ADbasic

Real-Time Development Tool for ADwin Systems

Version 3.0

License Key:

November 1999

Real-time processing, accurate to a microsecond
- quick and easy to program

Table of Contents

Dear reader!

This ***ADbasic 3.0*** manual is intended for ***ADwin*** board users who would like to create programs using ***ADbasic*** for fast real-time processing on the ***ADwin*** boards. You should be familiar with fundamental programming, e. g. in BASIC, before you use this manual.

Users of *ADwin-Pro* Systems:

The commands for getting access to the ***ADwin-Pro*** system with ***ADbasic*** can be found in specific INCLUDE files. It is recommended to read our documentation called: "***ADwin-Pro*** System Specifications. Programming in ***ADbasic***".

Users of *ADbasic 2.0*:

If you know already ***ADbasic 2.0***, please read chapter 1 (New Functions) first and then the chapters 4.3 (Memory management)) and 5.2 (Getting Execution Times with Timer Functions)

...Users who have already a wide experience in programming, but none with *ADbasic 2.0* are recommended to read additionally:

Chapter 2 (Software Installation), chapter 4.4 (***ADbasic*** Data Types) and chapter 4.2 (Structure of an ***ADbasic*** program).

Conventions used in this Manual

We use the following typographical conventions:

Example	Application
! Important:	Placed before a paragraph containing important information or explanations.
important	This font is used to highlight important points and for labeling technical terms and proper names.
♦ *Please enter*	At these sections you must enter something into your PC so that you can try out one of our examples.
`Typewriter`	We use this font for program commands and all other entries which you can make in the editor window. In the reference section the command being discussed is highlighted in bold print.
`italic`	This is used for variable names used in a program example.
`COMMAND`	***ADbasic*** commands are placed in uppercase letters to differentiate them. The form used in your program is left up to you since ***ADbasic*** also understands commands written in lowercase.

{COMMAND}	This part of a command is optional.
File ⇨ Save	The names of menus and submenus are printed in bold type. The arrow points to the nex lower menu level.
‘Delay’	The titles of entry or selection fields in dialog windows are placed in single quotation marks.
CAPS AND SMALL CAPS	Key names such as RETURN or CTRL.
Note:	Is placed in front of a paragraph with supplementary explanations.

1 New Functions

Contrary to the ***ADbasic*** version 2.0 the ***ADbasic*** version 3.0 has some more functions, shown as follows:

- ***ADbasic*** processes used for the processor module ADSP21062 from ANALOG DEVICES[1] in the ***ADwin*** systems[2] ***ADwin***, ***ADwin-light*** and ***ADwin-Pro*** can be compiled.
- The ***ADbasic*** dialog box "Parameters" meets all requirements for the ADSP-processor.
- ***ADbasic 3.0*** is now ready for operation under *Windows 95*, *Windows 98* and *Windows NT*, but no longer under Windows 3.1x

Note: For users who want to continue working with *Windows 3.1x* and ***ADbasic***, there is a subdirectory called win31 in the directory ADbasic on the CD-ROM.

In this subdirectory there is the ***ADbasic*** compiler, version 2.0, and a PDF-file (manual for ***ADbasic 2.0***).

1 This processor module is called ADSP in the following text .

2 The hardware systems ***ADwin***, ***ADwin-light*** (PC plug-in boards) and ***ADwin-Pro*** (19“-systems) are called ***ADwin*** systems in the following text, as far as the statements made apply to all hardware systems.

2 Software Installation

2.1 System Requirements

For the real-time development tool ***ADbasic 3.0*** you need a computer where *Windows 95*, *Windows 98* or *Windows NT* have been installed.

Moreover a CD-ROM drive is needed for installation of ***ADbasic 3.0***.

Note: It is possible for you to make yourself a set of ***ADbasic 3.0*** and ***ADwin*** driver disks in order to carry out the installation by disk. For this purpose, copy the directories Disk 1 to Disk 2 from the ***ADbasic*** directory or the directories Disk 1 to Disk 2 of the driver directory, being located on the CD-ROM.

Compiled ***ADbasic*** programs also run on a computer with *Windows 3.1x* when the ***ADwin*** driver and the Win32s extensions have been installed.

It is recommended to install one of the frequently used measurement data evaluation programs (such as *TestPoint*) which runs excellently in combination with ***ADwin*** and ***ADbasic***. But nevertheless, you can also write your own evaluation program, for instance with *Visual Basic*, *Visual C* or *Borland Delphi.*

The ***ADbasic*** development environment is a 32-bit Windows program which runs completely on the PC. An ***ADwin*** system is only needed for testing the ***ADbasic*** programs.

Although ***ADbasic*** does not require much memory, it will be better to have a fast CPU with a large memory capacity for the operation of measurement data evaluation programs. For further details please read the documentation of your application program.

2.2 Selecting the operating system

Version 3.0 of ***ADbasic*** is a 32-bit program and therefore requires *Windows 95*, *Windows 98* or *Windows NT*.

The setup-program identifies the operating system installed on your computer.

2.3 Installing the ADwin driver

Please insert the provided CD-ROM in the CD-ROM drive of your computer. The setup program starts automatically.[3] After choosing the language and the destination directory (it is recommended to confirm the default directory C:\ADBASIC3), the following files are copied to the specified directory.

ADWIN2.BTL	driver for the ***ADwin*** systems with T225 processor
ADWIN4.BTL	driver for the ***ADwin*** systems with T400 processor
ADWIN5.BTL	driver for the ***ADwin*** systems with T450 processor
ADWIN8.BTL	driver for the ***ADwin*** systems with T805 processor
ADWIN9.BTL	driver for the ***ADwin*** systems with ADSP-processor
TESTVE16.EXE	program to search and display all 16-bit ***ADwin*** DLLs installed on your system.

[3] If the setup-program does not start automatically, please carry out the program setup.exe which is located in the subdirectory: ...\Driver\Disk1.

TESTVE32.EXE	program to search and display all 32-bit ***ADwin*** DLLs installed on your PC
ADTEST.EXE	program to test the ***ADwin*** systems
ADWINSET.EXE	program to register the link address (only for *Windows NT*)
ADSERVER.EXE	program for initializing the ***ADwin*** network function

Depending on your operating system the following DLLs (Dynamic Link Libraries) are automatically copied to the Windows directory of your computer.

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

! Important: If you have already installed ***ADwin*** drivers of a previous ***ADbasic*** version on your PC then the setup-programm will ask you to delete all existing (DLL-) files. Confirm with "YES".

After the setup-program has finished correctly the message: "The setup was successful" appears.

! Important: If using *Windows NT* you have to restart your computer after driver installation!

2.3.1 Hardware installation when using Windows NT

All ***ADwin*** systems operate via link addresses, which are set by a DIP switch (we refer to our hardware manual). The default setting for the link address is 150H.

If you have installed *Windows NT* on your computer and want to use a link address different from the default setting, then you have to carry out the program *AdwinSet* which is installed in the program group ***ADwin***.

Note: When *AdwinSet* is carried out it has access to the registry data base. If you want to make changes here, you must have the administrator authority to do so.

In the dialogbox *AdwinSet* you can:

- enter up to eight link addresses to the registry database,
- enter link addresses,
- mark and change link addresses,
- delete link addresses.

After having confirmed the new entries in *AdwinSet*, you have to restart your system.

Note: Moreover, you have to indicate your link address in ***ADbasic*** (**Options ⇨ Compiler**) or respectively, in your own measurement data evaluation program.

2.4 Installing the ADbasic compiler

Please insert the CD-ROM into the CD-ROM drive of your computer. The setup program starts automatically.[4] Press the button '***ADbasic*** 3.0 Setup' After choosing the language and the destination directory (it is recommended to confirm the default directory C:\ADBASIC3), the following files are copied to the specified directory:

ADBASIC.EXE	***ADbasic*** compiler
ADBASICE.HLP	***ADbasic*** helpfile (English)
ADBASIC.HLP	***ADbasic*** helpfile (German)
ADSHOW.EXE	program for displaying the process parameters and variables
...\SAMPLES\	collection of ***ADbasic*** sample programs

[4] If the setup-program does not start automatically, please carry out the program setup.exe which is located in the subdirectory: ...\Driver\Disk1.

2.5 Loading the ADwin driver

The PC can only communicate with an ***ADwin*** system after the ***ADwin*** driver has been loaded. But in any case the ***ADwin*** driver must be loaded again after every power supply disconnection of the ***ADwin*** system.

! Important: When loading the ***ADwin*** driver all processes on the ***ADwin*** system will be deleted and all global variables are set to the value 0. When using an ADSP the value for the global delay is set o 25000 ns after loading the ***ADwin*** driver and to 1000 µs when using all other processors.

- *Start **ADbasic** by selecting in the Windows menu **Start ⇨ Programs ⇨ ADwin ⇨ ADbasic 3.0**.*

- *Check if the settings in the **ADbasic** menu **Options ⇨ Compiler** correspond to those of your **ADwin** system.*

- *Click this symbol in your icon spacing:*  *(optionally: **Project ⇨ Boot ADwin**)*

The successful loading of the driver is confirmed in the status line with the message: „ADwin is booted“.

3 Principles and operation methods of *ADbasic*

3.1 Introduction

ADbasic is a compiler whose programming language is very similar to BASIC, so that you can start using it very easily and quickly.

With ***ADbasic*** you produce programs for your ***ADwin*** system. The programs produced with ***ADbasic*** are processed only on these systems. So your system receives a real-time capability with extremely short response times. Together with evaluation programs, such as for instance *TestPoint* from Keithley, it forms a high performance system for simultaneous measurement and control under *MS Windows*.

The communication functions between the PC and the ***ADwin*** system are handled by ***ADbasic***. Therefore, you only need to concentrate on programming the cyclic sequence. Since the communication is bidirectional, measurement results can either be transferred from the ***ADwin*** system to the PC or control parameters and default values can be transferred from the PC to the ***ADwin*** system or process. In order to be able to operate as fast as possible, your ***ADwin*** program is not, as with many BASIC dialects, interpreted and then carried out, but first compiled and then transferred to the ***ADwin*** system where it is executed by the processor. Of course, you can also develop, test and compile programs so that they can be loaded to the ***ADwin*** system later on, for instance from your application program.

3.1.1 *ADbasic* and *ADwin*

With each ***ADwin*** system you have a transputer which can concentrate completely on measurement and control. This transputer has been developed and optimized for fast task changes. It is therefore able to react within a few microseconds. Using this system you can bypass the slow reactions and the lack of real-time

processing of *MS Windows*, but still make use of all its advantages, for instance the graphical interface or the easy handling.

With ***ADbasic*** you have a development tool with which you can develop measurement, control or monitoring programs. These processes are then integrated into the standard program which runs continuously on the ***ADwin*** system, handling all the communication and management tasks.. That means that the ADbasic programs rund *independent of* your PC which can concentrate on other tasks.

With ***ADbasic*** you can generate executable programs which you can load and start together with an evaluation program. Therefore, you only need ***ADbasic*** for the development of the programs and not for their execution. However, the programs compiled with ***ADbasic*** can only be executed on an ***ADwin*** system.

Note: The commands for calling an ***ADwin-Pro*** system with ***ADbasic*** are available in special INCLUDE-files. It is recommanded to read the documentation called: „***ADwin-Pro*** : System Specifications - Programming in ***ADbasic***".

3.1.2 What can you do with *ADbasic*?

With ***ADbasic*** you have an easy-to-use tool to write programs that are then executed at high speed from the processor of an ***ADwin*** system. You can then carry out real-time applications such as digital controllers, monitoring of setpoint variables, etc. A special advantage is that the response times to new events are extremely short when using a processor operating independent of the PC. Typical applications are:

- fast acquisition of measurement data up to sampling frequencies of 800 kHz
- development of fast digital controllers with sampling frequencies of up to 400 kHz
- simultaneous generation and measurement of analog signals, for instance for measurement of dynamic parameters.
- fast controlling and monitoring tasks with reliable response times of less than 300 ns

You can even run a number of processes simultaneously! To give you an impression of the execution speed and the ease of programming, we would like to show you the example of the proportional controller in Fig. 3-1.

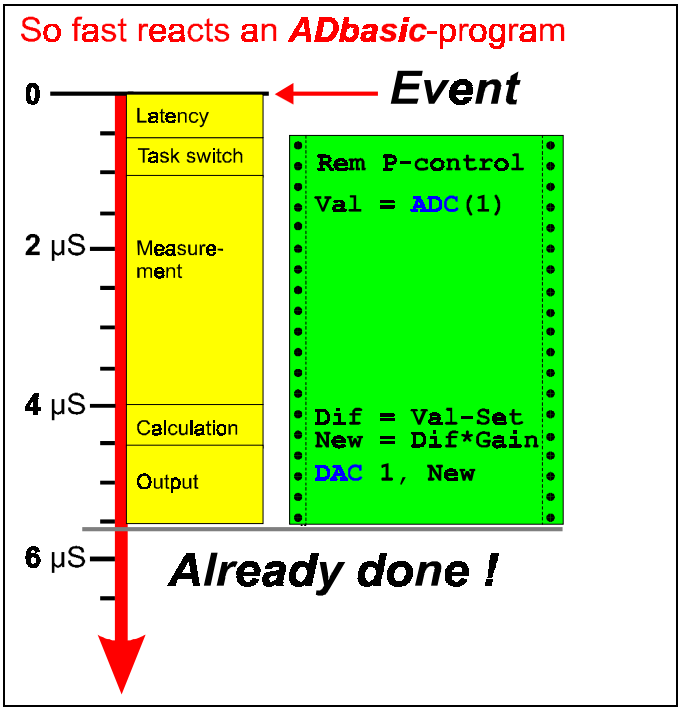


Fig. 3-1: The event concept shown at the example of a proportional controller

Here you can see that it is possible to get a proportional controller with only four program lines! If you set the internal timer so that an event signal is generated every 20 µs, you obtain a frequency of 50 kHz for the execution of the program.

4 Programming in *ADbasic*

In this chapter we would like to give you more detailed information about programming and the structure of a typical ***ADbasic*** program.

4.1 Starting the development environment

- *Start **ADbasic** by selecting in the Windows menu **Start ⇨ Programs ⇨ ADwin ⇨ ADbasic 3.0**.*

The ***ADbasic*** graphics interface appears with the typical *Windows* menus, a toolbar and the editor window.

Fig. 4-1: The ***ADbasic*** graphics interface

- *Please check in the menu **Options ⇨ Compiler**, if the processor type and the 'Link address' are set correctly. (If you have not made any changes at your **ADwin** system, the setting for the 'Link address' is 0x150.)*

4.2 Structure of an ADbasic program

! Important: Contrary to other BASIC compilers you must *first* declare the variables in each ***ADbasic*** program. Exception: the global variables PAR_x and FPAR_x.

The program text following the declaration of the variables can be divided into three segments:

```
INIT:

EVENT:

FINISH:
```

ADwin systems with an ADSP can have a fourth segment in the ADbasic program. This fourth segment is always executed with low priority, regardless of the priority defined for this process.

Note: The segment EVENT: *must* be included in your program. The other two segments can optionally be generated or left out.

The segment LOWINIT:

Note: The segment LOWINIT: can only be applied with ***ADwin*** systems equipped with an ADSP.

After program start this segment is executed only once, the same as with segment INIT:. The segment LOWINIT: is always executed with low priority. In this it differs from segment INIT:. The segment LOWINIT: has always to be placed before the segment INIT:

The segment INIT:

All program lines located between the commands INIT: and EVENT:, will be carried out exactly once after program start. This segment is used to generate a defined initial status for your process, for instance initializing your variables or setting the digital outputs.

In order you do not want any initializations, you can leave out this segment. Your first program segment after declaring the variables starts then with the segment EVENT:.

The segment EVENT:

The actual measurement data acquisition program is located in the segment EVENT: The program lines placed between the words EVENT: and FINISH:, are executed every time an event signal occurs. If your program has no segment FINISH:, the segment EVENT: encompasses the instruction EVENT: up to the end of your program.

An event signal can be generated either from the timer of your ***ADwin*** system or by a positive edge on the event input.

The segment EVENT: will be executed as long as the process is stopped by the PC or a predefined number of loops has been reached.

The segment FINISH:

All commands placed after the instruction FINISH: will be executed exactly once, after the process has been stopped. The segment FINISH: is used to bring the system to a defined final status. If you do not need the segment FINISH: it can be left out.

! **Important:** The segment FINISH: is always carried out with low priority, even if your process has been started with high priority.

You have the possibility of data interchange with the PC in all three blocks INIT:, EVENT: and FINISH: This can happen in both directions.

