wert2 = ABSF(*Wert1*)

Beschreibung:

Die Funktion ABSF liefert den Betrag einer Float-Variablen.

Zeitverhalten:

FLOAT: 0,1µs

FLOAT: 2,0µs

FLOAT: 6,5µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = -5.3
wert2 = ABSF(wert1)      'Ergebnis: wert2 = 5.3
```

ACTIVATE_PC

Syntax:

ACTIVATE_PC

Beschreibung:

Der Befehl ACTIVATE_PC startet die Actionlist des Realtime-Objekts unter *TestPoint*.

Dazu wird eine globale Variable auf 1 gesetzt. Der PC kann durch Abfragen dieser Variablen erkennen, daß die aktuelle PC-Funktion ausgeführt werden soll.

Hinweis: Dieser Befehl ist speziell für *TestPoint* entwickelt worden, um aus ***ADbasic*** *TestPoint*-Actions auslösen zu können.

Anwendungsbeispiel:

```
DIM wert AS INTEGER       'Deklaration

EVENT:
wert = ADC(1)             'Meßwerterfassung
IF (wert > 1000) THEN     'Vergleich
  PAR_1 = wert            'Meßwert im Parameter 1
                          'speichern
  ACTIVATE_PC             'PC aktivieren
ENDIF
```

ADC

Syntax:

Meßwert = ADC(*EingangsNr , Verstärkung*)

***ADwin*- Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)***-Karten.

Bei ***ADwin-Pro*** gibt es auch einen Befehl mit diesem Namen, für den diese Beschreibung NICHT gilt. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl ADC mißt einen analogen Eingang und gibt das Meßergebnis in ADC-Digits zurück. Falls eine Verstärkung benötigt wird, kann diese optional als zweiter Parameter übergeben werden. Die Verstärkungen 1, 2, 4, 8 sind wählbar. Der Befehl ADC stellt zunächst den Multiplexer auf den gewünschten Eingangskanal und wartet 4µs (***ADwin-GOLD*** 6,5µs) auf das Einschwingen des Multiplexers. Danach wird die Messung gestartet, und der Wert nach dem Ende der Konvertierung des ADCs ausgelesen.

Hinweis: Die Verstärkung ist bei ***ADwin-light***-Karten nicht einstellbar.

Zeitverhalten beim *ADwin-GOLD*-System:

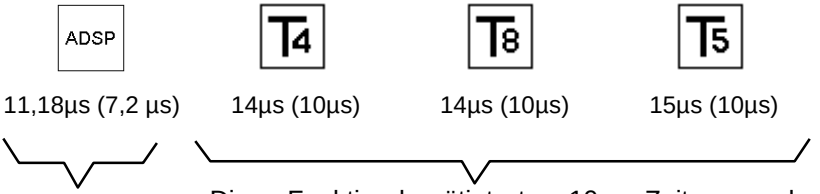
ADSP
14,4µs (7,7 µs)

Diese Funktion benötigt etwa 7,7 µs Zeit, wenn der Prozeß mit einem kürzeren Delay als 20 µs abläuft. Anderenfalls benötigt die Funktion ca. 14,4 µs.

Beim ***ADwin-GOLD***-System greift dieser Befehl auf einen 16- Bit AD- Wandler zu. Für den (schnelleren) 12- Bit AD- Wandler gibt es beim ***ADwin-GOLD***-System auch noch den Befehl ADC12.

...Fortsetzung Befehl ADC:

Zeitverhalten bei der *ADwin*-Karte und *ADwin-light*-Karte:



Diese Funktion benötigt etwa 10 µs Zeit, wenn der Prozeß mit einem kürzeren Delay als 31 µs abläuft. Anderenfalls benötigt die Funktion ca. 14 µs.

Diese Funktion benötigt etwa 7 µs Zeit, wenn der Prozeß mit einem kürzeren Delay als 20 µs abläuft. Anderenfalls benötigt die Funktion ca. 12 µs.

Bei den ***ADwin-***Karten greift dieser Befehl auf einen 12- Bit AD- Wandler zu.

Hinweis: Die kürzeren Ausführungszeiten bei kürzeren Delays sind wie folgt begründet : Anhand der sehr kleinen Delay Zeiten erkennt der Compiler automatisch, daß die Zeit sowieso nicht ausreichen würde, um den Multiplexer umzustellen. Er folgert daraus, daß vom Anwender eine Messung ohne Umstellung des Multiplexers beabsichtigt sein muß, und verzichtet demnach automatisch auf die normalerweise einzuhaltende Einschwingzeit des Multiplexers. (Falls man es mit so kurzen Delays zu tun hat, sollte man mindestens 4 µs (***ADwin-GOLD*** 6,5µs) vor der ersten Benutzung von ADC den Befehl SET_MUX entsprechend aufrufen – ansonsten ist der erste Messwert eventuell nicht korrekt). Prinzipiell wird bei sehr kleinen Delays (GLOBALDELAY) empfohlen, an Stelle des ADC Befehls die Kombination der Befehle SET_MUX, START_CONV, WAIT_EOC und READADC zu benutzen.

...Fortsetzung Befehl ADC:

! Achtung: Falls eine Eingangsnummer >16 übergeben wird, ist das Ergebnis undefiniert.

Anwendungsbeispiel:

```
DIM iw AS INTEGER          'Deklaration

EVENT:
iw = ADC(1,4)              'mißt den analogen Eingang
                           '1 mit der Verstärkung 4
PAR_1 = iw                 'Meßwert in den Parameter
                           '1 schreiben, wo er vom PC
                           'abgeholt werden kann
```

Umrechnung der ADC-Werte bei 16 Bit ADCs:

Die auf dem ***ADwin-GOLD***-System verwendeten 16 Bit ADCs teilen den Meßbereich (von 20 Volt) in 65536 gleich große Bereiche ein.

Für die Umrechnung auf die Eingangsspannung gilt folgende Formel:

$$Spannung = (Digits - 32768_{bipolar}) * \frac{Meßbereich}{65536 * Verstärkung}$$

Für den Fall, daß die Verstärkung gleich 1 gewählt wurde, gelten die in der Tabelle angegebenen Werte.

Eingangs-spannungs-bereich	**ADC-Wert**			
	0	32768	65535	1Digit
-10...+10 V	-10 V	0 V	+9,999695 V	305,175 µV

...Fortsetzung Befehl ADC**:**

Umrechnung der ADC-Werte bei 12 Bit ADCs:

Bei den ***ADwin***-Karten mit 12 Bit ADCs wird der gewählte Messbereich in 4096 gleich große Bereiche eingeteilt.

Für die Umrechnung auf die Eingangsspannung gilt folgende Formel:

$$Spannung = (Digits - 2048_{bipolar}) * \frac{Meßbereich}{4096 * Verstärkung}$$

Hinweis: Bei unipolarer Einstellung fällt der Offset von 2048 weg.

Für den Fall, daß die Verstärkung gleich 1 gewählt wurde, gelten die in der Tabelle angegebenen Werte.

Eingangs-spannungs-bereich	**ADC-Wert**			
	0	2048	4095	1Digit
0...+10 V	0 V	+5 V	+9,99756 V	2,44 mV
-5...+5 V	-5 V	0 V	+4,99756 V	2,44 mV
-10...+10 V	-10 V	0 V	+9,99512 V	4,88 mV

ADC12

Syntax:

Meßwert = ADC12(*EingangsNr , Verstärkung*)

***ADwin*- Systeme:**

Nur für ***ADwin-GOLD***.

Beschreibung:

Der Befehl ADC12 mißt einen analogen Eingang und gibt das Meßergebnis in ADC-Digits zurück. Falls eine Verstärkung benötigt wird, kann diese optional als zweiter Parameter übergeben werden. Die Verstärkungen 1, 2, 4, 8 sind wählbar. Der Befehl ADC12 stellt zunächst den Multiplexer auf den gewünschten Eingangskanal und wartet ca. 1,5 µs auf das Einschwingen des Multiplexers. Danach wird die Messung gestartet, und der Wert nach dem Ende der Konvertierung des ADCs ausgelesen.

Zeitverhalten beim *ADwin-GOLD*-System:

3,1 µs

Beim ***ADwin-GOLD***-System greift dieser Befehl auf einen 12- Bit AD- Wandler zu. Für den (genaueren) 16- Bit AD- Wandler gibt es beim ***ADwin-GOLD***-System auch noch den Befehl ADC.

Hinweis: Prinzipiell wird bei sehr kleinen Delays (GLOBALDELAY) empfohlen, an Stelle des ADC12 Befehls die Kombination der Befehle SET_MUX, START_CONV, WAIT_EOC und READADC12 zu benutzen.

...Fortsetzung Befehl ADC12:

Anwendungsbeispiel:

```
DIM iw AS INTEGER          'Deklaration

EVENT:
iw = ADC12(1,4)            'mißt den analogen Eingang
                           '1 mit der Verstärkung 4
PAR_1 = iw                 'Meßwert in den Parameter
                           '1 schreiben, wo er vom PC
                           'abgeholt werden kann
```

Umrechnung der ADC-Werte bei den *ADwin-GOLD* 12 Bit ADCs:

Die auf dem ***ADwin-GOLD***-System verwendeten 12 Bit ADCs teilen den Meßbereich (von 20 Volt) in 4096 gleich große Bereiche ein. Um den Vergleich mit den Messwerten der 16 Bit ADCs zu vereinfachen, gibt der Befehl ADC12 das Ergebnis in den höherwertigen Bits (Bits 31 bis 4) zurück. Somit liefert der Befehl ADC12(1) in den höherwertigen Bits das gleiche Ergebnis wie der (16 Bit) Befehl ADC(1). Die vier niederwertigsten Bits haben stets den Wert 0.

Für die Umrechnung auf die Eingangsspannung gilt folgende Formel:

$$Spannung = (Digits - 32768_{bipolar}) * \frac{Meßbereich}{65536 * Verstärkung}$$

Für den Fall, daß die Verstärkung gleich 1 gewählt wurde, gelten die in der Tabelle angegebenen Werte.

Eingangs-spannungs-bereich	**ADC12-Wert**			
	0	32768	65520	16Digits
-10...+10 V	-10 V	0 V	+9,99512 V	4,88 mV

AND

Syntax:

Wert3 = *Wert1* AND *Wert2*

oder bei IF ... THEN und DO ... UNTIL

Ausdruck1 AND *Ausdruck2*

Beschreibung:

Der Operator AND wird vom Compiler automatisch entweder als bitweiser Operator oder als logischer Operator interpretiert.

Als **bitweiser Operator** vergleicht er die einzelnen Bits von zwei Werten miteinander. Im Resultat dieser Operation steht nur bei denjenigen Bits eine 1, an denen bei beiden Werten an den entsprechenden Bitpositionen eine 1 enthalten ist.

Als **logischer Operator** innerhalb der *Bedingung* von IF ... THEN oder DO ... UNTIL Strukturen ermittelt er die Aussage wahr (1) oder falsch (0) für die UND- Verknüpfung von zwei Ausdrücken.

Anwendungsbeispiel (als bitweiser Operator):

```
DIM wert1, wert2, wert3 AS LONG

wert1 = 0100B
wert2 = 0110B
wert3 = wert1 AND wert2
                          'Ergebnis: wert3 = 0100B
```

Hinweis: Als bitweiser Operator nur für Integer- und Long-Variablen bzw. Konstanten.

...Fortsetzung Befehl AND:

Anwendungsbeispiel (als logischer Operator):

```
DIM f AS FLOAT
DIM wert4 AS LONG

f = 3.14
IF ((f < 9.1) AND (f > 3.1)) THEN
  wert4 = 1
ELSE
  wert4 = 0
ENDIF
                          'Ergebnis: wert4 = 1
```

Hinweis: Falls mehrere AND (oder OR) Operatoren in einer Zeile verwendet werden, sind entsprechend viele Klammern zu setzen.

ARCCOS

Syntax:

Wert2 = ARCCOS(*Wert1*)

Beschreibung:

Die Funktion ARCCOS liefert den Arcus-Cosinus eines Arguments. *Wert1* muß zwischen -1 und +1 liegen, das Ergebnis wird im Bogenmaß angegeben.

Zeitverhalten:

FLOAT: 2,85µs

FLOAT: 25µs

FLOAT: 100µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = 0.5
wert2 = ARCCOS(wert1)    'Ergebnis: wert2 = 1.0472
```

ARCSIN

Syntax:

Wert2 = ARCSIN(*Wert1*)

Beschreibung:

Die Funktion ARCSIN liefert den Arcus-Sinus eines Arguments. *Wert1* muß zwischen -1 und +1 liegen, das Ergebnis wird im Bogenmaß angegeben.