In order to be able to display and evaluate the data, you need a measurement data evaluation program such as *TestPoint* or *MATLAB*. Of course you can develop your own measurement data evaluation program using *Visual Basic, Visual C, Delphi,* etc. For each of these possibilities driver software is available, which links ***ADbasic*** and ***ADwin*** to the specified program - if you have particular questions, please do not hesitate to contact us.

4.3 Memory management

Your ***ADbasic*** programs use the memory of the ***ADwin*** systems for program code and data.

The file size in the editor is not limited. Also the size of the compiled file is only limited by the memory capacity of the ***ADwin*** system. Because the ***ADbasic*** programs are compiled, they are also compact and only need a few kBytes of your RAM memory on the ***ADwin***-system.

- *Click on this symbol in your toolbar*  *, to know the free memory (optionally:* ***Options ⇨ Parameter****)*

Note: The free memory is displayed in the parameter window. Please note, that because of the memory structure, the parameter window for the ADSP looks different from those of the processors T225, T400, T450 and T805.

A limitation of the program size may be possible when using systems with T225 processors, because these systems only have a memory of 64 kB; half of which is needed for the ***ADwin*** driver (adwin2.btl). In this case, please watch the structure of the remaining memory very carefully.

The ***ADwin*** drivers for the T400 and T805 processors (adwin4.btl and adwin8.btl, respectively) need approx. 70 kB, so that generally less than 100 kB of the RAM is occupied. The rest is then available for your data, which is approx. 900 kB (when you have a memory size of 1 MB).

When using systems with a T450 processor, the driver (adwin5.btl) needs approx. 120 kB. The remaining memory is for your data and ***ADbasic*** programs.

Systems with an ADSP have a very fast internal memory of 256 kB. This memory is divided into two sections of 128 kB each by the processor, the memory for programs and the memory for data. The first section is for the ***ADwin*** driver (adwin9.btl) as well as for the programs developed with ***ADbasic***.

The second 128 kB are available for data. In this area the global parameters PAR_1 to PAR_80 or FPAR_1 to FPAR_80 are located, as well as all locally declared, discrete variables for which another area is not explicitly defined.

For storing measurement data or signal processes, the standard ADSP is additionally equipped with 4 MB of dynamic memory (DRAM), which can be extended to 64 MB max. Here ***ADbasic*** stores all data from array structures, provided that no other memory area has explicitly been indicated at declaration. When array structures are declared, it is possible to specify the type of memory where the structures are to be stored.

Because access to the processor memory is five times as fast as the access to the dynamic memory, it is recommended to store short array structures, which require a very fast access time, to the data memory of the ADSP.

The examples given in Figure 4-1 illustrate all possible versions of memory specification.:

Memory type	dynamic memory (DRAM)
Declaration	`DIM DATA_1[1000] AS LONG`
Memory type	internal data memory of the ADSP
Declaration	`DIM signal[10] AS LONG AT DM_LOCAL`
Memory type	dynamic memory (DRAM)
Declaration	`DIM DATA_1[10000] AS LONG AT DRAM_EXTERN`
Memory type	static memory (SRAM)
Declaration	`DIM DATA_1[1000] AS LONG AT SRAM_EXTERN`

Table 4-1: Memory types of the ADSP 21062

Note: If you make no comments on the type of memory, ***ADbasic*** will store the structure automatically to the dynamic memory.

4.4 ADbasic data types

4.4.1 Notation of numbers

ADbasic numeric values can be optionally indicated in one of four possible notations. In the following examples the variable *x* is set to the decimal value 90.

Examples:

1. Decimal notation: `x = 90`
2. Exponential notation: `x = 9E1`

   In exponential notation the number behind the E indicates the power of ten. The number in front of the E is multiplied with the power of ten.
3. Binary notation: `x = 1011010B`
4. Hexadecimal notation: `x = 5aH`

   If the HEX-value begins with a letter, for instance feH, a zero must precede the f, that is: 0feH.

Note: The decimal separator for floating point numbers is the point (.), therefore do not use the German, but the English notation for numbers. The setting is made via PC operating system: **Start ⇨ Settings ⇨ System Control ⇨ Country.**

4.4.2 Predefined global variables

For bidirectional data transfer between parallel ***ADbasic*** processes or between PC and ***ADbasic*** processes, there are 80 global variables available, as well as up to 200 data sets (arrays). When using the T450, T805 and the ADSP, 80 global floating point variables are additionally predefined.

You can use these scalar variables in your program where you want to, *without* having to define them. The scalar variables are called:

- `PAR_1, PAR_2,..., PAR_80` for integer 32-bit values (LONG)

- FPAR_1, FPAR_2, ...FPAR_80 for floating point values on the T450, T805 and ADSP

Examples:

```
PAR_5 = 700             'parameter 5 includes the
                        'value 700.
PAR_72 = ADC(1)         'The voltage at the analog
                        'input 1 is measured and
                        'included into parameter
                        '72.
```

! Important: Contrary to the other variables, the global scalar variables PAR_x and FPAR_x *must not* be declared, because they are already known to the ***ADbasic*** compiler.

In addition to the scalar variables you can use for data transfer with other processes on the ***ADwin*** system or the PC, data sets called DATA, which enable you to transfer huge quantities of data.

Note: Since size and data type are selectable, you have to define data sets (DATA-arrays) at the beginning of your program.

The DATA-arrays are called:

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

Other names are not allowed. But you need not use consecutive numbers, the declaration of for instance DATA_5 (without DATA_1 to DATA_4) is allowed, too. In your program the compiler differentiates the data sets according to their numbers.

Example:

```
DIM DATA_5[20000] AS SHORT    'declares data set 5
                              'with 20000 elements
                              'of type Short.
```

The maximum size of the data sets is only dependent on the available memory space. A data set with 1.9 million short-elements can be declared on an ***ADwin*** system with 4 MB memory.

Once the data set has been declared, you can access each separate element. The first element in the data set has the index 1.

Examples:

```
PAR_1 = DATA_5[200]       'The value of the 200th
                          'element from data set 5
                          'is assigned to the global
                          'variable 1.
DATA_5[345] = 4000        'With this command the
                          '345th element in data set
                          '5 receives the value
                          '4,000.
```

You can also assign the number of the element via a variable.

```
number1 = 345
DATA_5[number1] = 4000    'As in the previous
                          'example, the 345th
                          'element of data set 5
                          'receives the value 4,000.
```

! Important: The data set number *must not* be assigned by a variable. The following command leads to an error message from the ***ADbasic*** compiler!

Example:

```
index = 2
```

```
DATA_index[300] = 20
```
INCORRECT !!

A constant number must be used instead of `index` .

4.4.3 User-definable variables

All the variables which you need for your process must be declared at the beginning of the ***ADbasic*** program. The two processors T225 and T400 only operate with integer values. The T805, the T450 and the ADSP also allow floating-point variables.

The T450 processor is not equipped with a floating-point unit (FPU), contrary to the T805 and the ADSP processors. Therefore all floating-point operations for the T450 are emulated by ***ADbasic***. As a consequence, when using the T450 processor, the additional effort for arithmetical operations should be minimized by economically applying floating-point variables.

Variable type	Processor				
	T225	**T400**	**T805**	**T450**	**ADSP**
SHORT	16 Bit	16 Bit	16 Bit	16 Bit	16 Bit
INTEGER	16 Bit	32 Bit	32 Bit	32 Bit	32 Bit
LONG	32 Bit	32 Bit	32 Bit	32 Bit	32 Bit
FLOAT	-	-	32 Bit	32 Bit	32 Bit

Table 4-2: Processors and available types of variables

Examples:

```
DIM value AS INTEGER     'Defines the variable
                         'value with the data type
                         'INTEGER

DIM value1, value2 AS INTEGER 'Defines the
                              'variables value1 and
                              'value2 with the data
                              'type INTEGER.
```

When you combine two variables by operations, ***ADbasic*** takes the data types into account and converts the variables in a way, that they can be combined with one another. For more information, see chapter " Type conversion".

You can also declare variables not just as scalar values, but also as arrays that means you can generate fields of variables and process them. The number of fields in the array is entered in square brackets after the name.

Important: All variables, used in your ***ADbasic*** program, have to be declared *before* starting the first program section.

Example:

```
DIM value[100] AS LONG   'Defines an array of the
                         'length 100 with the name
                         'value and the data type
                         'LONG.
```

! Important: All the variables which you use in your ***ADbasic*** program must be declared *before* the beginning of the first program segment.

4.4.4 The FIFO data structure

For applications in which large quantities of data need to be continuously transferred, there is a data structure which is organized as a FIFO (**F**irst **I**n, **F**irst **O**ut). All values written to the FIFO are put into a queuing condition. They are read out again by the PC or another ***ADbasic*** process in the same sequence in which they were written. You must specify the FIFO size in the declaration. Besides the clear-function there is also the possibility of querying the occupied and the free memory.

! Important: Since ***ADbasic*** treats the FIFO internally as a data set, the same data set number *must not be used simultaneously* as the number of FIFO and of a normal DATA variable.

! Important: When using ***ADwin*** systems with ADSP21062 the data structure FIFO *must not* be declared as a SHORT data type.

The declaration is similar as for the DATA structure.

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

This command declares an array with the data set number 1 and a length of 1000 integer values as FIFO ring buffer.

You can access the FIFO by indicating its number, the FIFO automatically writes the transferred value to the correct position or reads it out in the correct sequence.

Example:

```
DATA_1 = 95              'Writes the value 95 to
                         'the FIFO number 1

PAR_7 = DATA_1           'Reads a value from the
                         'FIFO and saves it in the
                         'global variable 'PAR_7
```

To ensure that there is still space in the FIFO, you should use the function FIFO_EMPTY. In the same manner, the function FIFO_FULL checks if unread values are present before reading out.

Example:

```
free = FIFO_EMPTY(1)
IF (free > 0) THEN
  DATA_1 = value1
ENDIF

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

The FIFO `DataSetNo` can be cleared with `FIFO CLEAR_ (DataSetNo).`

Note: Since the FIFOs are not automatically cleared on start-up, they should be cleared with this command in the `INIT:` program segment.

! Important: If you write data faster to the FIFO than you read them out, the FIFO will be full sometime and data will get lost.

4.4.5 Status variables

The following status variables are available to obtain information about the status of the ***ADwin*** system:

`PROZESS_RUNNING`

Indicates the process status. The value is 1 when the process is running.

`GLOBALSCHLEIFE`[5]

Number of loops which the process should execute. This status variable can be defined under **Options ⇨ Process** in the field 'Number of Loops'. This variable is decremented by an internal as well as external event.

`ANZAHLSCHLEIFE`[3]

Contains the number of loops to be carried out. This variable only exists when during compilation the number of loops is higher than zero.

[5] `GLOBALSCHLEIFE` and `ANZAHLSCHLEIFE` are not available when using an ADSP.

GLOBALDELAY

Time interval in microseconds between two events. This status variable can be defined under **Options ⇨ Parameter** in the field 'Delay in µs'.

NWTIME

Status of the internal timer at the moment of starting the last timer event.

! Important: You should only read these variables *within* an ***ADbasic*** program, and never overwrite them with new values, because the ***ADwin*** board may enter an instable mode.

4.4.6 Type conversion

If you link two variables to operations, ***ADbasic*** pays attention to the data types and converses the variables so that they can be linked with each other.

If in a line a value or variable is a floating point value/variable, all values/variables will be converted into floating point numbers before evaluation. If necessary, the result will be converted again to an integer variable at the end of the evaluation, that means the decimal places can be cut off.

! Note: Since LONG and FLOAT data types have different value ranges, a loss of precision occurs during FLOAT conversion of large LONG numbers. The deviation may be up to 64. (The LONG number 2000000064 is converted to the FLOAT data tpye 2000000000).

Example:

```
PAR_1 = 2000000001
PAR_2 = 2000000002

FPAR_3 = (PAR_2 - PAR_1) + 0.5
             ' FPAR_3 is 0.5 , because PAR_1 and PAR_2
             '  are converted to FLOAT before
             '  subtraction

PAR_9  =  PAR_2 - PAR_1
FPAR_4 =     PAR_9     + 0.5
             ' FPAR_4 is 1.5, because subtraction is
             ' done with LONG numbers
```

! Note: Even parentheses do not prevent an automatical type conversion into FLOAT. If calculations are to be done in LONG, a line has to be programmed for that (see PAR_9 in the example above).

An exception to the rule mentioned above, are the instructions `IF ... THEN` and `DO ... UNTIL`. Here only parts of the line are converted, not the whole line. The logical operators `AND` and `OR` as well as the word `THEN` separate the individual segments from each other.

Example:

```
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
                    ' The IF statement is true,
                    '  because PAR_1 is not converted
                    '  into FLOAT

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

                    ' Result:  PAR_3 = 1
                    ' that means subtraction is made
                    ' with LONG numbers, not with
                    ' FLOAT numbers
```

4.5 Using a number of ADbasic processes

Up to ten ***ADbasic*** processes can run simultaneously on each processor of an ***ADwin*** system. An exception is the processor T225, only two ***ADbasic*** processes can run simulatenously on this processor. Moreover, a range of standard I/O and control processes are available on each processor which also run simultaneously. In addition, the communication process runs with low priority in the background. This process controls the data transfer between the PC and the ***ADwin*** system. The various processes are shown in Fig. 4-2. Note: There are no PID-controller processes available for the ADSP.

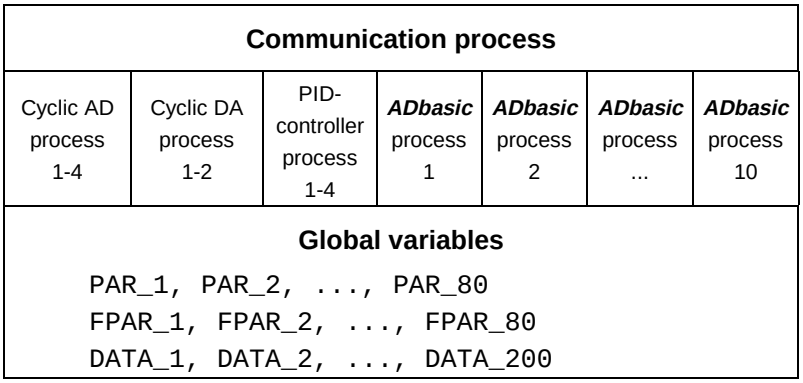


Fig. 4-2: ***ADwin*** processes

You have to allocate a process number between 1 and 10 using the menu **Options ⇨ Process Options** to each program which is to run on the system.

! Important: If you load two processes with the same process number onto the system, the one loaded first is overwritten!

4.5.1 Data interchange

Similar to the data interchange between different ***ADwin*** systems and the PC, the data interchange between different processes on the ***ADwin*** system is made possible by use of global variables(`PAR_1 ... PAR_80`) or the global data structures (`DATA`). All processes have access to these variables. Their declaration is identical for all processes.

The global variables can also be used in order to control a simultaneously running process out of another process.

Example:

The ***ADbasic*** process 1 records measurement data continously and stores them to the data set 1 (`DATA_1`), which is defined as a FIFO. A second simultaneously running ***ADbasic*** process queries the number of elements in the FIFO at regular time intervals with the function `FIFO_FULL(1)`. When a specified number of elements are found in the FIFO, the second ***ADbasic*** process starts evaluating the measurement data (e.g.. FFT, calculation of mean values, acceleration, etc.). In the meantime the ***ADbasic*** process 1 can carry on with the continous recording of measurement data. Due to the FIFO structure, the second ***ADbasic*** process is able to read out the measurement data in the same order as they have been stored by process 1. Process 2 only needs to access the same global data set (here DATA_1) which was also used by process 1 for storing the measurement data.

! Important: If a data set should be accessed in two processes, the data set has to be declared in *both* ***ADbasic*** processes in the *same* way.

4.5.2 Workload

When a number of processes run simultaneously on the ***ADwin*** system, they have to share the processing time. The whole workload of the ***ADwin*** system then consists of the sum of the workload, caused by each single process. You can observe the processor workload on the 'busy' display in the parameter window (menu **Options ⇨ Parameter**).

4.5.3 A number of high priority processes

A number of high priority processes can run simultaneously on the ***ADwin*** system. Since high priority processes cannot be interrupted, the internal time characteristic of the individual processes does not change.

However, differences occur for the time spans between the event calling the process and the actual start of processing. If a high priority process is called while a second high priority process is being executed, then the process just called must wait until the execution of the process already called is completed. That results in delays for the starting process.

Note: Please note that the maximum delay for carrying out a process is the sum of all execution times of all other processes running simultaneously with high priority on the ***ADwin*** system. If several high priority processes run simultaneously on an ***ADwin*** system, make sure that the execution time of the individual processes is as short as possible.

4.6 Subprograms and functions

In order to be able to structure your programs better, you have the possibility of using subprograms and functions in ***ADbasic.*** The syntax is very simple. Just use the terms SUB ... ENDSUB or FUNCTION ... ENDFUNCTION to enclose the particular procedures like brackets. On calling, variable values can also be passed, but not the variables themselves. In case you want to change the variable definition globally in a function or subprogram, we recommend to use the global variables PAR_1 to PAR_80.

You can also create a „library“ of subprograms or functions and save them in a separate file. This file can then be very easily included into your current program with an INCLUDE command. The INCLUDE command must however be located right at the beginning of your program. Functions and subprograms can also be located before the INIT block or after the FINISH segment at the end of your program.

Note: A documentation and an example of the commands for calling subprograms or functions are in the chapter Command reference.

4.7 Evaluation and visualizing of measurement data

You need a measurement data evaluation program such as *TestPoint* or *MATLAB* in order to be able to display and evaluate the data. Of course, you can also design an evaluation program of your own using *Visual Basic, Visual C, Delphi, Excel* etc. Driver software linking ***ADbasic*** and ***ADwin*** with the desired program, is available for each of these programs. The interaction and the communication between the **ADwin** system and other programs is explained in detail in the documentation for the drivers which can be obtained for these programs. Here you will also find out how you can load programs, which you have generated and compiled with ***ADbasic***, onto the ***ADwin*** system and start them. Since the measurement program on the ***ADwin*** system is always the same, you can also access the ***ADwin*** system from a number of different evaluation programs simultaneously.

Please contact us if you have special requirements.

5 Optimizing the timing characteristics

5.1 Priority and timing characteristics

The processors of the ***ADwin*** systems have their own process management. It differentiates between two priority levels when executing the separate processes. High or low priority levels can be assigned to each process.

This feature of the processor can also be used under ***ADbasic***. You can set the priority of each process in the menu **Options ⇨ Process Options**.

High priority

When using the processors T225, T400, T450, and T805 you can be sure that from the moment of calling a process with high priority, up to starting its processing, only 2.5 µs max. will pass. When using an ADSP21062 max. 300 ns will pass. The commands of the high priority process are executed completely before the processor of the ***ADwin*** system is available again for other processes. This ensures constant and exactly predictable timing characteristics.. The high priority process therefore enables reliable reaction and response times in the microsecond range.

The interval (delay) between two events produced by the internal timer on the ***ADwin*** system can be specified for ***ADwin*** systems with T225, T400, T450 or T805 processors with a resolution of 1 microsecond for a high priority process. ***ADwin*** systems with an ADSP21062 processor have a resolution of 25 ns.

Example:

If 'delay' is set to the value 100 when using an ***ADwin*** system with a T805 processor, then a time span of 100 µs occurs between two consecutive, timer controlled calls of the same process.

Since a high priority process cannot be interrupted, it must be ensured that the processing time of the process itself is clearly shorter than the delay time (in the example above 100 µs). Otherwise the processor of the ***ADwin*** system does not have any more time

available to serve other processes running concurrently, for instance the communications process.

! Important: The execution time of high priority processes should be kept as short as possible. Time-consuming loops or computations, the result of which is not immediately required for further processing, should always run with low priority.

The segment FINISH of each ***ADbasic*** process is always executed with low priority. If a process with high priority has been started, the priority on getting to the FINISH segment is automatically reduced.

Low priority

A process running with low priority can be interrupted at any time by a high priority process. Therefore, a number of processes with low priority can run simultaneously on the ***ADwin*** system without the timing characteristics of a high priority process being affected.

However, the ***ADwin*** system can only execute low priority processes if its computing power is not completely required by a high priority process.

! Important: Time-critical measurement processes cannot be disturbed by other processes running on the ***ADwin*** system at the same time, if the time-critical measurement process is running with high priority and all other processes are running with low priority.

The delay between two events produced by the internal timer on the ***ADwin*** system can be specified for ***ADwin*** systems with T225, T400, T450 or T805 processors with a resolution of 64 µs for a low priority process. When using ***ADwin*** systems with an ADSP21062 processor the resolution is 100 µs.

Example:

If the delay value is set to the value 5 in an ***ADwin*** system with ADSP21062 processor, then a time span of 500 µs occurs between two consecutive, timer-controlled calls of the same process.

! Important: When existing ***ADbasic*** processes are compiled for the ADSP21062, the different timer resolution has to be considered. In order to get the same delay for a high priority, timer-controlled process as for instance for the T805, the delay setting of the T805 has to be multiplied by the factor 40.

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

Table 5-1: Examples for setting the delays when compiling existing *ADbasic* processes for the ADSP21062.

5.2 Checking execution times with timer functions

The processor of the ***ADwin*** system has two internal timers which are defined as counters.

In ***ADwin*** systems with T225, T400, T450 or T805 processors the first timer is incremented by the value 1 every microsecond and is read out in a high priority process. The second timer is incremented by the value 1 only every 64 µs and is read out in a low priority process.

In ***ADwin*** systems with ADSP21062 the first timer is incremented by 1 every 25 ns and read out in high priority processes. The second timer is incremented by 1 only every 100 µs and read out in low priority processes.

After powering up the ***ADwin*** system both counters are set to 0. Afterwards they are continously incremented according to the clock rate mentioned above.

With the ***ADbasic*** function `READ_TIMER()` the present count rate can be determined.

Note: Please note that with the T225 the timer registers are only 16 bits wide. You will finder further information in the chapter Command reference under the command `READ_TIMER`.

Note: All ***ADbasic*** functions, which have access to the processor timer (`READ_TIMER()`, `NWTIME`, ...), present the count rate in 25 ns when using an ADSP.

With `READ_TIMER()`, for instance, the time difference between two externally triggered events can be easily determined. In relation with the parameter `NWTIME` the latency can also be determined, i. e. the time between calling and start of the process. The parameter `NWTIME` contains the count rate at which the current process is to be called. If the timer is read with the first command of the process and compared with the value of the parameter `NWTIME`, then the latency is obtained. The following example illustrates the procedure.

Example:

```
DIM time, latency, max_latency AS INTEGER

EVENT:
time = READ_TIMER()
latency = NWTIME - time
IF (latency > max_latency) THEN
  max_latency = latency
ENDIF
```

5.3 A faster measurement function

With the ADC command *one* A/D conversion for *one* channel is executed with a certain gain. The command is kept very simple so that it is easy to use. But the fact that there are two ADCs on the ***ADwin*** board with *which two different channels can be converted simultaneously*, is not considered.

Note: On the ***ADwin-light*** board there is only *one* ADC.

In order to use both ADCs on the ***ADwin*** board simultaneously, replace the ADC command by the commands, illustrated below. With these commands you can optimize the process speed.

Example:

```
SET_MUX(0)                  'Set the multiplexers to
                            'the channel 1 of the ADCs
     Wait for the settling of the multiplexers
START_CONV(3)               'Start conversion of both
                            'ADCs
WAIT_EOC(1)                 'Wait for end of
                            'conversion
ad1 = READADC(1)            'Read out ADC1
ad2 = READADC(2)            'Read out ADC2
```

Note: The areas highlighted in gray are waiting periods which are necessary for waiting that the multiplexer will be settled and the ADCs will be converted. For instance there is the possiblity of setting the multiplexer for the next measurement directly after the command START_CONV(3). The settling time of the ***ADwin-GOLD*** system is 6.5 µs (16bit), the conversion time is 7 µs, i.e. 13.5 µs altogether which you can use elsewhere. For more information about settling times, see the corresponding hardware manuals of the ***ADwin*** systems or see chapter 7 "Command Reference", the commands ADC and ADC12.

! **Important:** It is essential to keep the waiting periods and the conversion of the ADCs, because the AD conversion does not function otherwise and supplies *incorrect results*.

6 Menus and dialog windows

When you start ***ADbasic*** from your *MS-Windows* graphics interface, the development interface of ***ADbasic*** which is illustrated in Fig. 6-1 is displayed. The ***ADbasic*** interface is built just like all other *MS-Windows* graphic interfaces, so that it will be easy for you to use it.

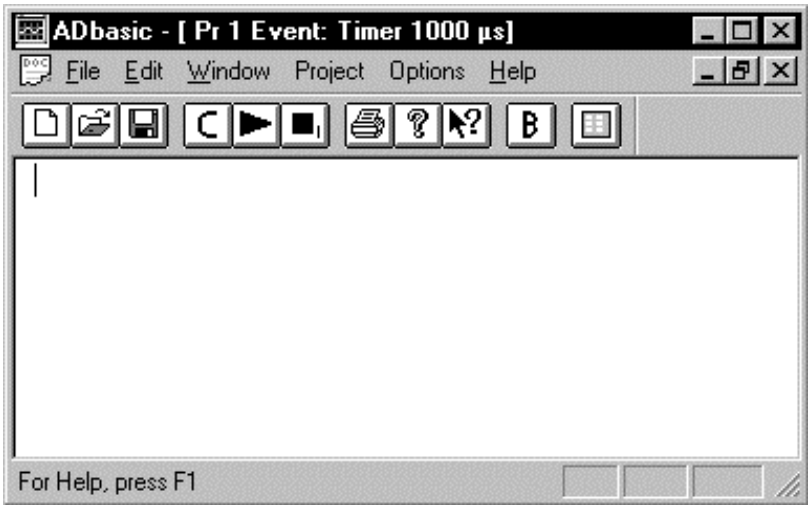


Fig. 6-1: The *ADbasic* development interface

The ***ADbasic*** graphics interface consists of the menu bar, the tool bar, and the editor window.

From the menu bar select the desired menu by either moving to the relevant field with the mouse cursor and pressing the left mouse key or by entering the key combination [ALT] + [FIRST LETTER] of the relevant menu.

With the tool bar you have the possibility of quickly accessing frequently used commands. Each button corresponds to a command in the menu bar.

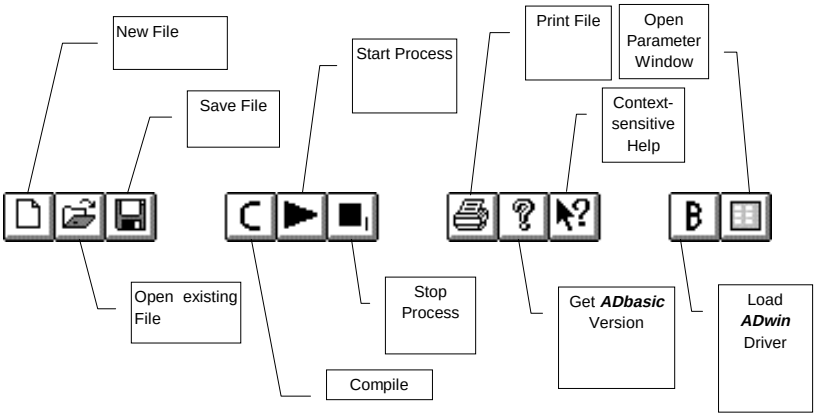


Fig. 6-2: The tool bar

6.1 The menu File

As is usual with *Windows* you can load and save files and create new files (i.e. editor windows) with the menu **File**. You can create any number of editor windows, but you can only load a maximum of ten programs/processes onto your ***ADwin*** system simultaneously.

This menu also contains the print functions (print, print preview, printer setup).

This menu also displays a list of the files which have been used earlier. The size of the list is limited to four files.

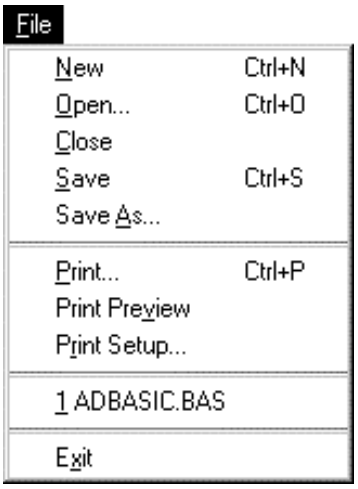


Fig. 6-3: The menu File

6.2 The menu Edit

The menu **Edit** also conforms to the *Windows* conventions. It includes Find and Replace functions.

Note: The Undo function is only reasonable for the Copy, Cut and Paste functions.

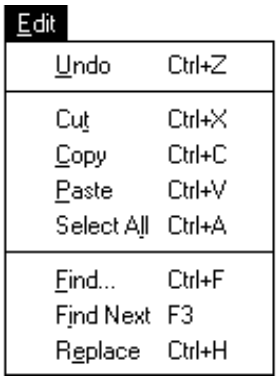


Fig. 6-4: The menu Edit

6.3 The menu Window

You can switch between different editor windows and arrange them on the screen with the menu **Window.**

Moreover you have the possibility to switch the tool bar and the status bar on and off.

In addition you will also find information here about the active programs: Name, type of event, and delay value.

Example:

In Fig. 6-5 two programs are active. The name of the program is ADBASIC.BAS; the process number of the program is 1, the event is generated by an internal timer, the time interval between two events is 1000 µs.

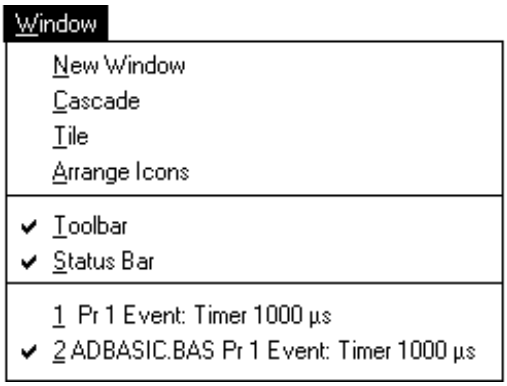


Fig. 6-5: The menu Window

6.4 The menu Project

You can compile, start or stop programs with the menu **Project.**

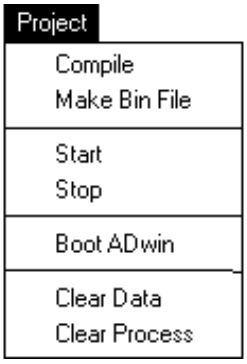


Fig. 6-6: The menu Project

Compile translates your current program and transfers it to the ***ADwin system*** for execution. If you have selected 'Yes' in the menu **Options ⇨ Compiler** 'Autostart', your process will be started immediately after the transfer. Otherwise you can start the execution of your program with the menu 'Start' .

Use the menu 'Stop', in order to stop the current process (active window). .

Make Bin File saves the current process in compiled form (as binary files). The file name extension is here automatically specified by the program. The letter "T" is always part of the file name extension. When using ***ADwin*** systems with the processor ADSP21062 a "9" follows after the "T". When using ***ADwin*** systems with the processor T805, an "8" follows, a "5" for the T450, a "4" for the T400 and a "2" for the T225. The second number illustrates the process number, which you have set in the menu **Options ⇨ Process Options.**

Example:

The file name extension *.T81 means that the processor T805 is being used and the process number 1 is involved.

If the corresponding ***ADwin*** driver has been loaded to your ***ADwin*** system, you will be able to load and start binary files generated by ***ADbasic*** to your ***ADwin*** system, even without using the ***ADbasic*** development environment. For more information please read our driver documentation which will be supplied to your program for visualizing measurement data.

Boot ADwin loads the driver program again onto the ***ADwin*** system. This deletes all running processes and all global variables are set to the value 0. After loading the ***ADwin*** driver the value for the global delay has the default setting of 25000 ns when using the ADSP and 1000 µs when using all other processor types.

! Important: If you load two processes with the same process number (menu **Options ⇨ Process Options**) onto your ***ADwin*** system, the process you loaded first will be overwritten without any security check.

6.5 The menu Options

In the menu **Options** you have the possibility of setting parameters to influence the type and execution of your program.

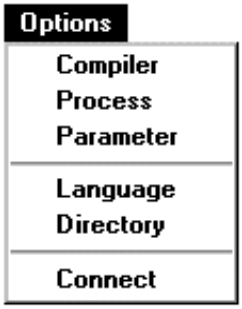


Fig. 6-7: The menu Options

The settings in the Compiler, Clear Data, Language, Directory, and Connect windows have an influence on *all* program windows.

The settings in the process-window refer *only* to the program in the editor-window; which has been active before calling the individual menu item. If you have opened several editor-windows, please check before calling, whether the window of the program for which you want to make the settings, is active in the foreground.

6.5.1 The compiler options

Options ⇨ Compiler

The dialog window opened with this menu item is used to give the compiler information about the specified processor and other parameters of your ***ADwin*** system. The values set here apply to all ***ADbasic*** processes.

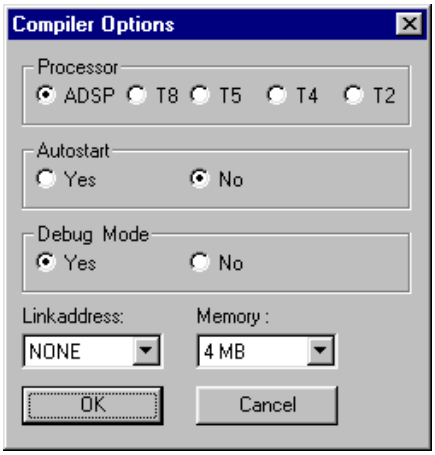


Fig. 6-8: The dialog window Compiler Options

'Processor':

Set the processor which you have on your ***ADwin*** system. If you are not sure which processor you should use, please refer to the manual of your ***ADwin*** board, your ***ADwin-Box*** or your ***ADwin-Pro*** system.

Note: Frequently, only abbreviations of the processors used in ***ADwin*** systems are mentioned. Table 6-1 compares the abbreviations and the full names.

Processor name	ADSP	T8	T5	T4	T2
Processor	ADSP 21062	T805	T450	T400	T225

Table 6-1: names of the processors

Since you can also compile programs for a processor other than the one that is installed, the value that was last set is kept. Only on compiling will the current configuration be checked. If it does not match with the settings, you can compile, but you cannot load the program onto the ***ADwin*** system.

'Autostart':

Here you can set if your program is to start immediately after compiling or only after the START key is pressed.

‘Debug mode’:

Here you can specify that additional security checks are integrated in your program, in order to detect the following run time errors and to display them in the "status" field of the parameter window.

- division by 0
- square root of values less than 0 (negative values)
- access to non-defined data elements
- indicating repetition rates that are too short
- access to array elements which are not defined

Note: Activating the debug mode requires computing time and extends the programs execution time by about 20 %. You should therefore only use this option during program development.

‘Linkaddress’:

Select here the link address (base address for the linkadapter), which is set on the ***ADwin*** system with the DIP switches. The default setting is 336 (150H). If you want to do the development without a processor, then select ‘NONE’

‘Memory’[6]:

Enter the memory size of your ***ADwin*** system.

6 Memory size will be determined automatically when using an ADSP.

6.5.2 The process options

Options ⇨ Process

The dialog window is used for defining the parameters specific to the process. The values set here apply to the active program window and have to be set before compiling and loading the process.

Depending on the processor, either the dialog window shown in Fig. 6-9 or that one shown in Fig. 6-10 will be opened.

6.5.2.1 Dialog window Process Options for the processors T225, T400, T450 and T805

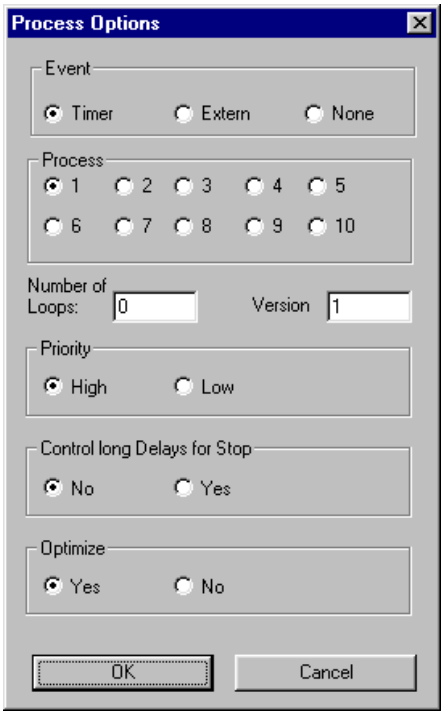


Fig. 6-9: The dialog window Process Options

‘Event’:

Here you can define how the events which start your program are generated. You can either use the internal 'timer' of the processor of your ***ADwin*** system for this or you use a signal (positive edge) at the event input at the rear of the ***ADwin*** board and select 'Extern' in the dialog window Process Options.

Note: How you can use an external event in an ***ADwin-Pro*** system, is explained more detailed in the documentation with the title ***"ADwin-Pro*** System Specifications – Programming in ***ADbasic***" (EVENTENABLE command).

With the setting ‘None’ your program is immediately started independent of any event and is executed again after the last command has been executed. Particularly where a process has high priority, you have to make sure with the setting 'None', that the process also has computing time available for other tasks (communication with the PC/with other processors connected to each other by a linkadapter).

Note: For more information, see the command LINKIN.

‘Process’:

This is the number of your process. If you would like to run a number of processes simultaneously on the ***ADwin*** system, you must assign a number to each process.

Note: With T805 processors Process1 runs in the internal memory and therefore faster.

‘Number of Loops’:

If required, you can set here the number of event-generated program calls for your program. Once the set number is reached, the program stops. This maximum number of runs can be defined again for each program start, without having to recompile the program.

If you enter the value 0, the program is repeated as long as you

- stop the process with the END-command in the ***ADbasic*** program,
- stop the process with a STOP_PROCESS command from the PC or from another ***ADbasic*** process.

- stop the process explicitly with the STOP key from ***ADbasic***.

'Version':

Here you can enter an integer value as version number. This version number will be displayed in a text editor window (only), when using a compiled program.

'Priority':

If you want to run a number of processes simultaneously on an ***ADwin*** system, you can set the priorities here for the execution of the active process.

You will find further information about this topic in Chapter 5 (Optimizing the timing characteristics).

'Control long Delays for Stop':

When using timer-controlled processes which are rarely called, i.e. processes with a delay value of more than 5 milliseconds, you should use this option in order to be faster able to stop the processes.

'Optimize':

If the optimizer is used it can shorten the program execution time for max. 20 %.

6.5.2.2 Dialog window Process Options for the processor ADSP21062

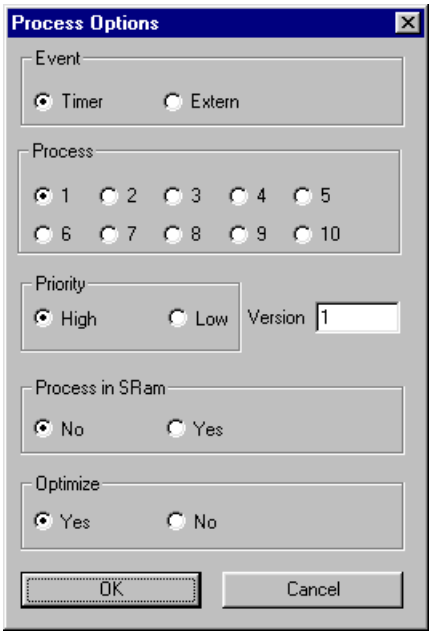


Fig. 6-10: The dialog window Process Options for the ADSP21062

'Event':

Here you can define how the events which start your program are generated. You can either use the internal 'timer' of the processor of your ***ADwin*** system for this or you use a signal (positive edge) at the event input at the rear of the ***ADwin*** board and select 'Extern' in the dialog window Process Options.

Note: timer-controlled processes generated for the ADSP must always run with high priority.

Note: How you can use an external event in an ***ADwin-Pro*** system, is explained more detailed in the documentation with the title ***"ADwin-Pro*** System Specifications – Programming in ***ADbasic***" (EVENTENABLE command).

‘Process’:

This is the number of your process. If you would like to run a number of processes simultaneously on the ***ADwin*** system, you must assign a number to each process.

‘Version’:

Here you can enter an integer value as version number. This version number will be displayed in a text editor window (only), when using a compiled program.

‘Priority’:

If you want to run a number of processes simultaneously on an ***ADwin*** system, you can set the priorities here for the execution of the active process.

You will find further information about this topic in Chapter 5 (Optimizing the timing characteristics).

‘Process in SRam’:

Use this option, if your ***ADwin*** system is equipped with an ADSP21062 processor with SRAM, and if you want that the active process will be loaded to the SRAM so that its execution time decreases.

‘Optimize’:

If the optimizer is used it can shorten the program execution time for max. 2 %.

6.5.3 The dialog window Parameter

Options ⇨ Parameter

You can only open the dialog window Parameter, if you have loaded the ***ADwin*** drivers to your PC and if your ***ADwin*** system has been booted. It gives you a brief overview of the values of a few of the global variables as well as of the free memory on your ***ADwin*** system. Its main use is for debugging, verifying and controlling your program.

Depending on the processor in use, the dialog window shown in Fig. 6-11 or in Fig. 6-12 will be opened.

6.5.3.1 Dialog window Parameter for the processors T225, T400, T450 and T805

The first line displays the workload of the ***ADwin*** system processor in percent (busy) and the free memory (free memory) of the ***ADwin*** system in bytes.

If you have compiled the program in the debug mode, it is possible that run time errors are displayed in the field 'Status'.

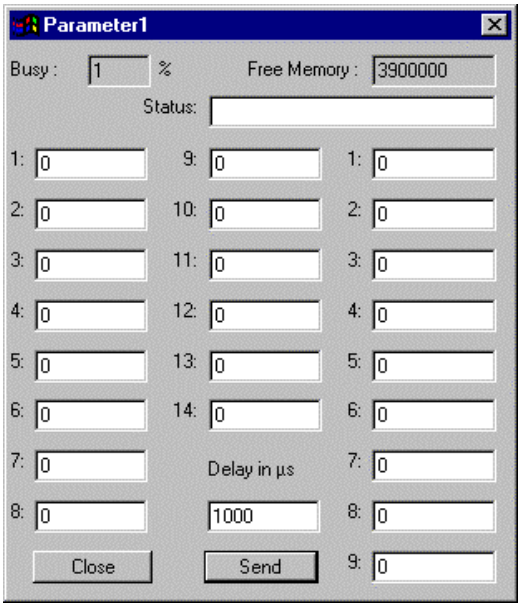


Fig. 6-11: The dialog window Parameter for the processors T225, T400, T450 and T805

The window shows 14 global integer variables (PAR_1 to PAR_14) and – if you have the T805 processor on your ***ADwin*** system - 9 global variables (FPAR_1 to FPAR_9)

All variables can be changed, even if the program is already running. All changed variables are transferred to the ***ADwin*** system by clicking the 'SEND' button.

The time interval (delay) between two events produced by the internal timers is also shown for the process which is active when opening the window.

For the high priority process the unit for the delay value is 1 µs and 64 µs for a low priority process (as is displayed in the window). When calculating the time interval for low priority processes, you must therefore multiply the indicated number with 64 µs: A display of 7 corresponds to a total time of 7 * 64 µs = 448 µs. For a high priority process there will be a display in microseconds.

Note: The display of the correct delay value, dependent on the specified priority, is shown for the active window in the title bar.

The delay value can be changed via this window or from most application programs (*TestPoint*, *MATLAB,* etc.).

6.5.3.2 Dialog window Parameter for the ADSP21062 processor

The first line indicates the free memory of the ADSP in bytes.

array name	memory type
PM	internal program memory
DM	internal data memory
DX	external data memory (DRAM)

The second line indicates the workload of the ADSP in percent (Busy). If you have compiled the program in the debug mode, it is possible that run time errors will be displayed in the field 'Status'.

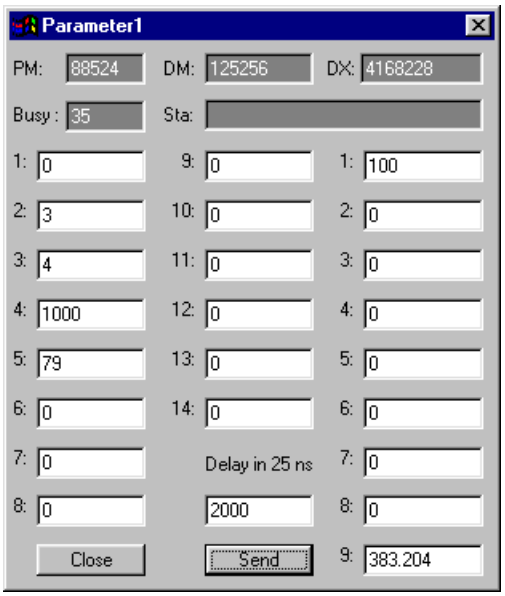


Fig. 6-12: The dialog window Parameter for the processor ADSP21062

The dialog windows displays 14 global integer variables (PAR_1 to PAR_14) and 9 global float variables (FPAR_1 to FPAR_9).

All variables can be changed, even if the program is already running. All changed variables are transferred to the ***ADwin*** system by clicking the 'SEND' button.

The time interval (delay) between two events produced by the internal timers is also shown for the process which is active when opening the window.

For the high priority process the unit for the delay value is 25 ns and 100 µs for a low priority process (as is displayed in the window). When calculating the time interval for low priority processes, you must therefore multiply the indicated number with 25 ns. A display of 7 corresponds to a total time of 7 *25 ns = 175 µs.

Note: The display of the correct delay value, dependent on the specified priority, is shown for the active window in the title bar.

The delay value can be changed via this window or from most application programs (*TestPoint*, *MATLAB,* etc.).

6.5.4 The language window

Options ⇨ Language

Here you can choose the language for the error messages of the compiler. You can choose between English and German.

6.5.5 The specification of the directory

Options ⇨ Directory

This dialog box specifies the directory which provides the driver file (*.BTL) for the compiler so that the ***ADwin*** system can be booted, as well as the directory with the ***ADbasic***-#INCLUDE-files. The compiler needs these #INCLUDE-files in case a directory is not explicitly indicated with the ***ADbasic*** #INCLUDE instruction.

6.5.6 The dialog box Connect

Options ⇨ Connect

With ***ADbasic*** you can get access to an ***ADwin*** system via a network (e.g. LAN, ISDN, Internet, ...) which is connected with any network server. First the program ***ADserver*** has to be started on the computer where the ***ADwin*** system is located. Afterwards you establish the communication between ***ADbasic*** and the network server, which is connected to the ***ADwin*** system, by pressing the button 'Connect'. As soon as the communication is established all ***ADbasic*** actions are carried out via network to the computer which is connected with the ***ADwin*** system.

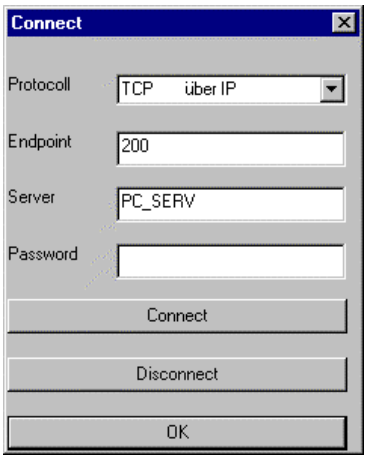


Fig. 6-13: The dialog box Connect

'Protocol':

Here you can set the network protocol. The protocol has to be installed properly on your server. The setting has to be identical to the setting in the program ***ADserver.***

‘Endpoint’:
Endpoint for the network communication. This setting has to be identical to the setting in the program ***ADserver***.

‘Server’:
Name or address of the server to whom the communication should be made.

‘Password’:
In case a password is entered for the program ***ADserver***, the same password is required here, too. The password is case sensitive.

6.5.7 The Help menue

With this menue you call the Windows help function for ***ADbasic***.

Fig. 6-14: The menu Help

7 Command reference

The following section contains an alphabetical listing of all ***ADbasic*** commands. The current command being described is indicated by bold printing. For clarification, a short application example is given for each command.

Note: Not all commands are always available. Please note what will be indicated correspondingly.

In addition, for commands requiring a large amount of time, the execution time is shown in dependence of the type of processor.

You will find a fold-out summary of all commands with page references on the last page.

Addition +

Syntax:

Value3 = *Value1* **+** *Value2*

Application example:

```
value3 = 9 + 4           'Result: value3 = 13
```

Subtraction -

Syntax:

Value3 = *Value1* **-** *Value2*

Application example:

```
value3 = 9 - 4           'Result: value3 = 5
```

Multiplication *

Syntax:

```
Value3 = Value1 * Value2
```

Application example:

```
value3 = 9 * 4           'Result: value3 = 36
```

Division /

Syntax:

```
Value3 = Value1 / Value2
```

Application example:

```
value3 = 36 / 9          'Result: value3 = 4
```

Power ^

Syntax:

```
Value3 = Value1 ^ Value2
```

Time relationship:

FLOAT: 3,65µs

INT.: 450µs
LONG: 450µs

INT.: 110µs
LONG: 110µs
FLOAT: 110µs

INT.: 430µs
LONG: 430µs
FLOAT: 430µs

The time required increases for increasing powers.

Application example:

```
Value3 = 4 ^ 3            'Result: Value3 = 64
```

Compare <=>

Syntax:

Value1 **>** *Value2*

Description:

The compare operators are used to compare two values or expressions. The result is either true (1) or false (0) and can be evaluated, for example, by the IF or UNTIL command.

operator	meaning
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
=	equal
<>	not equal

Application example:

```
DIM value1, value2 AS INTEGER

EVENT:
value1 = -5
if (value1 < 0) then value1 = 0
REM Result: value1 = 0
```

ABS

Syntax:

```
Value2 = ABS(value1)
```

Description:

The function ABS supplies the absolute value of an integer/long variable.

Time relationship:

LONG: 0,1µs

INT.: 3,4µs
LONG: 3,4

LONG: 2,9µs

LONG: 25µs

Application example:

```
DIM value1, value2 AS INTEGER

EVENT:
value1 = -5
value2 = ABS(value1)     'Result: value2 = 5
```

ABSF

Syntax:

Value2 = ABSF(*Value1*)

Description:

The function ABSF supplies the absolute value of a float variable.

Time relationship:

FLOAT: 0,1µs FLOAT: 2,0µs FLOAT: 6,5µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = -5.3
value2 = ABSF(value1)    'Result: value2 = 5.3
```

ACTIVATE_PC

Syntax:

ACTIVATE_PC

Description:

The command ACTIVATE_PC starts the action list of the real-time object under *TestPoint*.