Zeitverhalten:

FLOAT: 2,8µs

FLOAT: 25µs

FLOAT: 100µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = 0.5
wert2 = ARCSIN(wert1)    'Ergebnis: wert2 = 0.5236
```

ARCTAN

Syntax:

Wert2 = ARCTAN(*Wert1*)

Beschreibung:

Die Funktion ARCTAN liefert den Arcus-Tangens eines Arguments. Das Ergebnis wird im Bogenmaß angegeben.

Zeitverhalten:

FLOAT: 1,9µs

FLOAT: 29 µs

FLOAT: 120 µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = 0.5
wert2 = ARCTAN(wert1)    'Ergebnis: wert2 = 0.4636
```

CLEAR_DIGOUT

Syntax:

CLEAR_DIGOUT(*AusgangsNr*)

***ADwin-* Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)-***Karten.

Bei ***ADwin-Pro*** darf dieser Befehl nicht verwendet werden. Dort gibt es einen Befehl mit den Namen DIGOUT. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl CLEAR_DIGOUT setzt das Bit *AusgangsNr* des digitalen Ausgangs auf 0.

Hinweise: Die digitalen Ausgänge sind beim ***ADwin-GOLD***-Systems (in der Standardkonfiguration) von 16 bis 31, bei der ***ADwin***-Karte von 0 bis 15 durchnumeriert. Für die *AusgangsNr* muß in beiden Fällen eine konstante Zahl zwischen 0 und 15 eingesetzt werden. Beachten Sie aber bitte, daß sich auf einer ***ADwin-light***-Karte nur sechs digitale Ausgänge befinden und, daß die 16 digitalen Ausgänge auf der ***ADwin***-Karte nur über den mitgelieferten I/O-Erweiterungsstecker benutzt werden können.

Variablen dürfen in diesem Befehl nicht verwendet werden. Wenn Sie den zu löschenden Ausgang durch eine Variable bestimmen wollen, verwenden Sie den Befehl DIGOUT_WORD.

Beim ***ADwin-Gold***-System müssen die Ausgänge zuvor einmalig mit dem Befehl CONF_DIO(12) konfiguriert werden.

...Fortsetzung Befehl CLEAR_DIGOUT:

Anwendungsbeispiel:

```
DIM wert AS INTEGER      'Deklaration

INIT:
CONF_DIO(12)             'Dig. Ausgänge konfigu-
                         'rieren (nur ADwin-GOLD)
SET_DIGOUT(0)            'Dig. Ausgang DIO 16 bzw.
                         '0 setzen

EVENT:
wert = ADC(1)            'Meßwerterfassung

IF (wert > 3000) THEN
  CLEAR_DIGOUT(0)        'Dig. Ausgang DIO 16
                         '(ADwin-GOLD) bzw. 0
                         '(ADwin-Karten)
                         'zurücksetzen
ENDIF
```

CO4_CLEAR

Syntax:

CO4_CLEAR

***ADwin-* Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-CO1*** bzw. ***ADwin-CO1L***, ***ADwin-CO2***, ***ADwin-CO3*** und ***ADwin-CO4*** ausgerüstet sind.

Beschreibung:

Der Befehl CO4_CLEAR löscht alle 16-Bit-Zähler auf ***ADwin***- bzw. ***ADwin-light***-Karten mit Zähleroption, falls diese vorhanden ist.

Anwendungsbeispiel:

```
DIM iw, flag AS INTEGER  'Deklaration

EVENT:
iw = ADC(1,8)            'Meßwerterfassung
IF (iw > 3000) THEN      'Vergleich mit Schwellwert
  IF (flag = 0) THEN     'Abfrage, ob Flag gesetzt
    CO4_CLEAR            'Zähler rücksetzen
    CO4_START            'Starten des Zählers
    flag = 1             'Flag setzen
  ENDIF
ELSE
  IF (flag = 1) THEN     'Abfrage, ob Flag gesetzt
    CO4_STOP             'Zähler stoppen
    PAR_1 = CO4_READ(1) 'Zählerstand speichern
    flag = 0             'Flag rücksetzen
  ENDIF
ENDIF
```

CO4_READ

Syntax:

wert = CO4_READ(*Nummer*)

***ADwin*- Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-CO1*** bzw. ***ADwin-CO1L***, ***ADwin-CO2***, ***ADwin-CO3*** und ***ADwin-CO4*** ausgerüstet sind.

Beschreibung:

Die Befehl CO4_READ liest den 16-Bit-Zähler *Nummer* auf ***ADwin***- bzw. ***ADwin-light***-Karten mit Zähleroption aus, falls dieser vorhanden ist.

Anwendungsbeispiel:

```
DIM iw, flag AS INTEGER  'Deklaration

EVENT:
iw = ADC(1,8)            'Meßwerterfassung
IF (iw > 3000) THEN      'Vergleich mit Schwellwert
  IF (flag = 0) THEN     'Abfrage, ob Flag gesetzt
    CO4_CLEAR            'Zähler rücksetzen
    CO4_START            'Starten des Zählers
    flag = 1             'Flag setzen
  ENDIF
ELSE
  IF (flag = 1) THEN     'Abfrage, ob Flag gesetzt
    CO4_STOP             'Zähler stoppen
    PAR_1 = CO4_READ(1) 'Zählerstand speichern
    flag = 0             'Flag rücksetzen
  ENDIF
ENDIF
```

CO4_START

Syntax:

CO4_START

***ADwin-* Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-CO1*** bzw. ***ADwin-CO1L***, ***ADwin-CO2***, ***ADwin-CO3*** und ***ADwin-CO4*** ausgerüstet sind.

Beschreibung:

Der Befehl CO4_START startet alle 16-Bit-Zähler auf ***ADwin***- bzw. ***ADwin-light***-Karten mit Zähleroption, falls diese vorhanden ist.

Anwendungsbeispiel:

```
DIM iw, flag AS INTEGER  'Deklaration

EVENT:
iw = ADC(1,8)            'Meßwerterfassung
IF (iw > 3000) THEN      'Vergleich mit Schwellwert
  IF (flag = 0) THEN     'Abfrage, ob Flag gesetzt
    CO4_CLEAR            'Zähler rücksetzen
    CO4_START            'Starten des Zählers
    flag = 1             'Flag setzen
  ENDIF
ELSE
  IF (flag = 1) THEN     'Abfrage, ob Flag gesetzt
    CO4_STOP             'Zähler stoppen
    PAR_1 = CO4_READ(1) 'Zählerstand speichern
    flag = 0             'Flag rücksetzen
  ENDIF
ENDIF
```

CO4_STOP

Syntax:

CO4_STOP

***ADwin*- Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-CO1*** bzw. ***ADwin-CO1L***, ***ADwin-CO2***, ***ADwin-CO3*** und ***ADwin-CO4*** ausgerüstet sind.

Beschreibung:

Der Befehl CO4_STOP stoppt alle 16-Bit-Zähler auf ***ADwin***- bzw. ***ADwin-light***-Karten mit Zähleroption, falls diese vorhanden ist.

Anwendungsbeispiel:

```
DIM iw, flag AS INTEGER  'Deklaration

EVENT:
iw = ADC(1,8)            'Meßwerterfassung
IF (iw > 3000) THEN      'Vergleich mit Schwellwert
  IF (flag = 0) THEN     'Abfrage, ob Flag gesetzt
    CO4_CLEAR            'Zähler rücksetzen
    CO4_START            'Starten des Zählers
    flag = 1             'Flag setzen
  ENDIF
ELSE
  IF (flag = 1) THEN     'Abfrage, ob Flag gesetzt
    CO4_STOP             'Zähler stoppen
    PAR_1 = CO4_READ(1) 'Zählerstand speichern
    flag = 0             'Flag rücksetzen
  ENDIF
ENDIF
```

CONF_DIO

Syntax:

CONF_DIO(*Wert*)

***ADwin-* Systeme:**

Notwendig und verfügbar nur für das System ***ADwin-GOLD***. Bei allen anderen Systemen nicht notwendig.

Beschreibung:

Das ***ADwin-GOLD***-System stellt insgesamt 32 frei konfigurierbare digitale Ein- bzw. Ausgänge zur Verfügung. Nach dem Einschalten der Versorgungsspannung sind alle I/O-Anschlüsse zunächst als Eingang geschaltet. Sie können softwaremäßig blockweise zu jeweils acht als Ein- oder Ausgang konfiguriert werden.

Werden die digitalen Ein- und Ausgänge mit dem ***ADbasic***-Befehl CONF_DIO(12) konfiguriert (d.h. DIO 0-15 sind Eingänge und DIO 16-31 sind Ausgänge), so kann auch mit den ***ADbasic***-Befehlen DIGIN_WORD, DIGOUT_WORD, DIGIN, SET_DIGOUT und CLEAR_DIGOUT darauf zugegriffen werden. Dabei sind (in Bezug auf diese Befehle) Programme auf dem ***ADwin-GOLD***-System absolut (Quellcode-) kompatibel zu Programmen auf ***ADwin***-Karten.

Wir empfehlen daher dringend für normale Anwendungen die Konfiguration CONF_DIO(12).

Sind andere I/O-Konfigurationen gewünscht oder notwendig, so muß direkt per PEEK- und POKE-Befehl das entsprechende Hardware-Register ausgelesen bzw. beschrieben werden (siehe dazu weitere Informationen im ***ADwin-GOLD*** Hardware- Handbuch).

Hinweis: Falls das ***ADwin-GOLD***-System nicht mit diesem Befehl entsprechend konfiguriert wird, können keine digitalen Ausgänge gesetzt werden, da nach dem Einschalten der Versorgungsspannung alle I/O-Anschlüsse zunächst als Eingang geschaltet sind.