To achieve this, a global variable is set to 1. The PC will be able to know – when getting the variables – that the current PC function should be carried out.

Note: This command has been specially developed for *TestPoint* so that *TestPoint* actions can be initiated from ***ADbasic.***

Application example:

```
DIM value AS INTEGER     'Declaration

EVENT:
value = ADC(1)           'Acquire measurement
IF (value > 1000) THEN   'Comparison
  PAR_1 = value          'Save measurement
                         'in Parameter 1
  ACTIVATE_PC            'Activate PC
ENDIF
```

ADC

Syntax:

Measurement = ADC(*InputNo , Gain*)

***ADwin* Systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

When using an ***ADwin-Pro*** system, you wil also find a command with the same name, but it has a DIFFERENT MEANING. Please, see the document "***ADwin-Pro*** - System documentation - Programming in ***ADbasic***"

Description:

The command ADC measures an analog input and returns the measurement result as ADC digits. If amplification is required, this can be passed optionally as a second parameter. Gains of 1, 2, 4, and 8 can be selected. The command ADC first sets the multiplexer to the desired input channel and waits 4µs (***ADwin-GOLD*** 6,5 µs) for the multiplexer to be settled. Then the measurement is started and the value at the end of the ADC conversion is read out.

Note: The gain can only be set when using an ***ADwin*** board but not when using ***ADwin-light*** boards.

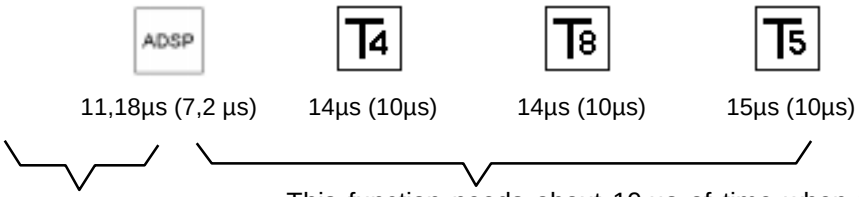
Time relationship when using an *ADwin-GOLD* system:

14,4µs (7,7 µs)

This function needs about 7,7 µs of time when the process runs with a delay shorter than 20 µs. Otherwise the function needs about 14 µs.

With an ***ADwin-GOLD*** system, this command accesses a 16-bit AD converter. For the (faster) 12-bit AD converter there is the command ADC12 available.

Time relationship when using an *ADwin* board and *ADwin-light*-board:



This function needs about 10 µs of time when the process runs with a delay shorter than 31 µs. Otherwise the function needs about 14 µs.

This function needs about 7 µs of time when the process runs with a delay shorter than 20 µs. Otherwise the function needs about 12 µs.

With ***ADwin-***boards, this command accesses a 12-bit AD converter.

Note: The shorter execution times at shorter delays are justified as follows: Because of the very short delay time, the compiler recognizes automatically, that there will not be enough time to set the multiplexer. The compiler concludes that the user intends to measure without setting the multiplexer anew and therefore it refuses automatically to accept the settling time of the multiplexer. (If you have such short delays, it is better to call the command SET MUX at least 4 µs (***ADwin-GOLD*** 6,5 µs) before you use the command ADC for the first time – otherwise the first measurement value may not be correct). If you have very short delays (GLOBALDELAY) it is generally recommended to use a combination of the commands SET_MUX, START_CONV, WAIT_EOC and READADC instead of the only command ADC.

! Important: If an input number >16 has been passed, the result is undefined.

Application example:

```
DIM iw AS INTEGER          'Declaration

EVENT:
iw = ADC(1,4)              'Measure Analog Input 1
                           'with a gain of 4
PAR_1 = iw                 'Write measurement in
                           'Parameter 1 where it can
                           'be fetched by the PC
```

Conversion of the ADC values (16 bit ADCs):

The ADCs used on the ***ADwin-GOLD*** system have a 16-bit resolution and therefore divide the selected measurement range (of 20 V) into 65.536 equally large steps.

The formula applies to the conversion of the input voltage:

$$Voltage = (Digits - 32768_{bipolar}) * \frac{Range}{65536 * Gain}$$

The values given in the table apply for a gain equal to one.

Input voltage range	ADC value			
	0	32768	65535	1Digit
-10...+10 V	-10 V	0 V	+9,999695 V	305,175 µV

Conversion of the 12-bit ADC values:

The ADCs used on the ***ADwin*** boards have a 12-bit resolution and therefore divide the selected measurement range into 4096 equally large steps.

The formula applies to the conversion of the input voltage:

$$Voltage = (Digits - 2048_{bipolar}) * \frac{Range}{4096 * Gain}$$

Note: With a unipolar setting the offset of 2048 is left out.

The values given in the table apply for a gain equal to one.

Input voltage range	**ADC value**			
	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:

Measurement value = ADC12(*InputNo , Gain*)

***ADwin* Systems:**

For ***ADwin-GOLD*** only:

Description:

The command ADC12 measures an analog input and returns the measurement result as ADC digits. If amplification is required, this can be passed optionally as a second parameter. Gains of 1, 2, 4, and 8 can be selected. The command ADC first sets the multiplexer to the desired input channel and waits 1.5 µs for the multiplexer to be settled. Then the measurement is started and the value at the end of the ADC conversion is read out.

Time relationship when using an *ADwin-GOLD* system:

3,1 µs

With an ***ADwin-GOLD*** system, this command accesses a 12-bit AD converter. For the (more exact) 16-bit AD converter, the command ADC is available.

Note: If you have very short delays (GLOBALDELAY) it is generally recommended to use a combination of the commands SET_MUX, START_CONV, WAIT_EOC and READADC12 instead of the only command ADC12.

...to be continued: command ADC12:

Application example:

```
DIM iw AS INTEGER          'Declaration

EVENT:
iw = ADC12(1,4)            'Measures analog input 1
                           'with a gain of 4
PAR_1 = iw                 'Write measurement in
                           'parameter 1 where it can
                           'be fetched by the PC
```

Conversion of the ADC values when using *ADwin-GOLD* 12-bit ADCs:

The ADCs used on the ***ADwin-GOLD*** system have a 12-bit resolution and therefore divide the selected measurement range (of 20 V) into 4096 equally large steps. In order to make a comparison with the measurement values of the 16-bit ADCs easier, the command ADC12 returns the result in high-priority bits (bits 31 to 4). Thus, the command ADC12 (1) presents the same result in the most significant bits, as the (16-bit) command ADC (1). The four least significant bits have always the value 0.

The formula applies to the conversion of the input voltage:

$$Voltage = (Digits - 32768_{bipolar}) * \frac{Range}{65536 * Gain}$$

The values given in the table apply for a gain = 1.

Input voltage range	ADC12-value			
	0	32768	65520	16Digits
-10...+10 V	-10 V	0 V	+9,99512 V	4,88 mV

AND

Syntax:

Value3 = *Value1* AND *Value2*

or with IF ... THEN and DO ... UNTIL

Expression1 AND *Expression2*

Description:

The operator AND is interpreted by the compiler either as a bitwise operator or as a Boolean operator.

As a **bitwise operator** it compares the individual bits of two values . In the result of this operation you can only find a 1 in those bits, which have a 1 at their corresponding bit positions in both values.

As **Boolean operator** in *statements* such as IF ... THEN or DO ... UNTIL, it determines for the AND operation of two statements, if a statement is true (1) or false (0).

Application example (as bitwise operator):

```
DIM value1, value2, value3 AS LONG

value1 = 0100B
value2 = 0110B
value3 = value1 AND value2
                          'result: value3 = 0100B
```

Note: As bitwise operator only for integer and long variables or constants.

Application example (as Boolean operator):

```
DIM f AS FLOAT
DIM value4 AS LONG

f = 3.14
IF ((f < 9.1) AND (f > 3.1)) THEN
  value4 = 1
ELSE
  value4 = 0
ENDIF
                          'result: value4 = 1
```

Note: If several AND (or OR) operators are used in one line, the correponding number of parentheses have to be set.

ARCCOS

Syntax:

value2 = ARCCOS(*value1*)

Description:

The function ARCCOS supplies the arccos of an argument.
Value1 must lie between -1 and +1 and the result is given in radians.

Time relationship:

FLOAT: 2,85µs FLOAT: 25µs FLOAT: 100µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = 0.5
value2 = ARCCOS(value1)  'Result: value2 = 1.0472
```

ARCSIN

Syntax:

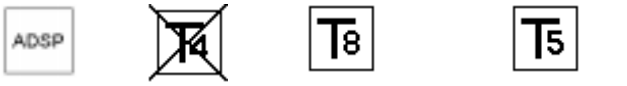
value2 = ARCSIN(*value1*)

Description:

The function ARCSIN supplies the arcsin of an argument.

Value1 must lie between -1 and +1 and the result is given in radians.

Time relationship:



FLOAT: 2,8µs FLOAT: 25µs FLOAT: 100µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = 0.5
value2 = ARCSIN(value1)  'Result: value2 = 0.5236
```

ARCTAN

Syntax:

value2 = ARCTAN(*value1*)

Description:

The function ARCTAN supplies the arctan of an argument.
The result is given in radians.

Time relationship:

FLOAT: 1,9µs

FLOAT: 29 µs

FLOAT: 120 µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = 0.5
value2 = ARCTAN(value1)  'Result: value2 = 0.4636
```

CLEAR_DIGOUT

Syntax:

CLEAR_DIGOUT(*outputNo*)

***ADwin* systems:**

For ***ADwin-GOLD***, ***ADwin-(light)*** systems**.**

When working with an ***ADwin-Pro*** system, this command must not be used. In this case there is the command DIGOUT. Please see the document: „***ADwin-Pro*** System Specifications - Programming in ***ADbasic***".

Description:

The command CLEAR_DIGOUT sets the Bit *OutputNo* of the digital output to zero.

Note: The digital outputs on the ***ADwin-GOLD*** system (default configuration) are numbered through from 16 to 31, on the ***ADwin*** boards from 0 to 15. You must use a constant between 0 and 15 for the *OutputNo.* Please pay attention to the fact that an ***ADwin-light*** board has only six digital outputs and that the six digital outputs on the ***ADwin*** board can only be used with the I/O add-on connector.

Variables must not be used in this command. If you want to define the output to be deleted by a variable, use the command DIGOUT_WORD.

When using an ***ADwin-GOLD*** system, the outputs have to be configured by the command CONF_DIO(12) before.

Application example:

```
DIM value AS INTEGER      'declaration

INIT:
CONF_DIO(12)              'configure dig. outputs
                          '(ADwin-GOLD only)
SET_DIGOUT(0)             'set dig. output DIO 16 or
                          '0

EVENT:
value = ADC(1)            'data acquisition

IF (value > 3000) THEN
  CLEAR_DIGOUT(0)         'reset dig. output DIO 16
                          '(ADwin-GOLD) or 0 (ADwin
                          'boards
ENDIF
```

CO4_CLEAR

Syntax:

CO4_CLEAR

***ADwin* systems:**

Only for ***ADwin-(light)-***boards, which are equipped with the counter options ***ADwin-CO1*** or ***ADwin-CO1L***, ***ADwin-CO2***, ***ADwin-CO3*** and ***ADwin-CO4***.

Description:

The command CO4_CLEAR clears all 16 bit counters on the ***ADwin*** or ***ADwin-light*** boards with counter option (in case the board is equipped with a counter option).

Application example:

```
DIM iw, flag AS INTEGER  'Declaration

EVENT:
iw = ADC(1,8)            'measurem. data acquisit.
IF (iw > 3000) THEN      'Compare with threshold
  IF (flag = 0) THEN     'Check whether flag
                         'is set
    CO4_CLEAR            'Reset counter
    CO4_START            'Start counter
    flag = 1             'Set flag
  ENDIF
ELSE
  IF (flag = 1) THEN     'Check whether flag is set
    CO4_STOP             'Stop counter
    PAR_1 = CO4_READ(1) 'Save counter reading
    flag = 0             'Reset flag
  ENDIF
ENDIF
```

CO4_READ

Syntax:

`value = CO4_READ(`*`number`*`)`

***ADwin* systems:**

Only for ***ADwin-(light)-***boards, which are equipped with the counter options ***ADwin-CO1*** or ***ADwin-CO1L***, ***ADwin-CO2***, ***ADwin-CO3*** and ***ADwin-CO4***.

Description:

The command CO4_READ reads out the 16 bit counter *number* when ***ADwin*** or ***ADwin-light*** boards with counter options are used.

Application example:

```
DIM iw, flag AS INTEGER  'Declaration

EVENT:
iw = ADC(1,8)            'measurement data
                         'acquisition
IF (iw > 3000) THEN      'Compare with threshold
  IF (flag = 0) THEN     'Check whether flag is set
    CO4_CLEAR            'Reset counter
    CO4_START            'Start counter
    flag = 1             'Set flag
  ENDIF
ELSE
  IF (flag = 1) THEN     'Check whether flag is set
    CO4_STOP             'Stop counter
    PAR_1 = CO4_READ(1)  'Save counter reading
    flag = 0             'Reset flag
  ENDIF
ENDIF
```

CO4_START

Syntax:

CO4_START

***ADwin* systems:**

Only for ***ADwin-(light)-***boards, which are equipped with the counter options ***ADwin-CO1*** or ***ADwin-CO1L***, ***ADwin-CO2***, ***ADwin-CO3*** and ***ADwin-CO4***.

Description:

The command CO4_START starts the 16 bit counter on the ***ADwin*** or ***ADwin-light*** boards with counter options.

Application example:

```
DIM iw, flag AS INTEGER  'Declaration

EVENT:
iw = ADC(1,8)            'measurement data
                         'acquisition
IF (iw > 3000) THEN      'Compare with threshold
  IF (flag = 0) THEN     'Check whether flag is set
    CO4_CLEAR            'Reset counter
    CO4_START            'Start counter
    flag = 1             'Set flag
  ENDIF
ELSE
  IF (flag = 1) THEN     'Check whether flag is set
    CO4_STOP             'Stop counter
    PAR_1 = CO4_READ(1) 'Save counter reading
    flag = 0             'Reset flag
  ENDIF
ENDIF
```

CO4_STOP

Syntax:

CO4_STOP

***ADwin* systems:**

Only for ***ADwin-(light)-***boards, which are equipped with the counter options ***ADwin-CO1*** or ***ADwin-CO1L***, ***ADwin-CO2***, ***ADwin-CO3*** and ***ADwin-CO4***.

Description:

The command CO4_STOP stopts the 16 bit counter on the ***ADwin*** or ***ADwin-light*** boards with counter options.

Application example:

```
DIM iw, flag AS INTEGER  'Declaration

EVENT:
iw = ADC(1,8)            'measurement data
                         'acquisition
IF (iw > 3000) THEN      'Compare with threshold
  IF (flag = 0) THEN     'Check whether flag is set
    CO4_CLEAR            'Reset counter
    CO4_START            'Start counter
    flag = 1             'Set flag
  ENDIF
ELSE
  IF (flag = 1) THEN     'Check whether flag is set
    CO4_STOP             'Stop counter
    PAR_1 = CO4_READ(1) 'Save counter reading
    flag = 0             'Reset flag
  ENDIF
ENDIF
```

CONF_DIO

Syntax:

CONF_DIO(*value*)

***ADwin systems*:**

The command is necessary and available only with the ***ADwin-GOLD*** system. All other systems do not need this command.

Description:

In an ***ADwin-GOLD*** system there are 32 inputs/outputs available, which can freely be configured. After power-up, all I/O connections are inputs. They can be configured via software in groups of eight as input or output.

If the digital inputs/outputs are configured with the ***ADbasic*** command CONF_DIO(12), (that means, DIO 0-15 are inputs and DIO 16-31 are outputs), it is also possible to access them with the ***ADbasic*** commands DIGIN_WORD, DIGOUT_WORD, DIGIN, SET_DIGOUT and CLEAR_DIGOUT. With regards to these commands, the programs on the ***ADwin-GOLD*** system are absolutely (source-code) compatible to the programs on the ***ADwin*** boards.

Therefore we recommend to use the configuration CONF_DIO(12) for all normal applications.

If other I/O configurations are needed, the corresponding hardware register must directly be read out, or it must be written into, by the commands PEEK- and POKE-. (for more information see the ***ADwin-GOLD*** hardware manual).

Note: If the ***ADwin-GOLD*** system is not configured with this command, no digital outputs can be set, because after power-up all I/O connections are inputs first.

...to be continued: comannd CONF_DIO:

Application example:

```
CONF_DIO(12)              'configures DIO 0 – 15 as
                          'inputs and DIO 16 - 31 as
                          'outputs
```

COS

Syntax:

value2 = COS(*value1*)

Description:

The function COS supplies the cosine of an argument which is indicated in radians.

Time relationship:

FLOAT: 1,3µs FLOAT: 28 µs FLOAT: 150 µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = -5.3
value2 = COS(value1)     'Result: value2 = 0.55...
```

DAC

Syntax:

DAC(*number*, *value*)

***ADwin* systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

When working with an ***ADwin-Pro*** system, there will be the same command. The description given here does not apply to this command. Please see the document: „***ADwin-Pro*** System Specifications - Programming in ***ADbasic***".

Description:

The command DAC outputs the specified *value* at the output *number*.

Note: The analog outputs are numbered through from 1 to 8. But please, note that on ***an ADwin-light*** board as well as on the ***ADwin*** board there are only two DACs. An ***ADwin*** board can optionally be equipped with four additional DACs. An ***ADwin-GOLD*** system can optionally be equipped with six additional DACs.

Application example:

```
REM Digital proportional controller
DIM sw, aw AS INTEGER    'Declaration
DIM v, act AS INTEGER    'Declaration

EVENT:
sw = PAR_1               'Setpoint value
v = PAR_2                'Gain
aw = sw - ADC(1)         'Compute control deviation
act = aw * v             'Compute actuating
                         'variable
DAC(1, act)              'Output actuating variable
```

...to be continued command DAC**:**

Conversion of the DAC values at 16-bit DACs:

The DACs used on the ***ADwin-GOLD*** system have a 16-bit resolution and therefore divide the selected measurement range (of 20 V) into 65536 equally large steps.

The following formula applies for the conversion of the output voltage:

$$Voltage = (Digits - 32768_{bipolar}) * \frac{Range}{65536 * Gain}$$

The following table shows the DAC output values for the corresponding output voltage range:

Output voltage range	**DAC value**			
	0	32768	65535	1Digit
-10...+10 V	-10 V	0 V	+9,999695 V	305,175 µV

Conversion of the DAC values at 12-bit DACs:

The DACs used on the ***ADwin*** boards have a resolution of 12 bits and therefore divide the selected output voltage range into 4096 equally large steps.

The formula applies to the conversion of the output voltage:

$$Voltage = (Digits - 2048_{bipolar}) * \frac{Range}{4096 * Gain}$$

Note: With a unipolar setting the offset of 2048 is left out.

The following table shows the DAC output values for the different settings of the output voltage range.

Output range	**DAC value**			
	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(value)`

Description:

The command DEC decrements the supplied value by 1.

Note: This command may not be used with data types other than INTEGER.

The instruction `DEC(value)` delivers the same result as `value=value-1`, but needs less processing time.

Application example:

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

INIT:
index=1000

EVENT:
DAC(1,DATA_1[index])      'Output value to DAC1
DEC(index)                'Decrement index
IF (index<1) THEN
  index=1000              'If first value of data is
ENDIF                     'reached,continue with the
                          'last
```

#DEFINE

Syntax:

`#DEFINE` *`NewName OldName`*

Description:

Using the function #DEFINE, parameters, data structures or any program lines can be replaced with a redefined name.

Application example:

```
#DEFINE Setpoint PAR_1
```

This command assigns the name `Setpoint` to `PAR_1`.

```
#DEFINE Measurements data_1
```

This command assigns the name `Measurements` to `data_1`.

! Important: Comments should never be placed in the `DEFINE` statement, because they would also be inserted.

DIGIN

Syntax:

result = DIGIN(*InputNo*)

***ADwin* Systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

Please do not use this command when working with an ***ADwin-Pro*** system.

Description:

The command DIGIN determines the value of the digital input *InputNo*. A TTL high level sets the variable *Result* to 1, a TTL low level to 0.

Note: The digital inputs are numbered through from 0 to 15. But please note that on an ***ADwin-light*** board there are only six digital inputs and that the 16 digital inputs on the ***ADwin*** board can only be used by the supplied I/O add-on connector.

Application example:

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

EVENT:
IF (DIGIN(0) = 1) THEN   'Check whether digital
                         'input 0 is set.
  DATA_1 = ADC(1)        'measurement data
                         'acquisition
ENDIF
```

DIGIN_WORD

Syntax:

result = DIGIN_WORD()

***ADwin* Systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

Please do not use this command when working with an ***ADwin-Pro*** system. In this case the commands DIGIN_WORD1 and DIGIN_WORD2 are available. Please see the document: „***ADwin-Pro*** System Specifications - Programming in ***ADbasic***“.

Description:

The function DIGIN_WORD() reads all 16 digital inputs in one operation.

The function returns a 16-bit value. One bit of the returned value is assigned to each digital input (see table). If a TTL high level is present on the input, then the corresponding bit is set to 1

Dig. Input	15	...	3	2	1	0
Bit	15	...	3	2	1	0
DEC-Value	32768	...	8	4	2	1

Computation example:

The function DIGIN_WORD() supplies the decimal value 11. From the bit pattern of decimal 11, you can deduce which digital inputs are set. In this example inputs 0, 1 and 3 are set, because the sum of the assigned bit values gives the decimal value 11.

Note: The digital inputs are numbered through from 0 to 15. But please note that on an ***ADwin-light*** board there are only six digital inputs and that the 16 digital inputs on the ***ADwin*** board can only be used by the supplied I/O add-on connector.

...to be continued command DIGIN_WORD:

Application example:

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

EVENT:
If (Digin_Word() AND 3 = 3) THEN
                          'Check whether inputs 0
                          'and 1 are set
  DATA_1 = ADC(1)         'measurement data
                          'acquisition
ENDIF
```

DIGOUT_WORD

Syntax:

DIGOUT_WORD(*Selection*)

***ADwin* Systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

Please do not use this command when working with an ***ADwin-Pro*** system. In this case the commands DIGOUT_WORD1 and DIGOUT_WORD2 are available. Please see the document: „***ADwin-Pro*** System Specifications - Programming in ***ADbasic***".

Description:

The command DIGOUT_WORD sets all the digital outputs on the ***ADwin/ADwin-light*** boards simultaneously to the value specified by *Selection*. *Selection* is a 16-bit value with each digital output being assigned a bit of this value (see table).

***ADwin-GOLD* system: digital output (DIO)**	31	30	...	18	17	16
***ADwin-I ADwin-light* boards: digital output**	15	14	...	2	1	0
Bit	15	14	...	2	1	0
DEC value	32768	16384	...	4	2	1

Computation example:

The outputs 0, 2 and 14 are to be set. You derive the bit pattern 0100000000000101, which corresponds to the decimal value 16389 and write: DIGOUT_WORD(16389).

! Important: All outputs for which the corresponding bit in the value *Selection* is not set, are deleted! In the above example these are the outputs 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 and 15.

Note: The digital outputs of an ***ADwin-GOLD*** system (default configuration) are numbered through from 16 to 31, those of the ***ADwin*** board from 0 to 15. In both cases you have to write the specified value into the least significant 16 bits of *selection*. But please note that on an ***ADwin-light*** board there are only six digital inputs and that the 16 digital inputs on the ***ADwin*** board can only be used by the supplied I/O add-on connector.

When using an ***ADwin-GOLD*** system the outputs have to be configured once by the command CONF_DIO (12).

Application example:

```
INIT:
CONF_DIO(12)              'configure digital outputs
                          '(ADwin-GOLD only)

EVENT:
DIGOUT_WORD(7)            'set digital outputs 16,
                          '17 a. 18 (ADwin-GOLD)
                          'or 0, 1 a. 2 (ADwin-
                          'boards), all other
                          'outputs are deleted!
```

DIM

Syntax:

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

Description:

Declares one or more variables *V1*, *V2* etc. of the type *type*. If you state a field length *L1*, *L2* etc. after the variable name, the compiler creates a single-dimension array. The arrays of the global DATA variables can in addition also be defined as FIFO ring buffers. Further information on this special type of variable can be found in the chapter about the command FIFO.

Note: The types available are SHORT, INTEGER, LONG and also FLOAT for the T805. Depending on the type of processor, the declarations may need varying amounts of memory space. Further information about this can be found in Chapter 4.4.3 (User-definable variables).

Application examples:

```
DIM var1 AS INTEGER          'Declares var1 as integer
                             'variable
DIM array1[1000] AS SHORT
                             'Declares array1 with a
                             'field length of 1000
                             'elements
DIM DATA_20[1000] AS INTEGER AS FIFO
                             'Declares the global
                             'variable DATA_20 with a
                             'length of 1000 elements
                             'as FIFO memory
```

Note: The type of memory can be additionally indicated for the ADSP (page 22/Table 4-1).

DO ... UNTIL

Syntax:

```
DO
   instruction block
UNTIL (condition)
```

Description:

The instruction block is executed repeatedly until *condition* is true.

! Important: In a high priority process the DO ... UNTIL loop cannot be interrupted by any other process. Therefore, the processor of the ***ADwin*** system cannot respond to other events during the period of loop execution. The DO ... UNTIL loop must therefore only be used in high priority processes when the number of loop executions is kept small.

Application example:

```
DIM value AS INTEGER

EVENT:
DO
   value = ADC(1, 4)       'Read out measurement
UNTIL (value < 2048)       'Repeat loop until the
                           'value is greater than or
                           'equal to 2048
ACTIVATE_PC                'Activate PC
```

END

Syntax:

END

Description:

The command END terminates the process.

Note: The commands in the segment FINISH: are still executed, if present, after the END command.

Application example:

```
EVENT:
IF (ADC(1) > 3000) THEN  'Measure and compare
  SET_DIGOUT(0)          'Set digital output
  END                    'Terminate process
ENDIF
```

EXP

Syntax:

value2 = EXP(*value1*)

Description:

The function EXP supplies the exponential value of an argument to the base e.

Time relationship:

FLOAT: 1,3µs

FLOAT: 32 µs

FLOAT: 130 µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = 5
value2 = EXP(value1)     'Result:
                         'value2 = 148.41...
```

FIFO

Syntax:

DIM DATA_*X*[*length*] AS *Var_type* AS FIFO

Description:

Dimensions the variable DATA_*X* as FIFO memory with *Length* elements of type *Var_type*. A number between 1 and 200 *must* be indicated instead of *X*.

Note: Since ***ADbasic*** manages the FIFO internally as a data set, the same data set number *must not be used simultaneously* as the number of a FIFO *and* a normal DATA variable.

! **Important:** The FIFOs are not automatically cleared on starting, so they should therefore be cleared with the command FIFO_CLEAR(*DataSetNo*), e. g. in the segment INIT: in the program.

! **Important:** If you write data faster to the FIFO than you read it out, the FIFO then at some point becomes full and data is lost.

Application example:

```
DIM DATA_20[1000] AS INTEGER AS FIFO
                          'Dimensions the global
                          'variable DATA_20 with a
                          'length of 1000 elements
                          'as FIFO memory
```

Note: The type of memory can be additionally indicated for the ADSP (page 22/Table 4-1).

FIFO_CLEAR

Syntax:

FIFO_CLEAR(*DatasetNo*)

Description:

The command FIFO_CLEAR clears the contents of the FIFO with the number *DatasetNo*.

! Important: The FIFOs are not automatically cleared on starting. Therefore, they should be cleared with this command in the INIT: segment of the program.

Application example:

```
DIM DATA_1[20000] AS SHORT AS FIFO
                          'Declaration

INIT:
FIFO_CLEAR(1)             'Clear FIFO

EVENT:
IF (FIFO_EMPTY(1) > 1) THEN
                          'querying the number of
                          'free places in the FIFO
  DATA_1 = ADC(1)         'measure analog input 1
                          'and save in the FIFO
ENDIF
```

FIFO_EMPTY

Syntax:

value = FIFO_EMPTY(*DatasetNo*)

Description:

The command FIFO_EMPTY determines the memory space in the FIFO with the number *DatasetNo*.

Note: Before writing data to the FIFO you should check with this command if there is enough free space in the FIFO.

Application example:

```
DIM DATA_1[20000] AS SHORT AS FIFO
                          'Declaration

EVENT:
IF (FIFO_EMPTY(1) > 1) THEN
                          'Check free space in the
                          'FIFO
  DATA_1 = ADC(1)         'Measure Analog Input 1
                          'and save in the FIFO
ENDIF
```

FIFO_FULL

Syntax:

value = FIFO_FULL(*DatasetNo*)

Description:

The command FIFO_FULL determines the amount of occupied memory in the FIFO with the number *DatasetNo*.

Note: Before reading out data from the FIFO, you should check with this command if there is still data in the FIFO.

Application example:

```
DIM DATA_1[20000] AS SHORT AS FIFO
                          'Declaration

EVENT:
IF (FIFO_FULL(1) > 0) THEN
                          'Check if the FIFO still
                          'contains data
  DAC(1, DATA_1)          'Output a value from the
                          'FIFO on Analog Output 1
ENDIF
```

FOR ... NEXT

Syntax:

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

Description:

The *Instruction block* within the FOR ... NEXT loop is repeated until the value of *X* is greater than the value of *Y*. The value of *X* is increased by *Z* after each loop.

Note: The statement STEP *Z* can be left out. The value 1 is taken for *Z* in this case.

X, Y and Z can be substituted by constant numerical values. These must be of the type Integer and *Z* must only take on positive values.

Contrary to other programming languages, the *instruction block* will at least once be executed in Basic, even if *X* is greater than *Y*.

Also the counter variable (here *i*) must be declared in the program header as an integer variable.

! Important: In a high priority process the FOR ... NEXT loop cannot be interrupted by another process. This means that during the period of loop processing it is not possible for the processor of the ***ADwin*** system to respond to other events. The FOR ... NEXT loop must therefore only be used in high priority processes when the number of loops is kept small.

FUNCTION ... ENDFUNCTION

Syntax:

FUNCTION *name*(*value1, value2, ...*) AS *Type*
... (commands)
ENDFUNCTION

Description:

The function *Name* is defined. On calling, a numerical value is transferred to the variables *valueX* and the commands between FUNCTION and ENDFUNCTION are executed. The data type to which the values of the function are transferred is defined as AS Type.

Notes: Local variables can be defined at the start of each function.

A function can be placed at the start of the ***ADbasic*** program (before the INIT: block), at the end of the ***ADbasic*** program (after the FINISH: block) or in its own file which you link with an INCLUDE command.

Application example:

```
FUNCTION means(w1, w2, w3) AS FLOAT
  DIM sum AS FLOAT       'computes the means of the
                         '3 values w1, w2 and w3
  sum = w1 + w2 + w3
  means = sum/3
ENDFUNCTION
```

Calling the function means is made by the program line:

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

IF ... THEN ... {ELSE}

Syntax:

IF (*condition*) THEN
  instruction block
{ELSE}
  {*instruction block*}
ENDIF

or

IF (*condition*) THEN *instruction*

Description:

The control structure IF enables the *instruction block* to be executed in dependence of *condition*.

Application example:

```
DIM value AS INTEGER     'Declaration

EVENT:
value = ADC(1)           'Acquire measurement
IF (value > 3000) THEN   'Start control structure
  CLEAR_DIGOUT(1)        'Reset DIGOUT 1
  SET_DIGOUT(0)          'Set DIGOUT 0
ELSE
  CLEAR_DIGOUT(0)        'Reset DIGOUT 0
  SET_DIGOUT(1)          'Set DIGOUT 1
ENDIF                    'End of control structure
```

INC

Syntax:

`INC(value)`

Description:

The command `INC` increments the passed `value` by 1.

Notes: This command may not be used with data types other than INTEGER .

`INC(value)` delivers the same result as `value=value+1,` but needs less processing time.