...Fortsetzung Befehl CONF_DIO:

Anwendungsbeispiel:

```
CONF_DIO(12)              'Konfiguriert DIO 0 – 15
                          'als Eingänge und DIO 16 -
                          '31 als Ausgänge
```

COS

Syntax:

Wert2 = COS(*Wert1*)

Beschreibung:

Die Funktion COS liefert den Cosinus eines Arguments, das im Bogenmaß angeben ist.

Zeitverhalten:

FLOAT: 1,3µs

FLOAT: 28 µs

FLOAT: 150 µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = -5.3
wert2 = COS(wert1)       'Ergebnis: wert2 = 0.55...
```

DAC

Syntax:

DAC(*Nummer, Wert*)

***ADwin*- Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)***-Karten.

Bei ***ADwin-Pro*** gibt es auch einen Befehl mit diesem Namen, für den diese Beschreibung NICHT gilt. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl DAC gibt den angegebenen *Wert* auf dem Ausgang *Nummer* aus.

Hinweis: Die analogen Ausgänge sind von 1 bis 8 durchnumeriert. Beachten Sie aber bitte, daß sich standardmäßig sowohl auf einer ***ADwin-(light)***-Karte als auch auf einem ***ADwin-GOLD***-System nur zwei DACs befinden. Eine ***ADwin***-Karte kann optional mit vier zusätzlichen DACs aufgerüstet sein. Ein ***ADwin-GOLD***-System kann optional mit sechs zusätzlichen DACs aufgerüstet sein.

Anwendungsbeispiel:

```
REM Digitaler P-Regler
DIM sw, aw AS LONG       'Deklaration
DIM v, stell AS LONG     'Deklaration

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

...Fortsetzung Befehl DAC:

Umrechnung der DAC-Werte bei 16 Bit DACs:

Die auf dem ***ADwin-GOLD***-System verwendeten DACs haben eine Auflösung von 16 Bit und teilen den Spannungsbereich (von 20 Volt) somit in 65536 gleich große Schritte ein.

Für die Umrechnung auf die Ausgabespannung gilt folgende Formel:

$$Spannung = (Digits - 32768_{bipolar}) * \frac{Spannungsbereich}{65536}$$

Die folgende Tabelle zeigt einige DAC-Ausgabewerte und die zugehörigen Ausgangsspannungswerte.

Ausgangs-spannungs-bereich	**DAC-Wert**			
	0	32768	65535	1Digit
-10...+10 V	-10 V	0 V	+9,999695 V	305,175 µV

Umrechnung der DAC-Werte bei 12 Bit DACs:

Bei den ***ADwin-***Karten mit 12 Bit DACs wird der gewählte Spannungsbereich in 4096 gleich große Schritte eingeteilt.

Für die Umrechnung auf die Ausgabespannung gilt folgende Formel:

$$Spannung = (Digits - 2048_{bipolar}) * \frac{Spannungsbereich}{4096}$$

Hinweis: Bei unipolarer Einstellung fällt der Offset von 2048 weg.

...Fortsetzung Befehl DAC:

Die folgende Tabelle zeigt die DAC-Ausgabewerte für die verschiedenen Ausgangsspannungsbereich-Einstellungen.

Ausgangs-spannungs-bereich	**DAC-Wert**			
	0	2048	4095	1Digit
0...+10 V	0 V	+5 V	+9,99756 V	2,44 mV
-5...+5 V	-5 V	0 V	+4,99756 V	2,44 mV
-10...+10 V	-10 V	0 V	+9,99512 V	4,88 mV

DEC

Syntax:

`DEC(wert)`

Beschreibung:

Der Befehl DEC dekrementiert den übergebenen `wert` um 1.

Hinweise: Dieser Befehl darf nur auf den Datentyp INTEGER angewendet werden.

Die Anweisung `DEC(wert)` führt zum gleichen Ergebnis wie die Programmzeile: `wert=wert-1`, benötigt jedoch eine geringere Ausführungszeit.

Anwendungsbeispiel:

```
DIM index AS INTEGER
DIM DATA_1[1000] AS INTEGER

INIT:
index=1000

EVENT:
DAC(1,DATA_1[index])     'Wert auf DAC1 ausgeben
DEC(index)               'index um 1 verringern
IF (index<1) THEN
  index=1000             'Nach 1000 Ausgaben von
ENDIF                    'vorne beginnen
```

#DEFINE

Syntax:

`#DEFINE` *`NeuerName AlterName`*

Beschreibung:

Mit der Funktion `#DEFINE` können Parameter, Datenstrukturen oder beliebige Programmzeilen durch einen neudefinierten Namen ersetzt werden.

Anwendungsbeispiel:

```
#DEFINE Sollwert par_1
```

Diese Anweisung gibt par_1 den Namen Sollwert

```
#DEFINE Messwerte data_1
```

Diese Anweisung gibt data_1 den Namen Messwerte

! Achtung: In der DEFINE-Zeile darf auf *keinen* Fall ein Kommentar stehen, da dieser mit eingesetzt würde.

DIGIN

Syntax:

Ergebnis = DIGIN(*EingangsNr*)

***ADwin-* Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)-***Karten.

Bei ***ADwin-Pro*** darf dieser Befehl nicht verwendet werden.

Beschreibung:

Der Befehl DIGIN ermittelt den Wert des digitalen Eingangs *EingangsNr*. Ein TTL-High-Pegel setzt die Variable *Ergebnis* auf 1, ein TTL-Low-Pegel auf 0.

Hinweis: Die digitalen Eingänge sind von 0 bis 15 durchnumeriert. Beachten Sie aber bitte, daß sich auf einer ***ADwin-light***-Karte nur sechs digitale Eingänge befinden und, daß die 16 digitalen Eingänge auf der ***ADwin***-Karte nur über den mitgelieferten I/O-Erweiterungsstecker benutzt werden können.

Anwendungsbeispiel:

```
DIM DATA_1[10000] AS INTEGER AS FIFO

EVENT:
IF (DIGIN(0) = 1) THEN    'Abfrage ob der digitale
                          'Eingang 0 gesetzt ist.
  DATA_1 = ADC(1)         'Meßwerterfassung
ENDIF
```

DIGIN_WORD

Syntax:

Ergebnis = DIGIN_WORD()

***ADwin-* Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)***-Karten.

Bei ***ADwin-Pro*** darf dieser Befehl nicht verwendet werden. Dort gibt es Befehle mit den Namen DIGIN_WORD1 und DIGIN_WORD2. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Die Funktion DIGIN_WORD() liest alle 16 digitalen Eingänge auf einmal aus.

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

Dig. Eingang	15	...	3	2	1	0
Bit	15	...	3	2	1	0
DEZ-Wert	32768	...	8	4	2	1

Berechnungsbeispiel:

Die Funktion DIGIN_WORD() liefert den Dezimalwert 11. Anhand des Bitmusters von dezimal 11 können Sie erkennen, welche digitalen Eingänge gesetzt sind. In diesem Beispiel sind die Eingänge 0, 1 und 3 gesetzt, da die Summe der zugeordneten Bitwerte den Dezimalwert 11 ergibt.

...Fortsetzung Befehl DIGIN_WORD:

Hinweis: Die digitalen Eingänge sind von 0 bis 15 durchnumeriert. Beachten Sie aber bitte, daß sich auf einer ***ADwin-light***-Karte nur sechs digitale Eingänge befinden und, daß die 16 digitalen Eingänge auf der ***ADwin***-Karte nur über den mitgelieferten I/O-Erweiterungsstecker benutzt werden können.

Anwendungsbeispiel:

```
DIM DATA_1[10000] AS INTEGER AS FIFO

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

DIGOUT_WORD

Syntax:

DIGOUT_WORD(*Auswahl*)

***ADwin*- Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)***-Karten.

Bei ***ADwin-Pro*** darf dieser Befehl nicht verwendet werden. Dort gibt es Befehle mit den Namen DIGOUT_WORD1 und DIGOUT_WORD2. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl DIGOUT_WORD setzt gleichzeitig alle digitalen Ausgänge des ***ADwin*-*GOLD***-Systems bzw. der ***ADwin***-/***ADwin-light***-Karte auf den durch *Auswahl* vorgegebenen Wert. *Auswahl* ist ein 16-Bit-Wert, wobei jedem digitalen Ausgang ein Bit dieses Wertes zugeordnet ist (siehe Tabelle).

***ADwin-GOLD*-System: Dig. Ausgang (DIO)**	31	30	...	18	17	16
***ADwin*-/*ADwin-light*-Karte: Dig. Ausgang**	15	14	...	2	1	0
Bit	15	14	...	2	1	0
DEZ-Wert	32768	16384	...	4	2	1

Berechnungsbeispiel:

Die Ausgänge 0, 2 und 14 sollen gesetzt werden. Sie erhalten das Bitmuster 0100000000000101, welches dem Dezimalwert 16389 entspricht und schreiben: DIGOUT_WORD(16389).

...Fortsetzung Befehl DIGOUT_WORD:

! Achtung: Alle Ausgänge, deren zugeordnetes Bit im Wert *Auswahl* 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.

Hinweis: Die digitalen Ausgänge sind beim ***ADwin-GOLD***-Systems (in der Standardkonfiguration) von 16 bis 31, bei der ***ADwin***-Karte von 0 bis 15 durchnumeriert. In beiden Fällen müssen Sie den gewünschten Wert in die niederwertigsten 16 Bit von *Auswahl* schreiben. Beachten Sie aber bitte, daß sich auf einer ***ADwin-light***-Karte nur sechs digitale Ausgänge befinden und, daß die 16 digitalen Ausgänge auf der ***ADwin***-Karte nur über den mitgelieferten I/O-Erweiterungsstecker benutzt werden können.

Beim ***ADwin-Gold***-System müssen die Ausgänge zuvor einmalig mit dem Befehl CONF_DIO(12) konfiguriert werden.

Anwendungsbeispiel:

```
INIT:
CONF_DIO(12)              'Dig. Ausgänge konfigu-
                          'rieren (nur ADwin-GOLD)

EVENT:
DIGOUT_WORD(7)            'Digitale Ausgänge DIO 16,
                          '17 u. 18 (ADwin-GOLD)
                          'bzw. 0, 1 u. 2 (ADwin-
                          'Karten) setzen, alle
                          'anderen Ausgänge werden
                          'gelöscht!
```

DIM

Syntax:

DIM *V1*{[*L1*]} {, *V2*{[*L2*]}} AS Var_*typ* {AS FIFO}

Beschreibung:

Deklariert eine oder mehrere Variablen *V1*, *V2* etc. vom Typ *Typ*. Wenn Sie nach dem Variablennamen eine Feldlänge *L1*, *L2* etc. angegeben, erzeugt der Compiler ein eindimensionales Array. Die Arrays der globalen DATA-Variablen können zusätzlich auch als FIFO-Ringspeicher definiert werden. Weitere Informationen zu dieser speziellen Variablenart finden bei der Erklärung des Befehls FIFO.

Hinweise: Die zur Verfügung stehenden Typen sind INTEGER, LONG und (außer bei dem Prozessor T4) FLOAT. Weitere Informationen hierzu finden Sie in dem Kapitel 4.4.3 (Selbstdefinierbare Variablen).

Anwendungsbeispiel:

```
DIM var1 AS INTEGER       'Dimensioniert var1 als
                          'Integer-Variable
DIM array1[1000] AS LONG
                          'Dimensioniert array1 mit
                          'einer Feldlänge von 1000
                          'Elementen
DIM DATA_20[1000] AS INTEGER AS FIFO
                          'Dimensioniert die globale
                          'Variable DATA_20 mit einer
                          'Länge von 1000 Elementen
                          'als Ringspeicher
```

Hinweis: Für den ADSP kann zusätzlich eine Angabe zu der Speicherart gemacht werden (Seite 24/Tabelle 4-1).

DO ... UNTIL

Syntax:

```
DO
   Anweisungsblock
UNTIL (Bedingung)
```

Beschreibung:

Der Anweisungsblock wird solange wiederholt ausgeführt, bis *Bedingung* wahr ist.

! Achtung: In einem hoch priorisierten Prozeß kann die DO ... UNTIL-Schleife von keinem anderen Prozeß unterbrochen werden. Dadurch ist es dem Prozessor des ***ADwin***-Systems während der Zeit der Schleifenabarbeitung nicht möglich, auf andere Events zu reagieren. Die DO ... UNTIL-Schleife darf deshalb in hoch priorisierten Prozessen nur verwendet werden, wenn die Anzahl der Schleifendurchläufe niedrig gehalten wird.

Anwendungsbeispiel:

```
DIM wert AS INTEGER

EVENT:
DO
   wert = ADC(1, 4)        'Meßwert auslesen
UNTIL (wert < 2048)        'Schleife wiederholen,
                           'solange der Wert größer
                           'oder gleich 2048 ist
ACTIVATE_PC                'PC aktivieren
```

END

Syntax:

END

Beschreibung:

Der Befehl END beendet einen Prozeß.

Hinweis: Nach dem Befehl END werden, falls vorhanden, noch die Anweisungen im Abschnitt FINISH: ausgeführt.

Anwendungsbeispiel:

```
EVENT:
IF (ADC(1) > 3000) THEN  'Messen und Vergleich
  SET_DIGOUT(0)          'Dig. Ausgang setzen
  END                    'Prozeß beenden
ENDIF
```

EXP

Syntax:

Wert2 = EXP(*Wert1*)

Beschreibung:

Die Funktion EXP liefert den Exponentialwert eines Arguments zur Basis e.

Zeitverhalten:

FLOAT: 1,3µs

FLOAT: 32 µs

FLOAT: 130 µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = 5
wert2 = EXP(wert1)        'Ergebnis:
                          'wert2 = 148.41...
```

FIFO

Syntax:

`DIM DATA_`*`X`*`[`*`Laenge`*`] AS `*`Var_typ`*` AS FIFO`

Beschreibung:

Dimensioniert die Variable `DATA_`*`X`* als FIFO-Ringspeicher mit *`Laenge`* Elementen vom Typ *`Var_typ`*. Anstelle von *X muß* eine Ziffer zwischen 1 und 200 stehen.

Hinweis: Da ***ADbasic*** den FIFO intern wie einen Datensatz verwaltet, darf dieselbe Datensatznummer *nicht gleichzeitig* als Nummer eines FIFO *und* einer normalen DATA-Variablen benutzt werden.

! Achtung: Die FIFOs werden beim Start nicht automatisch gelöscht, sie sollten deshalb mit dem Befehl `FIFO_CLEAR(`*`DatenSatzNr`*`)` z. B. im Abschnitt `INIT:` des Programms gelöscht werden.

! Achtung: Wenn Sie schneller Daten in den FIFO schreiben als Sie auslesen, ist der FIFO irgendwann voll und es gehen Daten verloren.

Anwendungsbeispiel:

```
DIM DATA_20[1000] AS INTEGER AS FIFO
                          'Dimensioniert die globale
                          'Variable DATA_20 mit einer
                          'Länge von 1000 Elementen
                          'als Ringspeicher
```

Hinweis: Für den ADSP kann zusätzlich eine Angabe zu der Speicherart gemacht werden (Seite 24/Tabelle 4-1).

FIFO_CLEAR

Syntax:

FIFO_CLEAR(*DatensatzNr*)

Beschreibung:

Der Befehl FIFO_CLEAR löscht den Inhalt des FIFO mit der Nummer *DatensatzNr*.

! Achtung: Die FIFOs werden beim Start nicht automatisch gelöscht. Deshalb sollten sie mit diesem Befehl im Abschnitt INIT: des Programms gelöscht werden.

Anwendungsbeispiel:

```
DIM DATA_1[20000] AS LONG AS FIFO
                          'Deklaration

INIT:
FIFO_CLEAR(1)             'FIFO löschen

EVENT:
IF (FIFO_EMPTY(1) > 1) THEN
                          'Anzahl der freien Plätze
                          'im FIFO abfragen
  DATA_1 = ADC(1)         'Analogen Eingang 1
                          'messen und im FIFO
                          'speichern
ENDIF
```

FIFO_EMPTY

Syntax:

Wert = FIFO_EMPTY(*DatensatzNr*)

Beschreibung:

Der Befehl FIFO_EMPTY ermittelt die Anzahl der freien Speicherplätze im FIFO mit der Nummer *DatensatzNr*.

Hinweis: Bevor Daten in den FIFO geschrieben werden, sollten Sie mit diesem Befehl überprüfen, ob noch genügend Platz im FIFO frei ist.

Anwendungsbeispiel:

```
DIM DATA_1[20000] AS LONG AS FIFO
                          'Deklaration

EVENT:
IF (FIFO_EMPTY(1) > 1) THEN
                          'Anzahl der freien Plätze
                          'im FIFO abfragen
  DATA_1 = ADC(1)         'Analogen Eingang 1
                          'messen und im FIFO
                          'speichern
ENDIF
```

FIFO_FULL

Syntax:

Wert = FIFO_FULL(*DatensatzNr*)

Beschreibung:

Der Befehl FIFO_FULL ermittelt die Anzahl der belegten Speicherplätze im FIFO mit der Nummer *DatensatzNr*.

Hinweis: Bevor Daten aus dem FIFO gelesen werden, sollten Sie mit diesem Befehl überprüfen, ob im FIFO noch Daten enthalten sind.

Anwendungsbeispiel:

```
DIM DATA_1[20000] AS LONG AS FIFO
                          'Deklarationen

EVENT:
IF (FIFO_FULL(1) > 0) THEN
                          'Abfrage, ob Daten im FIFO
                          'enthalten sind
  DAC(1, DATA_1)          'Einen Wert aus dem FIFO
                          'auf dem analogen Ausgang1
                          'ausgeben
ENDIF
```

FOR ... NEXT

Syntax:

```
FOR i = X TO Y {STEP Z}
  Anweisungsblock
NEXT i
```

Beschreibung:

Der *Anweisungsblock* innerhalb der FOR ... NEXT-Schleife wird solange wiederholt ausgeführt, bis der Wert von *X* größer als der Wert *Y* ist. Nach jedem Durchlauf wird der Wert *X* um *Z* erhöht.

Hinweise: Die Anweisung STEP *Z* kann entfallen. Für *Z* wird dann der Wert 1 angenommen.

Für *X, Y* und *Z* können konstante Zahlenwerte oder Variablen eingesetzt werden. Diese müssen vom Typ Integer sein, wobei *Z* nur positive Werte annehmen darf.

Im Gegensatz zu anderen Programmiersprachen wird bei Basic der *Anweisungsblock* mindestens einmal ausgeführt, selbst wenn *X* größer als *Y* ist.

Auch die Zählvariable (hier *i*) muß im Programmkopf als Integer-Variable deklariert werden.

! **Achtung:** In einem hoch priorisierten Prozeß kann die FOR ... NEXT-Schleife von keinem anderen Prozeß unterbrochen werden. Dadurch ist es dem Prozessor des ***ADwin***-Systems während der Zeit der Schleifenabarbeitung nicht möglich, auf andere Events zu reagieren. Die FOR ... NEXT-Schleife darf deshalb in hoch priorisierten Prozessen nur verwendet werden, wenn die Anzahl der Schleifendurchläufe niedrig gehalten wird.

FUNCTION ... ENDFUNCTION

Syntax:

FUNCTION *Name*(*Wert1, Wert2, ...*) AS *Type*
... (Befehle)
ENDFUNCTION

Beschreibung:

Die Funktion *Name* wird definiert. Beim Aufruf wird in die Variablen *Wertx* ein Zahlenwert übergeben und die Befehle zwischen FUNCTION und ENDFUNCTION ausgeführt. Der Datentyp in dem die Werte der Funktion zurückgegeben werden, wird mit AS *Type* bestimmt.

Hinweise: Am Anfang jeder Funktion können lokale Variablen definiert werden.

Eine Funktion kann an den Anfang des ***ADbasic***-Programms (vor den INIT-Block), ans Ende des ***ADbasic***-Programms (nach dem FINISH-Block) oder in eine eigene Datei geschrieben werden, die Sie mit einer INCLUDE-Anweisung einbinden.

Anwendungsbeispiel:

```
FUNCTION Mittelwert(w1, w2, w3) AS FLOAT
  DIM sum AS FLOAT       'Berechnet den Mittelwert
  sum = w1 + w2 + w3     'aus den 3 Werten w1, w2
  Mittelwert = sum/3     'und w3
ENDFUNCTION
```

Der Aufruf der Funktion Mittelwert erfolgt mit der Programmzeile:

```
x = Mittelwert (x1, x2, x3)
```

IF ... THEN ... {ELSE}

Syntax:

```
IF (Bedingung) THEN
  Anweisungsblock
{ELSE}
  {Anweisungsblock}
ENDIF
```

oder

```
IF (Bedingung) THEN Anweisung
```

Beschreibung:

Die Kontrollstruktur IF erlaubt es Ihnen, den *Anweisungsblock* in Abhängigkeit von *Bedingung* auszuführen.

Anwendungsbeispiel:

```
DIM wert AS INTEGER      'Deklaration

EVENT:
wert = ADC(1)            'Meßwerterfassung
IF (wert > 3000) THEN    'Kontrollstruktur Anf.
  CLEAR_DIGOUT(1)        'Rücksetzen DIGOUT 1
  SET_DIGOUT(0)          'Setzen DIGOUT 0
ELSE
  CLEAR_DIGOUT(0)        'Rücksetzen DIGOUT 0
  SET_DIGOUT(1)          'Setzen DIGOUT 1
ENDIF                    'Kontrollstruktur Ende
```

INC

Syntax:

`INC(wert)`

Beschreibung:

Der Befehl `INC` inkrementiert den übergebenen `wert` um 1

Hinweise: Dieser Befehl darf nur auf den Datentyp INTEGER angewendet werden.

Die Anweisung `INC(wert)` führt zum gleichen Ergebnis wie die Programmzeile: `wert=wert+1`, benötigt jedoch eine geringere Ausführungszeit.