Application example:

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

INIT:
index=1

EVENT:
DATA_1[index] = ADC(1)   'Store in DATA
INC(index)               'Increment index
IF (index>1000) THEN END 'Finish program after 1000
                         'samples
```

#INCLUDE

Syntax:

```
#INCLUDE FileName
```

The file *FileName* is linked with all the definitions and programs it contains. *FileName* should also denote the complete path specification. Otherwise ***ADbasic*** is only searching in the standard-include-directory. (see menu **Options ⇨ Directory**).

Notes: The INCLUDE command must be placed at the start of the ***ADbasic*** program.

You should always specify the complete path name, because otherwise only the current directory is searched.

Application example:

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

LINKIN

Syntax:

LINKIN(*channel*, *value*, *number*)

Description:

The command LINKIN reads as many bytes as specified by *Number* from the link interface *Channel*. The read bytes are saved in the variable or in the data set *Value*.

! Important: Since the only available Link 0 of the ADSP21062 is used for the communication between the PC/***ADlink*** or ***ADpcmcia***, this command is not implemented in the processor.

Time relationship:

The command LINKIN interrupts the current process until all *Number* bytes have been transferred. Therefore, this command should not be used while a measurement is being taken.

Notes: The processor was developed for building multiprocessor systems. To simplify the data interchange between the processors, each processor module has four serial interfaces. The LINK commands are used for the data transfer via these interfaces and are numbered through from 0 to 3.

Further information about the configuration and application can be taken from the hardware manual for your ***ADwin*** board or from the hardware manual of the ***ADwin Pro*** system.

...to be continued command LINKIN:

Application example:

```
DIM value as LONG

EVENT:
LINKIN(0, value, 4)       'Reads a long integer
                          'value from LINK 0
```

LINKOUT

Syntax:

LINKOUT(*channel*, *value*, *number*)

Description:

The command LINKOUT outputs as many bytes via the link interface *Channel* as specified by *Number*. The bytes to be output are located in the variable or in the data set *Value*.

! **Important:** Since the only available Link 0 of the ADSP21062 is used for the communication between the PC/***ADlink*** or ***ADpcmcia***, this command is not implemented in the processor.

Time relationship:

The command LINKOUT interrupts the current process until all *Number* bytes have been transferred. Therefore, this command should not be used while a measurement is being taken.

Notes: The processor was developed for building multiprocessor systems. To simplify the data interchange between the processors, each processor module has four serial interfaces. The LINK commands are used for the data transfer via these interfaces and are numbered through from 0 to 3.

Further information about the configuration and application can be taken from the hardware manual for your ***ADwin*** board or from the hardware manual of the ***ADwin Pro*** system.

...to be continued command LINKOUT:

Application example:

```
DIM value as LONG

EVENT:
LINKOUT(2, value, 4)      'Outputs a long integer
                          'value on LINK 2
```

LOG

Syntax:

value2 = LOG(*value1*)

Description:

The function LOG supplies the logarithm of an argument.

Time relationship:

FLOAT: 1,45µs

FLOAT: 35 µs

FLOAT: 145 µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = 5.3
value2 = LOG(value1)     'Result:
                         'value2 = 0.724...
```

NOT

Syntax:

value2 = NOT(*value1*)

Description:

Inverts the individual bits of *value1*.

Note: *Value1* is expected to be from the type INTEGER or LONG. If *value1* is from the type FLOAT, *value1* will be converted to LONG beforehand. Please consider that the decimal places are cut off, and that FLOAT and LONG have different value ranges.

Application example:

```
DIM value1 AS LONG
DIM value2 AS LONG

value1 = 11111111111111111111111111111101B ' = -3
value2 = NOT(value1)
                    'Result: value2 = 010B = 2
```

Note: NOT cannot be used for a logical negation.

Example:

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

WRONG !!

OR

Syntax:

value3 = *value1* OR *value2*

or with IF ... THEN and DO ... UNTIL

expression1 OR *expression2*

Description:

The operator OR is interpreted by the compiler either as a bitwise operator or as a Boolean operator.

As a **bitwise operator** it compares the individual bits of two values . In the result of this operation you can only find a 1 in those bits, which have a 1 at their corresponding bit positions in both values.

As **Boolean operator** in *statements* such as IF ... THEN or DO ... UNTIL, it determines for the OR operation of two statements, if a statement is true (1) or false (0).

Application example (as bitwise operator):

```
DIM value1, value2, value3 AS LONG

value1 = 0100B
value2 = 0110B
value3 = wert1 OR wert2
                          'Result: value3 = 0110B
```

Note: as bitwise operator only for integer and long variables or constants.

Application example (as Boolean operator):

```
DIM x AS LONG
DIM value4 AS LONG

x = 15
IF ((x < 9) OR (x > 3)) THEN
  value4 = 1
ELSE
  value4 = 0
ENDIF
                          'Result: value4 = 1
```

Note: If several AND (or OR) operators are used in one line, the correponding number of parentheses have to be set.

PEEK

Syntax:

`value = PEEK(`*`address`*`)`

Description:

The command PEEK reads the value stored in *`address.`*

Application example:

```
value = PEEK(20400030H)  'reads the value of memory
                         'address 20400030H.
```

Note: On ***ADwin*** boards with ADSP processor and in ***ADwin-GOLD*** systems the data register of ADC1 can be found at the address 20400030H used in the example above. Further information about configuration and application can be taken from the hardware manual for your ***ADwin*** board or ***ADwin-GOLD*** system.

POKE

Syntax:

POKE(*address*, *value*)

Description:

The command POKE saves *value* to the memory location *address*.

Application example:

```
POKE(20400050H, 10000)    'saves 10000 to the
                          'address 20400050H
POKE(20400060H, 20000)    'saves 20000 to the
                          'address 20400060H
START_CONV (4)            'starts all DACs at the
                          'same time
```

Note: On ***ADwin*** boards with ADSP processor and in ***ADwin-GOLD*** systems the data registers of the DAC1 and DAC2 can be found at the address 20400050H and 20400060H used in the example above. Consequently this example corresponds to the command sequence DAC(1,10000) DAC(2,20000), but with the difference, that conversion is started after short time intervals when using the DAC commands. Otherwise conversion is started exactly at the same time. (The DAC command needs a little bit more execution time, because it checks the output value if there are limit values).

Further information about configuration and application as well as further important *addresses* can be taken from the hardware manuals for your ***ADwin*** boards or ***ADwin-GOLD*** systems.

! Important: With the command POKE the data are directly written to the indicated memory address. All data being at this address will be deleted. If there are program data, the program will be destroyed.

READADC

Syntax:

value = READADC(*AdcNo*)

***ADwin* Systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

When using an ***ADwin-Pro*** system, you wil also find a command with the same name, but it has a DIFFERENT MEANING. Please, see the document "***ADwin-Pro*** - System Specifications - Programming in ***ADbasic***".

Description:

The command READADC reads a value from the A/D converter with the number *AdcNo*.

Notes: This function directly addresses one of the two A/D converters ADC1 or ADC2.

Only the values 1 or 2 are valid for *AdcNo*.

Please note that an ***ADwin-light*** board has only one ADC!

Application example:

```
DIM value1, value2 AS INTEGER
                          'Declaration

EVENT:
SET_MUX(9)                'Set multiplexers for both
                          'ADCs

  Wait 4µs (ADwin-GOLD 6,5 µs) for the settling of
                          the multiplexer

START_CONV(3)             'Start A/D conversion for
                          'both ADCs
WAIT_EOC(3)               'Wait for both ADCs to
                          'end conversions
```

```
value1 = READADC(1)       'Read value from ADC1
value2 = READADC(2)       'Read value from ADC2
```

Conversion of the ADC values at 16 bit ADCs:

The ADCs used on the ***ADwin-GOLD*** system have a 16-bit resolution and therefore divide the selected measurement range (of 20 V) into 65.536 equally large steps.

The formula applies to the conversion of the input voltage:

$$Voltage = (Digits - 32768_{bipolar}) * \frac{Range}{65536 * Gain}$$

The values given in the table apply for a gain equal to one.

Input voltage range	READADC value			
	0	32768	65535	1Digit
-10...+10 V	-10 V	0 V	+9,999695 V	305,175 µV

Conversion of the 12-bit ADC values:

The ADCs used on the ***ADwin*** boards have a 12-bit resolution and therefore divide the selected measurement range into 4096 equally large steps.

The formula applies to the conversion of the input voltage:

$$Voltage = (Digits - 2048_{bipolar}) * \frac{Range}{4096 * Gain}$$

Note: With a unipolar setting the offset of 2048 is left out.

The values given in the table apply for a gain equal to one.

Input voltage range	**READADC value**			
	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:

value = READADC12(*AdcNo*)

***ADwin* systems:**

Only for ***ADwin-GOLD***.

Description:

The command READADC12 reads a value from the 12 bit A/D-converter with the number *AdcNo*.

Note: This function accesses directly one of the two A/D-converters ADC12-1 or ADC12-2.
For *AdcNo* *only* the values 1 or 2 apply.

Application example:

```
DIM value1, value2 AS INTEGER

EVENT:
SET_MUX(9)                  'set multiplexer for both
                            'ADCs

 wait 1,5 µs for the settling of the multiplexer

START_CONV(24)              'start AD-conversion for
                            'both 12-bit ADCs
WAIT_EOC(24)                'wait for end of conver-
                            'sion of both 12-bit ADCs
value1 = READADC12(1)       'read value of ADC12-1
value2 = READADC12(2)       'read value of ADC12-2
```

...to be continued: command READADC12:

Conversion of the ADC values when using *ADwin-GOLD* 12-bit ADCs:

The ADCs used on the ***ADwin-GOLD*** system have a 12-bit resolution and therefore divide the selected measurement range (of 20 V) into 4096 equally large steps. In order to make a comparison with the measurement values of the 16-bit ADCs easier, the command ADC12 returns the result in high-priority bits (bits 31 to 4). Thus, the command ADC12 (1) presents the same result in the most significant bits, as the (16-bit) command ADC (1). The four least significant bits have always the value 0.

The formula applies to the conversion of the input voltage:

$$Voltage = (Digits - 32768_{bipolar}) * \frac{Range}{65536 * Gain}$$

The values given in the table apply for a gain = 1.

input voltage range	READADC12-Wert			
	0	32768	65520	16Digits
-10...+10 V	-10 V	0 V	+9,99512 V	4,88 mV

READ_TIMER

Syntax:

value = READ_TIMER()

Description:

The command READ_TIMER reads the count rate for the current process.

Notes: All processors in ***ADwin*** systems have two integrated timers.

When using the processors T400, T450 and T805 the timer is incremented by one every microsecond for a high priority process. With a low priority process this occurs every 64 µs.

When using an ADSP processor the timer is incremented by one every 25 ns for a high priority process. With a low priority process this occurs every 100 µs.

More information and examples can be found in chapter 5.2 "Checking execution times with timer functions".

Application example:

```
DIM timer_reading AS INTEGER

EVENT:
timer_reading = READ_TIMER()
```

REM

Syntax:

REM *comment*

Description:

The complete text located after REM in the same line is ignored by ***ADbasic*** during compilation. This enables you to insert comments into the ***ADbasic*** program.

Notes: The command REM only applies to the line in which it is used. If a comment requires more than one text line, then you must begin each line with the REM command.

A comment line can also be introduced by the single quotation mark '.

If the comment is to be accommodated in the same line in which an ***ADbasic*** command is located, then the REM must be preceded by a colon. If you use the quotation mark, the colon is not needed.

Application example:

```
REM This is a comment, which needs more than
REM one text line

'This is also a comment line

DIM min AS INTEGER       :REM variable for

                         :REM min. value

DIM max AS INTEGER       'Variable for max. value
```

SET_DIGOUT

Syntax:

SET_DIGOUT(*OutputNo*)

***ADwin* systems:**

For ***ADwin-GOLD***, ***ADwin-(light)*** systems**.**

When working with an ***ADwin-Pro*** system, this command must not be used. In this case there is the command DIGOUT. Please see the document: „***ADwin-Pro*** System Specifications - Programming in ***ADbasic***".

Description:

The command SET_DIGOUT sets the digital output *OutputNo*.

Note: The digital outputs on the ***ADwin-GOLD*** system (default configuration) are numbered through from 16 to 31, on the ***ADwin*** boards from 0 to 15. You must use a constant between 0 and 15 for the *OutputNo.* Please pay attention to the fact that an ***ADwin-light*** board has only six digital outputs and that the six digital outputs on the ***ADwin*** board can only be used with the I/O add-on connector.

Variables must not be used in this command. If you want to define the output to be deleted by a variable, use the command DIGOUT_WORD.

When using an ***ADwin-GOLD*** system, the outputs have to be configured by the command CONF_DIO(12) before.

Application example:

```
DIM value AS INTEGER     'declaration

INIT:
CONF_DIO(12)             'configure dig. outputs
                         '(ADwin-GOLD only)

EVENT:
value = ADC(1)           'data acquisition

IF (value > 3000) THEN
  SET_DIGOUT(0)          'set dig. output DIO 16
                         '(ADwin-GOLD) or 0 (ADwin
                         'boards
ENDIF
```

SET_MUX

Syntax:

SET_MUX(*selection*)

***ADwin* Systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

When using an ***ADwin-Pro*** system, you wil also find a command with the same name, but it has a DIFFERENT MEANING. Please, see the document "***ADwin-Pro*** - System Specifications - Programming in ***ADbasic***".

Description:

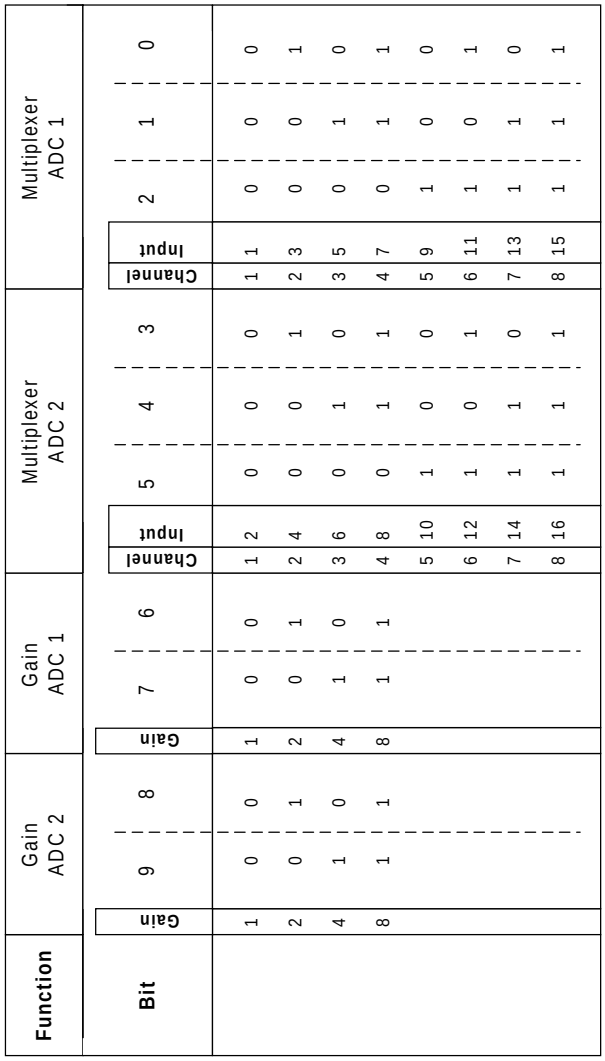
The command SET_MUX sets the multiplexers and amplifiers at the specified input. Since the multiplexers have 8 inputs, 3 bits are used. The amplifiers have four possible settings (1, 2, 4, or 8fold) which need a further two bits. The bits are set by either directly stating the binary number or by previously converting them into HEX or DEC code. For HEX and BIN codes, label the numbers with the appropriate letters (H for HEX, B for BIN).

Note: Please note that the multiplexer needs about 4 µs (***ADwin-GOLD*** 16 bit: 6.5 µs or 12 bit: 1.5 µs) until it has settled. You must therefore provide a waiting period or insert a command which has an execution time that does not fall below the settling time of the multiplexer.

Please note, that on the ***ADwin-light*** board there is only one ADC and that no gain can be set for its inputs.

Please note, too, that on an ***ADwin*** board you can only use the 8 analog inputs per multiplexer with the supplied I/O add-on connector.

The bit combinations relevant to the settings can be seen in the following table:



Function	Bit	Gain ADC 2					Gain ADC 1					Multiplexer ADC 2						Multiplexer ADC 1					
		9	8	Gain			7	6	Gain			5	4	3	Input	Channel		2	1	0	Input	Channel	
				1					1						2	1					1	1	
				2					2						4	2					3	2	
				4					4						6	3					5	3	
				8					8						8	4					7	4	
															10	5					9	5	
															12	6					11	6	
															14	7					13	7	
															16	8					15	8	
		0	0				0	0				0	0	0				0	0	0			
		0	0				0	0				0	0	0				0	0	1			
		0	1				0	1				0	0	1				0	1	0			
		0	1				0	1				0	0	1				0	1	1			
		1	0				1	0				0	1	0				1	0	0			
		1	0				1	0				0	1	0				1	0	1			
		1	1				1	1				0	1	1				1	1	0			
		1	1				1	1				0	1	1				1	1	1			

...to be continued command SET_MUX:

Computation examples:

You would like to set the multiplexer for ADC1 to channel 3 and need gain 8 *and* simultaneously set the multiplexer for ADC2 to channel 4 at a gain of 2:

Bit pattern: **01 11 011 010** Command: **SET_MUX(474)**

As an alternative to this notation in DEC code, in BIN code it would be written: **SET_MUX(0111011010B)**

Application example:

```
DIM value1 AS INTEGER     'Declaration

EVENT:
SET_MUX(0)                'Set multiplexer for ADC1
                          'to channel 1
  Wait 4µs (ADwin-GOLD 6.5 µs) for the settling of
                          the multiplexer
START_CONV(1)             'Start A/D conversion ADC1
WAIT_EOC(1)               'Wait for end of
                          'conversion for ADC1
value1 = READADC(1)       'Read value from ADC1
```

SHIFT_LEFT

Syntax:

value2 = SHIFT_LEFT(*value1, number*)

Description:

The command SHIFT_LEFT shifts all bits of *Value1* by *Number* places to the left. The empty places to the right are filled with zeroes.

Note: *value1* is expected to be of the LONG or INTEGER type. If *value1* is from the type FLOAT, *value1* will be converted to LONG beforehand. Please consider that the decimal places are cut off, and that FLOAT and LONG have different value ranges.

This command can be used, in order to multiply the variable *value1* with its integer multiple of the number 2. Here the execution time is shorter than the time needed for a comparable multiplication command, that means *value2* = SHIFT_LEFT(*value1,3)* is faster than *value2 = value1 * 8* .

number	**multiplicator**
1	2
2	4
3	8
.	.
.	.
.	.

Application example:

```
DIM value1,value2 AS LONG

value1 = 1024
value2 = SHIFT_LEFT(value1, 2)
                          ' Result: value2 = 4096
```

SHIFT_RIGHT

Syntax:

Value1 = SHIFT_RIGHT(*Value2*, *Number*)

Description:

The command SHIFT_RIGHT shifts all bits of *Value2* by *Number* places to the left. The empty places to the right are filled with zeroes.

Note: *value1* is expected to be of the LONG or INTEGER type. If *value1* is from the type FLOAT, *value1* will be converted to LONG beforehand. Please consider that the decimal places are cut off, and that FLOAT and LONG have different value ranges.

If the variable *value1* is a positive number, this command can be used, in order to divide the variable *value1* by integer multiples of the number 2. Here the execution time is shorter than the time needed for a comparable division command, that means *value2* = SHIFT_LEFT(*value1,3)* is faster than *value2 = value1 / 8* is shorter than *value2 =* SHIFT_RIGHT *(value1,3)*.

number	**divisor**
1	2
2	4
3	8
.	.
.	.
.	.

Application example:

```
DIM value1,value2 AS LONG

value1 = 1024
```

```
value2 = SHIFT_RIGHT(value1, 3)
                          ' Result: value2 = 128
```

SIN

Syntax:

value2 = SIN(*value1*)

Description:

The function SIN supplies the sine of an argument specified in radians.

Time relationship:

FLOAT: 1,2µs

FLOAT: 24 µs

FLOAT: 72 µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = -5.3
value2 = SIN(value1)     'Result: value2 = 0.83...
```

SQRT

Syntax:

value2 = SQRT(*value1*)

Description:

The function SQRT supplies the square root of *value1*.

Time relationship:

FLOAT: 0,775µs

SHORT: 25µs

FLOAT: 14µs

FLOAT: 40 µs

Application example:

```
DIM value1, value2 AS SHORT

EVENT:
value1 = 16
value2 = SQRT(value1)    'Result: value2 = 4
```

START_CONV

Syntax:

START_CONV(*selection*)

***ADwin* Systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

When using an ***ADwin-Pro*** system, you wil also find a command with the same name, but it has a DIFFERENT MEANING. Please, see the document "***ADwin-Pro*** - System Specifications - Programming in ***ADbasic***".

Description:

The command START_CONV starts the ADCs or DACs specified with *Selection*. Only 4 of the 5 least significant bits of *Selection* are used for selecting the ADCs: bit 0 starts ADC1, bit 1 starts ADC2.

Bit(s)	DEC value	*ADwin-Gold*	*ADwin board*	*ADwin-light*-board
0	1	ADC16-1	ADC1	ADC1
1	2	ADC16-2	ADC2	-
0,1	3	ADC16-1 ***and*** ADC16-2	ADC1 ***and*** ADC2	-
2	4	***all*** DACs	***all*** DACs	***all*** DACs
0,2	5	ADC16-1 ***and all*** DACs	ADC1 ***and all*** DACs	ADC1 ***and all*** DACs
3	8	ADC12-1	-	-
4	16	ADC12-2	-	-
3,4	24	ADC12-1 ***and*** ADC12-2	-	-
0,3	9	ADC16-1 ***and*** ADC12-1	-	-
1,4	18	ADC16-2 ***and*** ADC12-2	-	-
0,1,3,4	27	***all*** ADCs	-	-

Note: If you set bit 2, you can start all DACs at the same time. See the example for the command Poke.

Please consider that ADC1 and ADC2 can either be a 12 or 16 bit analog-digital converter. For further information see the hardware manual for your ***ADwin-(light)*** boards.

! Important: When using an ADSP *selection* must be a constant, the reason is code optimization. When using other processors, *selection* may also be a variable.

Computation example:

You only want to start ADC2 or ADC16-2 respectively:

Bit pattern: **10** Command: **START_CONV (2)**

Application example:

```
DIM value1 AS INTEGER

EVENT:
SET_MUX(0)                  'Set multiplexer for ADC1
                            'or ADC16-1 to channel 1
  Wait 4µs (ADwin-Gold 6.5 µs) for the settling of
                            the multiplexer
START_CONV(1)               'Start ADC1 A/D conversion
WAIT_EOC(1)                 'Wait for end of
                            'conversion for ADC1 or
                            'ADC16-1
value1 = READADC(1)         'Read value
```

START_PROCESS

Syntax:

START_PROCESS(*ProcessNo*)

Description:

The command START_PROCESS starts the ***ADbasic*** process with the number *ProcessNo*.

Notes: The process to be started must be first loaded onto the ***ADwin*** board.

If the process is already running, this command has no effect.

Application example:

```
EVENT:
IF (ADC(1) > 3072) THEN
  START_PROCESS(2)       'Start process 2, if above
                         'threshold value
  END
ENDIF
```

STOP_PROCESS

Syntax:

STOP_PROCESS(*ProcessNo*)

Description:

The command STOP_PROCESS stops the ***ADbasic*** process with the number *ProcessNo*.

Note: If the process is not running, this command has no effect.

Application example:

```
EVENT:
IF (ADC(1) > 3072) THEN
  STOP_PROCESS(2)        'Stop process 2, if above
                         'limit value
  END
ENDIF
```

SUB ... ENDSUB

Syntax:

SUB *name*(*value1, value2, ...*)
... (commands)
ENDSUB

Description:

The subroutine *Name* is defined. On calling, a numerical value is transferred to the variables *valueX* and the commands between SUB and ENDSUB are executed.

Notes: Local variables can be defined at the start of each subroutine.

A subprogram can be placed at the start of the ***ADbasic*** program (before the INIT: block), at the end of the ***ADbasic*** program (after the FINISH: block) or in its own file which you link with an #INCLUDE command.

Application example:

```
SUB Fast_Dac1(value1)
REM Outputs value1 on Analog Output 1
  POKE(50H, value1)
  POKE(10H, 3)
ENDSUB
```

Calling the subroutine *name* occurs with the program line:

```
Fast_Dac1(NewValue)
```

TAN

Syntax:

Value2 = TAN(*Value1*)

Description:

The function TAN supplies the tangent of an argument which is specified in radians.

Time relationship:

FLOAT: 1,275µs

FLOAT: 30 µs

FLOAT: 150 µs

Application example:

```
DIM value1, value2 AS FLOAT

EVENT:
value1 = 5.3
value2 = TAN(value1)     'Result:
                         'value2 = -1.50...
```

VR6_CLEAR

Syntax:

VR6_CLEAR(*Selection*)

***ADwin* systems:**

Only for ***ADwin-(light)-***boards, which are equipped with the counter option ***ADwin-VR6***, and for ***ADwin-X-VR6-***Karten.

Description:

The command VR6_CLEAR clears the up/down counters defined by *Selection*. The corresponding bit must be set in *Selection* for each counter which is to be cleared. You can clear a number of counters simultaneously. The following table shows the relationship between the bit to be set and the counter number:

counter number	6	5	4	3	2	1
bit	5	4	3	2	1	0
DEC value	32	16	8	4	2	1

Computation examples:

You would like to clear counter 5, i.e. you must set bit 4.

Bit pattern: **010000** Command: **VR6_CLEAR(16)**

Alternatively to this notation in DEC code, in BIN code it would be written: **VR6_CLEAR(010000B)**

Application example:

```
EVENT:
VR6_CLEAR(63)                'Clears all VR6 counters
```

VR6_LATCH

Syntax:

VR6_LATCH(*selection*)

***ADwin* systems:**

Only for ***ADwin-(light)-***boards, which are equipped with the counter option ***ADwin-VR6***, and for ***ADwin-X-VR6-***Karten.

Description:

The command VR6_LATCH transfers the current counter reading of the up/down counter specified by *selection* into the latch. A bit must be set in *selection* for each counter. You can transfer a number of counter readings simultaneously because each counter has its own latch. The following table shows the relationship between the bit to be set and the counter number.

counter number	6	5	4	3	2	1
bit	5	4	3	2	1	0
DEC value	32	16	8	4	2	1

Computation examples:

You would like to transfer the value of the 5th counter to the LATCH, i.e. you must set bit 4.

Bit pattern: **010000** Command: **VR6_LATCH(16)**

You would like to simultaneously transfer the values of the 3rd and 5th counter, i.e. you must set bits 2 and 4.

Bit pattern: **010100** Command: **VR6_LATCH(20)**

Application example:

```
EVENT:
VR6_LATCH(63)                'transfers simultaneously
                             'all counter values
```

VR6_READ

Syntax:

value = VR6_READ(*number*)

***ADwin* systems:**

Only for ***ADwin-(light)***-boards, which are equipped with the counter option ***ADwin-VR6***, and for ***ADwin-X-VR6***-Karten.

Description:

The command VR6_READ supplies the count rate defined by *number*. This command sets the latch before reading out the current value.

Note: The counters are numbered through from 1 to 6.

Application example:

```
EVENT:
value = VR6_READ(3)       'Supplies the value of
                          'Counter 3
```

VR6_READLATCH

Syntax:

value = VR6_READLATCH(*number*)

***ADwin* systems:**

Only for ***ADwin-(light)***-boards, which are equipped with the counter option ***ADwin-VR6***, and for ***ADwin-X-VR6***-Karten.

Description:

The command VR6_READLATCH reads the latch of the counter defined by *number*. The current value of the specified counter is *not* transferred to the latch before.

Notes: The counters are numbered through from 1 to 6.

Application example:

```
EVENT:
VR6_LATCH(63)             'Transfers all counter
                          'readings simultaneously
                          'to the latches
value = VR6_READLATCH(1) 'Supplies the value of the
                          'latch of counter 1
```

VR6_SETMODE

Syntax:

VR6_SETMODE(*selection*)

Only for ***ADwin-(light)***-boards, which are equipped with the counter option ***ADwin-VR6***, and **for *ADwin-X-VR6***-Karten.

Description:

The command VR6_SETMODE sets the counters to one of two modes. An unset bit (=0) corresponds to fourfold edge evaluation, i.e. two encoders supply signals phase-shifted by 90° which are evaluated to detect the direction. A set bit (=1) corresponds to one clock and one direction input per counter. For more details please refer to the documentation for your up/down counter.

The following table shows the relationship between the bit to be set and the counter number.

counter number	6	5	4	3	2	1
bit	5	4	3	2	1	0
DEC value	32	16	8	4	2	1

Computation example:

You would like to set the mode of the 5th counter to 1 and the modes of all other counters to 0: Set bit 4.

Bit pattern: **010000** Command: **VR6_SETMODE(16)**

Application example:

```
EVENT:
VR6_SETMODE(63)          'Sets all counters
                         'simultaneously to clock
                         'and direction signal
```

VR6_START

Syntax:

VR6_START(*selection*)

Only for ***ADwin-(light)***-boards, which are equipped with the counter option ***ADwin-VR6***, and for ***ADwin-X-VR6***-Karten.

Description:

The command VR6_START starts the up/down counter on the VR6 supplementary board defined by *selection*. A bit must be set in *selection* for each counter which is to be started. You can start a number of counters simultaneously. The following table shows the relationship between the bit to be set and the counter number:

counter numberr	6	5	4	3	2	1
bit	5	4	3	2	1	0
DEC value	32	16	8	4	2	1

Computation examples:

You would like to start counter 3: Set bit 2.

Bit pattern: **000100** Command: **VR6_START(4)**

You would like to simultaneously start counters 3 and 5: Set bits 2 and 4.

Bit pattern: **010100** Command: **VR6_START(20)**

Application example:

```
EVENT:
VR6_START(63)                'Starts all VR-counters
```

VR6_STOP

Syntax:

VR6_STOP(*selection*)

Only for ***ADwin-(light)-***boards, which are equipped with the counter option ***ADwin-VR6***, and for ***ADwin-X-VR6***-Karten.

Description:

The command VR6_STOP stops the up/down counter specified by *selection* on the VR6 supplementary board. A bit must be set in *selection* for each counter that is to be stopped. You can simultaneously stop a number of counters. The following table shows the relationship between the bit to be set and the counter number.

counter number	6	5	4	3	2	1
bit	5	4	3	2	1	0
DEC value	32	16	8	4	2	1

Computation example:

You would like to simultaneously stop counters 3 and 5: Set bits 2 and 4.
Bit pattern: **010100** Command: **VR6_STOP(20)**

Application example:

```
EVENT:
VR6_STOP(63)                    'Stops all VR6 counters
```

WAIT_EOC

Syntax:

`WAIT_EOC(`*`selection`*`)`

***ADwin* Systems:**

For ***ADwin-GOLD***, ***ADwin-(light)-***boards.

When using an ***ADwin-Pro*** system, you wil also find a command with the same name, but it has a DIFFERENT MEANING. Please, see the document "***ADwin-Pro*** - System Specifications - Programming in ***ADbasic***".

Description:

The command `WAIT_EOC` waits until the ADC defined in *`selection`* has finished converting. The two least significant bits (LSBs) of *`selection`* are used for selecting the converter. Bit 0 signifies ADC1, bit 1 ADC2.

Bit(s)	DEC value	*ADwin-Gold*	*ADwin board*	*ADwin-light*-board
0	1	ADC16-1	ADC1	ADC1
1	2	ADC16-2	ADC2	-
0,1	3	ADC16-1 ***and*** ADC16-2	ADC1 ***and*** ADC2	-
3	8	ADC12-1	-	-
4	16	ADC12-2	-	-
3,4	24	ADC12-1 ***and*** ADC12-2	-	-
0,3	9	ADC16-1 ***and*** ADC12-1	-	-
1,4	18	ADC16-2 ***and*** ADC12-2	-	-
0,1,3,4	27	***all*** ADCs	-	-

Note: Please consider that ADC1 and ADC2 can either be a 12 or 16 bit analog-digital converter. For further information see the hardware manual for your ***ADwin-(light)*** boards.

Application example:

```
DIM value AS INTEGER

EVENT:
SET_MUX(8)                'Set multiplexer for ADC2
                          'to channel 2 (input 4)

  Wait 4µs (ADwin-GOLD 6.5 µs) for the settling of
                          the multiplexer

START_CONV(2)             'Start A/D conversion ADC2
WAIT_EOC(2)               'Wait for end of
                          'conversion of ADC2
value = READADC(2)        'Read value
```

XOR

Syntax:

value3 = *value1* XOR *value2*

Description:

Links the bits of *value1* und *value2* using EXCLUSIVE-OR
The result of this operation can be seen in the following table:

If there is a bit in value1 =	and a bit in value2 =	result :
0	0	0
0	1	1
1	0	1
1	1	0

Note: Since it is a **bitwise operator**, it applys only for integer and long variables.

Application example:

```
DIM value3 AS LONG

value3 = 0100B XOR 0110B
                          'result: value3 = 0010B
```

8 How to solve problems

If problems already occur during installation, please refer to the documentation for your ***ADwin*** system. Make sure all settings have been carried out properly and completely. Please check then, referring to Chapter 2 (Software), if the software has been correctly installed. Also check if the base address, the processor type, etc. are set correctly in the menu **Options**. If your problems still persist, please give us a call.

If you need help of a more substantial nature, then please contact us directly.

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

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

W

X