Anwendungsbeispiel:

```
DIM index AS INTEGER
DIM DATA_1[1000] AS INTEGER

INIT:
index=1

EVENT:
DATA_1[index] = ADC(1)   'Messwert im DATA ablegen
INC(index)               'index um 1 erhöhen
IF (index>1000) THEN END 'Nach 1000 Messungen wird
                         'das Programm beendet
```

#INCLUDE

Syntax:

```
#INCLUDE DateiName
```

Beschreibung:

Die Datei *DateiName* wird mit allen dort vorhandenen Definitionen und Programmen eingebunden. *DateiName* sollte auch die vollständige Pfadangabe enthalten. Anderenfalls sucht ***ADbasic*** im Standard-Include-Verzeichnis (s. Menü **Options ⇨ Directory**).

Hinweise: Die INCLUDE-Anweisung muß ganz am Anfang des ***ADbasic***-Programms stehen.

Sie sollten immer den vollständigen Pfadnamen angeben, da ansonsten nur im gerade aktuellen Verzeichnis gesucht wird.

Anwendungsbeispiel:

```
#INCLUDE C:\ADBASIC3\demofunc.inc
```

LINKIN

Syntax:

LINKIN(*Kanal, Wert, Anzahl*)

Beschreibung:

Der Befehl LINKIN liest von der Link-Schnittstelle *Kanal* so viele Bytes, wie durch *Anzahl* vorgegeben werden. Die gelesenen Bytes werden in der Variablen oder dem Datensatz *Wert* abgelegt.

! **Achtung:** Da der einzig verfügbare Link 0 des ADSP21062 für die Kommunikation mit dem PC/***ADlink*** bzw. ***ADpcmcia*** verwendet wird, ist dieser Befehl für diesen Prozessor nicht implementiert.

Zeitverhalten:

Der Befehl LINKIN unterbricht den laufenden Prozeß solange, bis alle *Anzahl* Bytes übertragen wurden. Daher sollte dieser Befehl nicht während einer laufenden Messung eingesetzt werden.

Hinweise: Der Prozessor des ***ADwin***-Systems wurde für den Aufbau von Multiprozessor-Systemen entwickelt. Um den Datenaustausch zwischen den Prozessoren zu vereinfachen, verfügt jedes Prozessormodul über serielle Link-Schnittstellen. Die LINK-Befehle dienen zum Datentransfer über diese Schnittstellen und sind von 0 bis 3 durchnumeriert.

Weitere Informationen zu Konfiguration und Anwendung entnehmen Sie bitte dem Hardware-Handbuch zu Ihrer ***ADwin***-Karte oder dem Hardware-Handbuch zu ***ADwin-Pro***.

...Fortsetzung Befehl `LINKIN:`

Anwendungsbeispiel:

```
DIM wert as LONG

EVENT:
LINKIN(0, Wert, 4)          'Liest einen Long-Integer-
                            'Wert vom LINK 0
```

LINKOUT

Syntax:

LINKOUT(*Kanal, Wert, Anzahl*)

Beschreibung:

Der Befehl LINKOUT gibt über die Link-Schnittstelle *Kanal* so viele Bytes aus, wie durch *Anzahl* vorgegeben werden. Die auszugebenden Bytes befinden sich in der Variablen oder dem Datensatz *Wert*.

! **Achtung:** Da der einzig verfügbare Link 0 des ADSP21062 für die Kommunikation mit dem PC/***ADlink*** bzw. ***ADpcmcia*** verwendet wird, ist dieser Befehl für diesen Prozessor nicht implementiert.

Zeitverhalten:

Der Befehl LINKOUT unterbricht den laufenden Prozeß solange, bis alle *Anzahl* Bytes übertragen wurden. Daher sollte dieser Befehl nicht während einer laufenden Messung eingesetzt werden.

Hinweise: Der Prozessor des ***ADwin***-Systems wurde für den Aufbau von Multiprozessor-Systemen entwickelt. Um den Datenaustausch zwischen den Prozessormodulen zu vereinfachen, verfügt jeder Prozessor über serielle Link-Schnittstellen. Die LINK-Befehle dienen zum Datentransfer über diese Schnittstellen und sind von 0 bis 3 durchnumeriert.

Weitere Informationen zu Konfiguration und Anwendung entnehmen Sie bitte dem Hardware-Handbuch zu Ihrer ***ADwin***-Karte oder dem Hardware-Handbuch zu ***ADwin-Pro***.

...Fortsetzung Befehl LINKOUT:

Anwendungsbeispiel:

```
DIM wert as LONG

EVENT:
LINKOUT(2, wert, 4)       'Gibt einen Long-Integer-
                          'Wert auf 'LINK 2 aus
```

LOG

Syntax:

Wert2 = LOG(*Wert1*)

Beschreibung:

Die Funktion LOG liefert den Logarithmus eines Arguments.

Zeitverhalten:

FLOAT: 1,45µs

FLOAT: 35 µs

FLOAT: 145 µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = 5.3
wert2 = LOG(wert1)        'Ergebnis:
                          'wert2 = 0.724...
```

NOT

Syntax:

Wert2 = NOT(*Wert1*)

Beschreibung:

Invertiert die einzelnen Bits von *Wert1*.

Hinweis: *Wert1* sollte vom Typ INTEGER oder LONG sein. Falls *Wert1* vom Typ FLOAT ist, wird *Wert1* zuvor in LONG konvertiert. Dabei ist zu beachten, dass Nachkommastellen abgeschnitten werden, und FLOAT und LONG unterschiedlich grosse Wertebereiche haben.

Anwendungsbeispiel:

```
DIM wert1 AS LONG
DIM wert2 AS LONG

wert1 = 11111111111111111111111111111101B ' = -3
wert2 = NOT(wert1)
                      'Ergebnis: wert2 = 010B = 2
```

Hinweis: NOT kann nicht zur logischen Negation verwendet werden.

Beispiel:

```
IF ( NOT( PAR_2 > 2 ) ) THEN
   ...
```

FALSCH !!

OR

Syntax:

Wert3 = *Wert1* OR *Wert2*

oder bei IF ... THEN und DO ... UNTIL

Ausdruck1 OR *Ausdruck2*

Beschreibung:

Der Operator OR wird vom Compiler automatisch entweder als bitweiser Operator oder als logischer Operator interpretiert.

Als **bitweiser Operator** vergleicht er die einzelnen Bits von zwei Werten miteinander. Im Resultat dieser Operation steht nur bei denjenigen Bits eine 1, an denen bei mindestens einem der beiden Werte an den entsprechenden Bitpositionen eine 1 enthalten ist.

Als **logischer Operator** innerhalb der *Bedingung* von IF ... THEN oder DO ... UNTIL Strukturen ermittelt er die Aussage wahr (1) oder falsch (0) für die ODER- Verknüpfung von zwei Ausdrücken.

Anwendungsbeispiel (als bitweiser Operator):

```
DIM wert1, wert2, wert3 AS LONG

wert1 = 0100B
wert2 = 0110B
wert3 = wert1 OR wert2
                          'Ergebnis: wert3 = 0110B
```

Hinweis: Als bitweiser Operator nur für Integer- und Long-Variablen bzw. Konstanten.

...Fortsetzung Befehl OR:

Anwendungsbeispiel (als logischer Operator):

```
DIM x AS LONG
DIM wert4 AS LONG

x = 15
IF ((x < 9) OR (x > 3)) THEN
  wert4 = 1
ELSE
  wert4 = 0
ENDIF
                          'Ergebnis: wert4 = 1
```

Hinweis: Falls mehrere OR (oder AND) Operatoren in einer Zeile verwendet werden, sind entsprechend viele Klammern zu setzen.

PEEK

Syntax:

Wert = PEEK(*Adresse*)

Beschreibung:

Der Befehl PEEK liest den Wert, der an *Adresse* gespeichert ist.

Anwendungsbeispiel:

```
Wert = PEEK(20400030H)   'Liest den Wert der
                         'Speicheradresse
                         '20400030H.
```

Hinweis: An der in dem Anwendungsbeispiel verwendeten Adresse 20400030H befindet sich bei den ***ADwin***-Karten mit ADSP-Prozessor und beim ***ADwin-GOLD***-System das Datenregister des ADC1.) Weitere Informationen zu Konfiguration und Anwendung entnehmen Sie bitte dem Hardware-Handbuch zu Ihrer ***ADwin***-Karte bzw. ***ADwin-GOLD***-System.

POKE

Syntax:

POKE(*Adresse*, *Wert*)

Beschreibung:

Der Befehl POKE speichert *Wert* an der Speicherstelle *Adresse*.

Anwendungsbeispiel:

```
POKE(20400050H, 10000)    'Speichert 10000 an der
                          'Adresse 20400050H
POKE(20400060H, 20000)    'Speichert 20000 an der
                          'Adresse 20400060H
START_CONV(4)             'startet gleichzeitig alle
                          'DACs
```

Hinweise: An der in dem Anwendungsbeispiel verwendeten Adressen 20400050H und 20400060H befinden sich beim ***ADwin-GOLD***-System die Datenregister des DAC1 und des DAC2. Somit entspricht dieses Beispiel prinzipiell der Befehlsfolge `DAC(1,10000)` `DAC(2,20000)` mit dem wesentlichen Unterschied, daß bei den `DAC` Befehlen die Konvertierung in einem kurzen Zeitabstand hintereinander gestartet wird, im andern Fall wird exakt zum gleichen Zeitpunkt gestartet. (Übrigens benötigt der `DAC` Befehl etwas mehr Ausführungszeit, da er den auszugebenden Wert auf Grenzwerte überprüft).

Weitere Informationen zu Konfiguration und Anwendung sowie weitere wichtige *Adressen* entnehmen Sie bitte dem Hardware-Handbuch zu Ihrer ***ADwin***-Karte bzw. ***ADwin***-System.

...Fortsetzung Befehl POKE:

! **Achtung:** Der Befehl POKE schreibt die Daten direkt an die angegebene Speicheradresse. Die an dieser Adresse befindlichen Daten werden gelöscht. Falls sich dort Programmdaten befinden, wird das Programm dadurch zerstört.

READ_TIMER

Syntax:

Wert = READ_TIMER()

Beschreibung:

Der Befehl READ_TIMER liest den Timerstand des aktuellen Prozesses aus.

Hinweise: Alle in ***ADwin***-Systemen eingesetzten Prozessoren verfügen über zwei integrierte Timer.

Bei den Prozessoren T400, T450 und T805 wird der Timerstand beim hoch priorisierten Prozeß in jeder Mikrosekunde um eins erhöht. Bei einem niedrig priorisierten Prozeß geschieht dies alle 64 µs.

Bei dem Prozessor ADSP wird der Timerstand beim hoch priorisierten Prozeß alle 25 ns um eins erhöht. Bei einem niedrig priorisierten Prozeß geschieht dies alle 100 µs.

Weitere Hinweise und Beispiele finden Sie in dem Kapitel 5.2 „Ermitteln der Ablaufzeiten mit Timerfunktionen“.

Anwendungsbeispiel:

```
DIM timerstand AS INTEGER

EVENT:
timerstand = READ_TIMER()
```

READADC

Syntax:

Wert = READADC(*AdcNr*)

***ADwin-* Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)-***Karten.

Bei ***ADwin-Pro*** gibt es auch einen Befehl mit diesem Namen, für den diese Beschreibung NICHT gilt. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl READADC liest einen Wert vom A/D-Wandler mit der Nummer *AdcNr*.

Hinweise: Diese Funktion spricht direkt einen der beiden A/D-Wandler ADC1 oder ADC2 bzw. ADC16-1 oder ADC16-2 an. Für *AdcNr* sind *nur* die Werte 1 oder 2 gültig.

Beachten Sie bitte, daß sich auf einer ***ADwin-light***-Karte nur ein ADC befindet!

Anwendungsbeispiel:

```
DIM wert1, wert2 AS INTEGER

EVENT:
SET_MUX(9)                  'Multiplexer für beide
                            'ADCs setzen
4 µs (ADwin-GOLD 6,5 µs) auf das Einschwingen des
                            Multiplexers warten
START_CONV(3)               'AD-Wandlung für beide
                            'ADCs starten
WAIT_EOC(3)                 'Auf das Ende der Wand-
                            'lungen beider ADCs warten
wert1 = READADC(1)          'Wert von ADC1 einlesen
wert2 = READADC(2)          'Wert von ADC2 einlesen
```

...Fortsetzung Befehl READADC:

Umrechnung der ADC-Werte bei 16 Bit ADCs:

Die auf dem ***ADwin-GOLD***-System verwendeten 16 Bit ADCs teilen den Meßbereich (von 20 Volt) in 65536 gleich große Bereiche ein.

Für die Umrechnung auf die Eingangsspannung gilt folgende Formel:

$$Spannung = (Digits - 32768_{bipolar}) * \frac{Meßbereich}{65536 * Verstärkung}$$

Für den Fall, daß die Verstärkung gleich 1 gewählt wurde, gelten die in der Tabelle angegebenen Werte.

Eingangs-spannungs-bereich	**READADC-Wert**			
	0	32768	65535	1Digit
-10...+10 V	-10 V	0 V	+9,999695 V	305,175 µV

Umrechnung der ADC-Werte bei 12 Bit ADCs:

Bei den ***ADwin***-Karten mit 12 Bit ADCs wird der gewählte Messbereich in 4096 gleich große Bereiche eingeteilt.

Für die Umrechnung auf die Eingangsspannung gilt folgende Formel:

$$Spannung = (Digits - 2048_{bipolar}) * \frac{Meßbereich}{4096 * Verstärkung}$$

Hinweis: Bei unipolarer Einstellung fällt der Offset von 2048 weg.

...Fortsetzung Befehl READADC**:**

Für den Fall, daß die Verstärkung gleich 1 gewählt wurde, gelten die in der Tabelle angegebenen Werte.

Eingangs-spannungs-bereich	**READADC-Wert**			
	0	2048	4095	1Digit
0...+10 V	0 V	+5 V	+9,99756 V	2,44 mV
-5...+5 V	-5 V	0 V	+4,99756 V	2,44 mV
-10...+10 V	-10 V	0 V	+9,99512 V	4,88 mV

READADC12

Syntax:

Wert = READADC12(*AdcNr*)

***ADwin*- Systeme:**

Nur für ***ADwin-GOLD***.

Beschreibung:

Der Befehl READADC12 liest einen Wert vom 12 Bit A/D-Wandler mit der Nummer *AdcNr*.

Hinweise: Diese Funktion spricht direkt einen der beiden A/D-Wandler ADC12-1 oder ADC12-2 an.
Für *AdcNr* sind *nur* die Werte 1 oder 2 gültig.

Anwendungsbeispiel:

```
DIM wert1, wert2 AS INTEGER

EVENT:
SET_MUX(9)                  'Multiplexer für beide
                            'ADCs setzen

 1,5 µs auf das Einschwingen der Multiplexer warten

START_CONV(24)              'AD-Wandlung für beide 12
                            'Bit ADCs starten
WAIT_EOC(24)                'Auf das Ende der
                            'Wandlungen der beiden
                            '12 bit ADCs warten
wert1 = READADC12(1)        'Wert von ADC12-1 einlesen
wert2 = READADC12(2)        'Wert von ADC12-2 einlesen
```

...Fortsetzung Befehl READADC12:

Umrechnung der ADC-Werte bei den *ADwin-GOLD* 12 Bit ADCs:

Die auf dem ***ADwin-GOLD***-System verwendeten 12 Bit ADCs teilen den Meßbereich (von 20 Volt) in 4096 gleich große Bereiche ein. Um den Vergleich mit den Messwerten der 16 Bit ADCs zu vereinfachen, gibt der Befehl READADC12 das Ergebnis in den höherwertigen Bits (Bits 31 bis 4) zurück. Somit liefert der Befehl READADC12(1) in den höherwertigen Bits das gleiche Ergebnis wie der (16 Bit) Befehl READADC(1). Die vier niederwertigsten Bits haben stets den Wert 0.

Für die Umrechnung auf die Eingangsspannung gilt folgende Formel:

$$Spannung = (Digits - 32768_{bipolar}) * \frac{Meßbereich}{65536 * Verstärkung}$$

Für den Fall, daß die Verstärkung gleich 1 gewählt wurde, gelten die in der Tabelle angegebenen Werte.

Eingangs-spannungs-bereich	**READADC12-Wert**			
	0	32768	65520	16Digits
-10...+10 V	-10 V	0 V	+9,99512 V	4,88 mV

REM

Syntax:

REM *Kommentar*

Beschreibung:

Der gesamte Text, der hinter REM in derselben Zeile steht, wird beim Kompilieren von ***ADbasic*** ignoriert. Dadurch haben Sie die Möglichkeit, in das ***ADbasic***-Programm Kommentare einzufügen.

Hinweise: Der Befehl REM gilt nur für die Zeile, in der er benutzt wird. Benötigt ein Kommentar mehr als eine Textzeile, so müssen Sie jede einzelne Zeile mit der Anweisung REM beginnen.

Eine Kommentarzeile kann auch durch das einfache Anführungszeichen ' eingeleitet werden.

Wenn der Kommentar in der gleichen Zeile untergebracht werden soll, in der bereits eine ***ADbasic***-Anweisung steht, dann müssen Sie bei der Verwendung von REM vorher einen Doppelpunkt setzen. Verwenden Sie das Anführungszeichen, dann entfällt der Doppelpunkt.

Anwendungsbeispiele:

```
REM Dies ist ein Kommentar, der mehr als eine
REM Textzeile Platz benötigt

'Dies ist auch eine Kommentarzeile

DIM min AS INTEGER       :REM Variable für Min_wert

DIM max AS INTEGER       'Variable für Max_wert
```

SET_DIGOUT

Syntax:

SET_DIGOUT(*AusgangNr*)

***ADwin-* Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)-***Karten.

Bei ***ADwin-Pro*** darf dieser Befehl nicht verwendet werden. Dort gibt es einen Befehl mit den Namen DIGOUT. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl SET_DIGOUT setzt den digitalen Ausgang *AusgangsNr*.

Hinweise: Die digitalen Ausgänge sind beim ***ADwin-GOLD***-Systems (in der Standardkonfiguration) von 16 bis 31, bei der ***ADwin***-Karte von 0 bis 15 durchnumeriert. Für die *AusgangsNr* muß in beiden Fällen eine konstante Zahl zwischen 0 und 15 eingesetzt werden. Beachten Sie aber bitte, daß sich auf einer ***ADwin-light***-Karte nur sechs digitale Ausgänge befinden und, daß die 16 digitalen Ausgänge auf der ***ADwin***-Karte nur über den mitgelieferten I/O-Erweiterungsstecker benutzt werden können.

In dem Befehl SET_DIGOUT dürfen Sie keine Variablen verwenden. Soll der zu setzende Ausgang durch eine Variable bestimmt werden, dann verwenden Sie den Befehl DIGOUT_WORD.

Beim ***ADwin-Gold***-System müssen die Ausgänge zuvor einmalig mit dem Befehl CONF_DIO(12) konfiguriert werden.

...Fortsetzung Befehl SET_DIGOUT:

Anwendungsbeispiel:

```
DIM Wert AS INTEGER      'Deklaration

INIT:
CONF_DIO(12)             'Dig. Ausgänge konfigu-
                         'rieren (nur ADwin-GOLD)

EVENT:
Wert = ADC(1)            'Meßwerterfassung
IF (Wert > 3000) THEN
  SET_DIGOUT(0)          'Dig. Ausgang DIO 16
                         '(ADwin-GOLD) bzw. 0
                         '(ADwin-Karten) setzen
ENDIF
```

SET_MUX

Syntax:

SET_MUX(*Auswahl*)

***ADwin-* Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)-***Karten.

Bei ***ADwin-Pro*** gibt es auch einen Befehl mit diesem Namen, für den diese Beschreibung NICHT gilt. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl SET_MUX setzt die Multiplexer und die Verstärker auf den jeweiligen Eingang. Da die Multiplexer 8 Eingänge haben, werden 3 Bits verwendet. Die Verstärker haben 4 Einstellungsmöglichkeiten (1-, 2-, 4- oder 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).

Hinweis: Bitte achten Sie darauf, daß der Multiplexer etwa 4 µs (***ADwin-GOLD*** 16 bit : 6,5µs bzw. 12 bit : 1,5µs) benötigt, bis er eingeschwungen ist. Zwischen der Neueinstellung der Multiplexer und dem Konvertierungsbeginn müssen Sie deshalb eine entsprechende Wartezeit einhalten oder eine Anweisung einfügen, deren Bearbeitungszeit die Multiplexer Einschwingzeit nicht unterschreitet.

Beachten Sie bitte, daß sich auf einer ***ADwin-light***-Karte nur ein ADC befindet und daß für dessen Eingänge keine Verstärkung eingestellt werden kann.

Beachten Sie außerdem, daß die 8 analogen Eingänge pro Multiplexer auf der ***ADwin***-Karte nur über den mitgelieferten I/O-Erweiterungsstecker benutzt werden können.

...Fortsetzung Befehl `SET_MUX`:

Die für die Einstellungen maßgeblichen Bitkombinationen ersehen Sie aus der folgenden Tabelle:

Funktion	Verstärkung ADC 2			Verstärkung ADC 1			Multiplexer ADC 2					Multiplexer ADC 1				
Bit	Verstärkung	9	8	Verstärkung	7	6	Kanal	Eingang	5	4	3	Kanal	Eingang	2	1	0
	1	0	0	1	0	0	1	2	0	0	0	1	1	0	0	0
	2	0	1	2	0	1	2	4	0	0	1	2	3	0	0	1
	4	1	0	4	1	0	3	6	0	1	0	3	5	0	1	0
	8	1	1	8	1	1	4	8	0	1	1	4	7	0	1	1
							5	10	1	0	0	5	9	1	0	0
							6	12	1	0	1	6	11	1	0	1
							7	14	1	1	0	7	13	1	1	0
							8	16	1	1	1	8	15	1	1	1

...Fortsetzung Befehl SET_MUX:

Berechnungsbeispiele:

Sie möchten den Multiplexer für ADC1 auf Kanal 3 stellen und benötigen Verstärkung 8 *und* gleichzeitig den Multiplexer für ADC2 auf Kanal 4 bei einer Verstärkung von 2:

Bitmuster: **01 11 011 010** Befehl: **SET_MUX(474)**

Alternativ zu dieser Schreibweise im DEZ-Code lautet die Schreibung im BIN-Code: **SET_MUX(0111011010B)**

Anwendungsbeispiel:

```
DIM wert1 AS INTEGER      'Deklaration

EVENT:
SET_MUX(0)                'Multiplexer für ADC1
                          '(und ADC2) setzen
 4 µs (ADwin-GOLD 6,5 µs) auf das Einschwingen des
                          Multiplexers warten
START_CONV(1)             'Start AD-Wandlung ADC1
WAIT_EOC(1)               'Warten auf Wandlungsende
                          'für ADC1
wert1 = READADC(1)        'Wert vom ADC1 einlesen
```

SHIFT_LEFT

Syntax:

Wert2 = SHIFT_LEFT(*Wert1*, *Anzahl*)

Beschreibung:

Der Befehl SHIFT_LEFT verschiebt alle Bits von *Wert1* um *Anzahl* Stellen nach links. Die leere Stelle rechts wird mit 0 aufgefüllt.

Hinweise: *Wert1* sollte vom Typ INTEGER oder LONG sein. Falls *Wert1* vom Typ FLOAT ist, wird *Wert1* zuvor in LONG konvertiert. Dabei ist zu beachten, dass Nachkommastellen abgeschnitten werden, und FLOAT und LONG unterschiedlich grosse Wertebereiche haben.

Dieser Befehl kann dazu verwendet werden, um die Variable *Wert1* mit ganzzahligen Vielfachen der Zahl 2 zu multiplizieren. Die Ausführungszeit ist dabei geringer als bei einem vergleichbaren Multiplikationsbefehl, d.h. *Wert2* = SHIFT_LEFT(*Wert1,3)* ist schneller als *Wert2 = Wert1 * 8* .

Anzahl	**Multiplikator**
1	2
2	4
3	8
...	...

Anwendungsbeispiel:

```
DIM Wert1,Wert2 AS LONG

Wert1 = 1024
Wert2 = SHIFT_LEFT(Wert1, 2)
                          ' Ergebnis: Wert2 = 4096
```

SHIFT_RIGHT

Syntax:

Wert2 = SHIFT_RIGHT(*Wert1, Anzahl*)

Beschreibung:

Der Befehl SHIFT_RIGHT verschiebt alle Bits von *Wert1* um *Anzahl* Stellen nach rechts. Die leere Stelle links wird mit 0 aufgefüllt.

Hinweis: *Wert1* sollte vom Typ INTEGER oder LONG sein. Falls *Wert1* vom Typ FLOAT ist, wird *Wert1* zuvor in LONG konvertiert. Dabei ist zu beachten, dass Nachkommastellen abgeschnitten werden, und FLOAT und LONG unterschiedlich grosse Wertebereiche haben.

Falls die Variable Wert1 eine positive Zahl ist, kann dieser Befehl dazu verwendet werden, um *Wert1* durch ganzzahlige Vielfache der Zahl 2 zu dividieren. Die Ausführungszeit ist dabei geringer als bei dem vergleichbaren Divisionsbefehl, d.h. *Wert2 = Wert1 / 8* ist langsamer als *Wert2* = SHIFT_RIGHT(*Wert1,3)* .

Anzahl	**Divisor**
1	2
2	4
3	8
...	...

Anwendungsbeispiel:

```
DIM Wert1,Wert2 AS LONG

Wert1 = 1024
Wert2 = SHIFT_RIGHT(Wert1, 3)
                          ' Ergebnis: Wert2 = 128
```

SIN

Syntax:

Wert2 = SIN(*Wert1*)

Beschreibung:

Die Funktion SIN liefert den Sinus eines Arguments, das im Bogenmaß angegeben ist.

Zeitverhalten:

FLOAT: 1,2µs

FLOAT: 24 µs

FLOAT: 72 µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = -5.3
wert2 = SIN(wert1)       'Ergebnis: wert2 = 0.83...
```

SQRT

Syntax:

Wert2 = SQRT(*Wert1*)

Beschreibung:

Die Funktion SQRT liefert die quadratische Wurzel von *Wert1*.

Zeitverhalten:

FLOAT: 0,775µs

LONG: 25µs

FLOAT: 14µs

FLOAT: 40 µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = 16
wert2 = SQRT(wert1)      'Ergebnis: wert2 = 4
```

START_CONV

Syntax:

START_CONV(*Auswahl*)

***ADwin*- Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)***-Karten.

Bei ***ADwin-Pro*** gibt es auch einen Befehl mit diesem Namen, für den diese Beschreibung NICHT gilt. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl START_CONV startet die mit *Auswahl* angegebenen ADCs und/oder DACs. Für die Auswahl der ADCs werden nur 4 von den 5 niedrigsten Bits von *Auswahl* verwendet: Bit 0 startet ADC1, Bit 1 startet ADC2.

Bit(s)	DEZ-Wert	*ADwin-GOLD*	*ADwin*-Karte	*ADwin-light*-Karte
0	1	ADC16-1	ADC1	ADC1
1	2	ADC16-2	ADC2	-
0,1	3	ADC16-1 ***und*** ADC16-2	ADC1 ***und*** ADC2	-
2	4	***alle*** DACs	***alle*** DACs	***alle*** DACs
0,2	5	ADC16-1 ***und alle*** DACs	ADC1 ***und alle*** DACs	ADC1 ***und alle*** DACs
3	8	ADC12-1	-	-
4	16	ADC12-2	-	-
3,4	24	ADC12-1 ***und*** ADC12-2	-	-
0,3	9	ADC16-1 ***und*** ADC12-1	-	-
1,4	18	ADC16-2 ***und*** ADC12-2	-	-
0,1,3,4	27	***alle*** ADCs	-	-

...Fortsetzung Befehl START_CONV:

Hinweis: Mit dem Setzen von Bit 2 kann man gleichzeitig alle DACs starten. Siehe hierzu das Beispiel zum Befehl Poke.

Beachten Sie bitte, daß es sich bei ADC1 und ADC2 entweder um einen 12 oder um einen 16 Bit Analog Digital Wandler handeln kann. Entnehmen Sie weitere Informationen hierzu bitte dem Hardware-Handbuch zu Ihrer ***ADwin-(light)-***Karte.

! Achtung: Beim ADSP muß *Auswahl* aus Gründen der Codeoptimierung eine Konstante sein. Bei allen anderen Prozessoren darf *Auswahl* auch eine Variable sein.

Berechnungsbeispiel:

Sie möchten nur ADC2 bzw. ADC16-2 starten:
Bitmuster: **10** Befehl: **START_CONV (2)**

Anwendungsbeispiel:

```
DIM wert1 AS INTEGER

EVENT:
SET_MUX(0)                  'Multiplexer für ADC1 bzw.
                            'ADC16-1 auf Kanal 1
                            'setzen
4 µs (ADwin-GOLD 6,5 µs) auf das Einschwingen des
                            Multiplexers warten
START_CONV(1)               'Start ADC1 A/D-Wandlung
WAIT_EOC(1)                 'Warten auf Wandlungsende
                            'für ADC1 bzw. ADC16-1
wert1 = READADC(1)          'Wert auslesen
```

START_PROCESS

Syntax:

START_PROCESS(*ProzessNr*)

Beschreibung:

Der Befehl START_PROCESS startet den ***ADbasic***-Prozeß mit der Nummer *ProzessNr*.

Hinweise: Der zu startende Prozeß muß vorher auf das ***ADwin***-System geladen worden sein.

Wenn der Prozeß bereits läuft, dann hat dieser Befehl keine Auswirkung.

Anwendungsbeispiel:

```
EVENT:
IF (ADC(1) > 3072) THEN
  START_PROCESS(2)       'Meßprozeß 2 starten,
                         'falls der Schwellwert
                         'überschritten ist
  END
ENDIF
```

STOP_PROCESS

Syntax:

STOP_PROCESS(*ProzessNr*)

Beschreibung:

Der Befehl STOP_PROCESS stoppt den ***ADbasic***-Prozeß mit der Nummer *ProzessNr*.

Hinweis: Wenn der Prozeß nicht läuft, hat dieser Befehl keine Auswirkung.

Anwendungsbeispiel:

```
EVENT:
IF (ADC(1) > 3072) THEN
  STOP_PROCESS(2)         'Meßprozeß 2 stoppen,
                          'falls der Schwellwert
                          'überschritten ist
  END
ENDIF
```

SUB ... ENDSUB

Syntax:

SUB *Name*(*Wert1, Wert2, ...*)
... (Befehle)
ENDSUB

Beschreibung:

Das Unterprogramm *Name* wird definiert. Beim Aufruf wird in die Variablen *Wertx* ein Zahlenwert übergeben und die Befehle zwischen SUB und ENDSUB ausgeführt.

Hinweise: Am Anfang jeder Subroutine können lokale Variablen definiert werden.

Ein Unterprogramm kann an den Anfang des ***ADbasic*** Programms (vor den INIT-Block), ans Ende des ***ADbasic***-Programms (nach dem FINISH-Block) oder in eine eigene Datei geschrieben werden, die Sie mit einer INCLUDE-Anweisung einbinden.

Anwendungsbeispiel:

```
SUB Fast_Dac1(wert1)
REM Gibt wert1 auf dem analogen Ausgang 1 aus
  POKE(50H, wert1)
  POKE(10H, 3)
ENDSUB
```

Der Aufruf des Unterprogramms *Name* erfolgt mit der Programmzeile:
Fast_Dac1(NeuerWert)

TAN

Syntax:

Wert2 = TAN(*Wert1*)

Beschreibung:

Die Funktion TAN liefert den Tangens eines Arguments, das im Bogenmaß angegeben ist.

Zeitverhalten:

FLOAT: 1,275µs

FLOAT: 30 µs

FLOAT: 150 µs

Anwendungsbeispiel:

```
DIM wert1, wert2 AS FLOAT

EVENT:
wert1 = 5.3
wert2 = TAN(wert1)        'Ergebnis:
                          'wert2 = -1.50...
```

VR6_CLEAR

Syntax:

VR6_CLEAR(*Auswahl*)

***ADwin*- Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-VR6*** ausgerüstet sind, und für ***ADwin-X-VR6-***Karten.

Beschreibung:

Der Befehl VR6_CLEAR löscht die durch *Auswahl* bestimmten Vorwärts-/Rückwärts-Zähler. Für jeden Zähler, der gelöscht werden soll, muß in *Auswahl* das entsprechende Bit gesetzt werden. Sie können mehrere Zähler gleichzeitig löschen. Die folgende Tabelle zeigt den Zusammenhang zwischen zu setzendem Bit und Zählernummer:

Zählernummer	6	5	4	3	2	1
Bit	5	4	3	2	1	0
DEZ-Wert	32	16	8	4	2	1

Berechnungsbeispiele:

Sie möchten den Zähler 5 löschen, d. h. Sie müssen Bit 4 setzen.
Bitmuster: **010000** Befehl: **VR6_CLEAR(16)**
Alternativ zu dieser Schreibweise in DEZ-Code lautet die Schreibung im BIN-Code: **VR6_CLEAR(010000B)**

Anwendungsbeispiel:

```
EVENT:
VR6_CLEAR(63)                 'löscht alle VR-Zähler
```

VR6_LATCH

Syntax:

VR6_LATCH(*Auswahl*)

***ADwin-* Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-VR6*** ausgerüstet sind, und für ***ADwin-X-VR6-***Karten.

Beschreibung:

Der Befehl VR6_LATCH übernimmt den aktuellen Zählerstand der durch *Auswahl* bestimmten Vorwärts-/Rückwärts-Zähler in das Latch. Für jeden Zähler muß in *Auswahl* ein Bit gesetzt werden. Sie können mehrere Zählerstände gleichzeitig übernehmen, da jeder Zähler sein eigenes Latch besitzt. Die folgende Tabelle zeigt den Zusammenhang zwischen zu setzendem Bit und Zählernummer:

Zählernummer	6	5	4	3	2	1
Bit	5	4	3	2	1	0
DEZ-Wert	32	16	8	4	2	1

Berechnungsbeispiele:

Sie möchten den Wert des 5. Zählers in das LATCH übernehmen, d. h. Sie müssen Bit 4 setzen.

Bitmuster: **010000** Befehl: **VR6_LATCH(16)**

Sie möchten gleichzeitig die Werte des 3. und 5. Zählers übernehmen, d. h. Sie müssen Bit 2 und Bit 4 setzen.

Bitmuster: **010100** Befehl: **VR6_LATCH(20)**

Anwendungsbeispiel:

```
EVENT:
VR6_LATCH(63)                 'Überträgt gleichzeitig
                              'alle Zählerwerte
```

VR6_READ

Syntax:

Wert = VR6_READ(*Nummer*)

***ADwin-* Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-VR6*** ausgerüstet sind, und für ***ADwin-X-VR6-***Karten.

Beschreibung:

Der Befehl VR6_READ liefert den Zählerstand des durch *Nummer* angegebenen Zählers. Dieser Befehl setzt das Latch vor dem Auslesen auf den aktuellen Wert.

Hinweise: Die Zähler sind von 1 bis 6 durchnumeriert.

Anwendungsbeispiel:

```
EVENT:
Wert = VR6_READ(3)          'Liefert den Wert des
                            'Zählers 3
```

VR6_READLATCH

Syntax:

Wert = VR6_READLATCH(*Nummer*)

***ADwin-* Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-VR6*** ausgerüstet sind, und für ***ADwin-X-VR6-***Karten.

Beschreibung:

Der Befehl VR6_READLATCH liest das Latch des durch *Nummer* übergebenen Zählers aus. Der aktuelle Wert des spezifizierten Zählers wird vorher *nicht* in das Latch übertragen.

Hinweise: Die Zähler sind von 1 bis 6 durchnumeriert.

Anwendungsbeispiel:

```
EVENT:
VR6_LATCH(63)              'Überträgt alle
                           'Zählerwerte gleichzeitig
                           'in die Latches
Wert = VR6_READLATCH(1)    'Liefert den Wert des
                           'Latches von Zähler 1
```

VR6_SETMODE

Syntax:

VR6_SETMODE(*Auswahl*)

***ADwin*- Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-VR6*** ausgerüstet sind, und für ***ADwin-X-VR6-***Karten.

Beschreibung:

Der Befehl VR6_SETMODE setzt die Zähler auf einen von zwei Modi. Bit nicht gesetzt (= 0) entspricht Vierfachflankenauswertung, d. h. zwei Drehimpulsgeber liefern um 90° phasenverschobene Drehzahlsignale, die zur Richtungserkennung ausgewertet werden. Bit gesetzt (= 1) entspricht einem Takt und einem Richtungseingang pro Zähler.

Bitte sehen Sie die Details dazu in den speziellen Unterlagen zu Ihrem Vorwärts-/Rückwärts-Zähler nach.

Die folgende Tabelle zeigt den Zusammenhang zwischen zu setzendem Bit und Zählernummer:

Zählernummer	6	5	4	3	2	1
Bit	5	4	3	2	1	0
DEZ-Wert	32	16	8	4	2	1

Berechnungsbeispiel:

Sie möchten den Modus des 5. Zählers auf 1 und die Modi aller anderen Zähler auf 0 setzen: Bit 4 setzen.

Bitmuster: **010000** Befehl: **VR6_SETMODE(16)**

Anwendungsbeispiel:

```
VR6_SETMODE(63)          'Setzt alle Zähler
                         'gleichzeitig auf Takt und
                         'Richtungssignal
```

VR6_START

Syntax:

VR6_START(*Auswahl*)

***ADwin-* Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-VR6*** ausgerüstet sind, und für ***ADwin-X-VR6-***Karten.

Beschreibung:

Der Befehl VR6_START startet die durch *Auswahl* bestimmten Vorwärts-/Rückwärts-Zähler auf der VR6-Zusatzplatine. Für jeden Zähler, der gestartet werden soll, muß in *Auswahl* ein Bit gesetzt werden. Sie können mehrere Zähler gleichzeitig starten. Die folgende Tabelle zeigt den Zusammenhang zwischen zu setzendem Bit und Zählernummer:

Zählernummer	6	5	4	3	2	1
Bit	5	4	3	2	1	0
DEZ-Wert	32	16	8	4	2	1

Berechnungsbeispiele:

Sie möchten Zähler 3 starten: Bit 2 setzen.
Bitmuster: **000100** Befehl: **VR6_START(4)**
Sie möchten gleichzeitig Zähler 3 und 5 starten: Bit 2 und 4 setzen.
Bitmuster: **010100** Befehl: **VR6_START(20)**

Anwendungsbeispiel:

```
EVENT:
VR6_START(63)                 'Startet alle VR-Zähler
```

VR6_STOP

Syntax:

VR6_STOP(*Auswahl*)

***ADwin-* Systeme:**

Nur für ***ADwin-(light)-***Karten, die mit der Zähleroption ***ADwin-VR6*** ausgerüstet sind, und für ***ADwin-X-VR6-***Karten.

Beschreibung:

Der Befehl VR6_STOP hält die durch *Auswahl* bestimmten Vorwärts-/Rückwärts-Zähler auf der VR6-Zusatzplatine an. Für jeden Zähler, der angehalten werden soll, muß in *Auswahl* ein Bit gesetzt werden. Sie können mehrere Zähler gleichzeitig anhalten. Die folgende Tabelle zeigt den Zusammenhang zwischen zu setzendem Bit und Zählernummer:

Zählernummer	6	5	4	3	2	1
Bit	5	4	3	2	1	0
DEZ-Wert	32	16	8	4	2	1

Berechnungsbeispiele:

Sie möchten gleichzeitig Zähler 3 und Zähler 5 anhalten: Bit 2 und Bit 4 setzen.

Bitmuster: **010100** Befehl: **VR6_STOP(20)**

Anwendungsbeispiel:

```
EVENT:
VR6_STOP(63)                'Hält alle VR-Zähler an
```

WAIT_EOC

Syntax:

WAIT_EOC(*Auswahl*)

***ADwin-* Systeme:**

Für ***ADwin-GOLD***, ***ADwin-(light)-***Karten.

Bei ***ADwin-Pro*** gibt es auch einen Befehl mit diesem Namen, für den diese Beschreibung NICHT gilt. Siehe hierzu das Dokument : „ADwin-Pro Systembeschreibung Programmierung in ADbasic“.

Beschreibung:

Der Befehl WAIT_EOC wartet, bis die durch *Auswahl* gekennzeichneten ADCs fertig konvertiert haben. Vier der fünf niedrigsten Bits von *Auswahl* werden zur Auswahl der Konverter verwendet: Bit 0 gilt für ADC1, Bit1 für ADC2.

Bit(s)	DEZ-Wert	*ADwin-GOLD*	*ADwin*-Karte	*ADwin-light*-Karte
0	1	ADC16-1	ADC1	ADC1
1	2	ADC16-2	ADC2	-
0,1	3	ADC16-1 ***und*** ADC16-2	ADC1 ***und*** ADC2	-
3	8	ADC12-1	-	-
4	16	ADC12-2	-	-
3,4	24	ADC12-1 ***und*** ADC12-2	-	-
0,3	9	ADC16-1 ***und*** ADC12-1	-	-
1,4	18	ADC16-2 ***und*** ADC12-2	-	-
0,1,3,4	27	***alle*** ADCs	-	-

...Fortsetzung Befehl WAIT_EOC:

Hinweis: Beachten Sie bitte, daß es sich bei ADC1 und ADC2 entweder um einen 12 oder um einen 16 Bit Analog Digital Wandler handeln kann. Entnehmen Sie weitere Informationen hierzu bitte dem Hardware-Handbuch zu Ihrer ***ADwin-(light)-***Karte.

Anwendungsbeispiel:

```
DIM wert AS INTEGER

EVENT:
SET_MUX(8)                  'Multiplexer für ADC2 auf
                            'Kanal 2 (Eingang 4)
                            'setzen

4 µs (ADwin-GOLD 6,5 µs) auf das Einschwingen des
                            Multiplexers warten

START_CONV(2)               'Start A/D-Wandlung ADC2
WAIT_EOC(2)                 'Auf Wandlungsende von
                            'ADC2 warten
wert = READADC(2)           'Wert auslesen
```

XOR

Syntax:

Wert3 = *Wert1* XOR *Wert2*

Beschreibung:

Verknüpft die einzelnen Bits von *Wert1* und *Wert2* mit EXKLUSIV-ODER.

Das Resultat dieser Operation kann der folgenden Wahrheitstabelle entnommen werden:

Falls Bit in Wert1 =	und Bit in Wert2 =	Resultat :
0	0	0
0	1	1
1	0	1
1	1	0

Hinweis: Da **bitweiser Operator**, nur für Integer- und Long-Variablen bzw. Konstanten.

Anwendungsbeispiel:

```
DIM wert3 AS LONG

wert3 = 0100B XOR 0110B
                          'Ergebnis: wert3 = 0010B
```

8 Was tun bei Problemen?

Wenn Sie bereits bei der Installation Probleme haben, ziehen Sie bitte die Dokumentation zu Ihrem ***ADwin***-System zu Rate. Überprüfen Sie, ob alle Einstellungen richtig und vollständig durchgeführt wurden. Kontrollieren Sie dann bitte anhand von Kapitel 2 (Software-Installation), ob die Software richtig installiert wurde. Prüfen Sie auch, ob im Menü **Options** die Basisadresse, der Prozessortyp etc. richtig sind. Sollten Ihre Probleme dann immer noch bestehen, rufen Sie uns bitte an.

Sollten Sie weitergehende Hilfe benötigen, so setzen Sie sich bitte direkt mit uns in Verbindung:

Jäger Computergesteuerte Messtechnik GmbH
Rheinstraße 4
D-64653 Lorsch

Tel.: (0 62 51) 9 63 20
Fax: (0 62 51) 5 68 19
E-Mail: info@adwin.de

9 Index

Sonderzeichen

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Z