ADbasic

Tutorial and Programming Examples

ADbasic Tutorial version 1.4, July 2003

Table of contents

1 Important Notes

Important! Before you start programming with ***ADbasic***, you must have completely installed your hardware and software. For this please have a look into the installation manual - "***ADwin*** Driver Installation".
(See also the file <Driver-Installation_eng.pdf> in the directory <C:\ADwin\Documents\Setup>).

The sample programs described on the following pages are an easy introduction into programming with ***ADbasic***, so that you will be able to find practicable solutions for your programming tasks in a relatively short time.

Please take into consideration that the complete performance of ***ADbasic*** can only be experienced when working with the program everyday and when using the ***ADbasic*** manual or the online help for further instructions.

Using sample programs

For nearly each example you find the program source code in the directory <C:\ADwin\ADbasic\Samples_ADwin> (standard installation).

For testing a sample program you need a properly working and connected ***ADwin*** system, ***ADbasic*** for editing the programs and the ***ADwin*** drivers which should be installed on your computer.

2 First Steps with *ADbasic*

We assume that you have already successfully installed your ***ADwin*** system with the help of the installation manual. And that you have assigned the right device number with the program *ADconfig* to the ***ADwin*** system (default setting: `150h`).

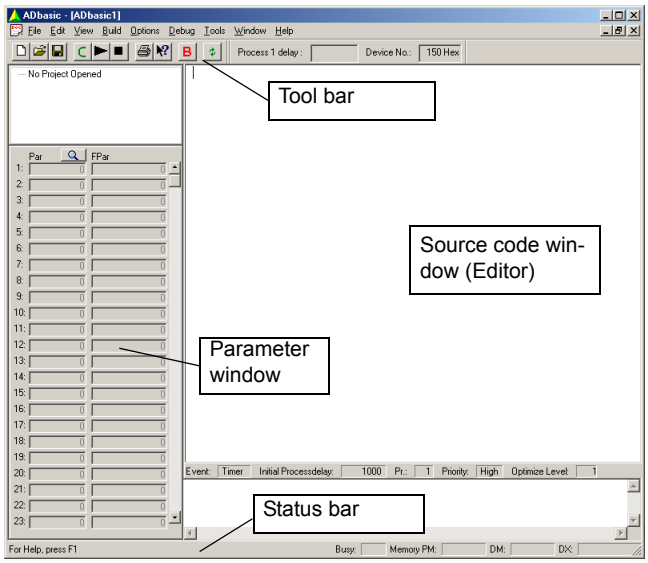
2.1 Checking the Communication

The first step should be to check once more the communication between the computer and your ***ADwin*** system. If you do not have made any changes after the installation, the right settings are automatically selected.

➢ Start the development environment ***ADbasic*** from the Windows start menu: `Programs` ▸ `ADwin` ▸ `ADbasic`.

You are seeing now the standard user interface of the development environment:

Some of *ADbasic*'s control elements



➢ Do a left mouse click on the icon B (Boot) in the tool bar (at the top of the window).
Thus you start the boot process that transfers the operating system to your ***ADwin*** system and starts it there.

With the transfer of the operating system (booting) you initialize the ***ADwin*** system at the same time. You set the system into a defined initial status, in which the memory is empty for receiving new programs. The boot processes must be repeated after each power-on of the ***ADwin*** system.

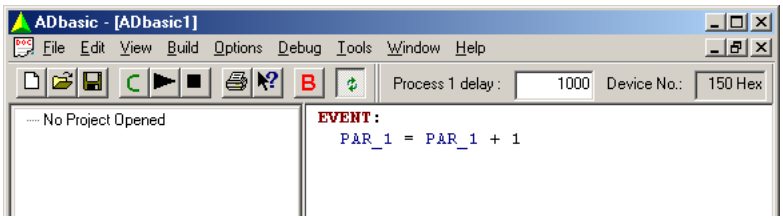
➢ Each of the following changes in the ***ADbasic*** development environment shows that the operating system has been successfully transferred:
- In the status line the message `ADwin is booted` appears.
- The arrays in the parameter window change to light-grey, that is they are active.
- In the tool bar the icon changes into the active status ("`Enable cyclic update`", at right, next to the boot icon B).
- When you have an ***ADwin-Gold*** system or a processor module of an ***ADwin-Pro*** system the LED starts blinking.

If the communication with your ***ADwin*** system is successful you will find the first programming example in the next chapter. Otherwise you will find information about error handling in the annex.

2.2 The First Program

The first program consists only of 2 lines; it periodically increments the global variable PAR_1.

➢ Type in the following program into the editor window:



Do not forget neither the colon behind **EVENT** nor the underscore in PAR_1.

➢ Do a left mouse click on the icon (Compile) in the tool bar.

With this instruction you compile your program, transfer it to the ***ADwin*** system and start it there.

➢ Watch the changes in the parameter window.

You see, that in the input field at the upper left no zero but a continuously incrementing number is indicated. This number is the actually current value of the global variable PAR_1.

The continuous incrementing of the variable PAR_1 is the consequence of the fact that the program is started in fixed time intervals, that means cyclically.

In this program, the line with the keyword **EVENT:** defines the start of a section, which is cyclically called, in this case a single program line

```
PAR_1 = PAR_1 + 1
```

which increments upon each call the value of the variable PAR_1 by 1.

The cycle time, in which the program line is called, can be directly entered in the input field Process 1 delay, in units of 25ns (= 1 clock cycle of the processor). If you enter a small value, processing – here: the incrementing – is executed faster, if you enter a higher value, it is executed more slowly. The value 40,000,000 for instance, causes the line to be executed every second.

Pay attention to the display Busy in the status bar (at the bottom of the window). It displays how much workload your process needs on the ***ADwin*** system. The workload should definitely remain under 100 %, if not, the system can become unstable.

The variable in this example has the name PAR_1. This characterizes a predefined, global variable for integer numbers, which is used for the bidirectional data exchange between simultaneously running ***ADbasic*** processes or between the computer and ***ADbasic*** processes. It is also called parameter. As a whole 80 of these variables (PAR_1 up to PAR_80) are available. You can use them anywhere in the program.

Local variables, arrays, and other data types are described later in the text.

Notes

3 A/D and D/A Conversion

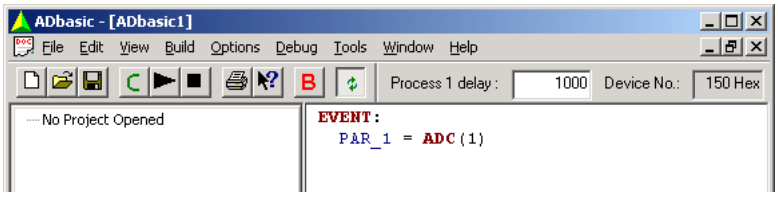
If you have an ***ADwin-Pro*** system continue with chapter 3.2.

3.1 *ADwin-Gold* and *ADwin-light-16* Systems

In order to convert an analog voltage with the A/D converter of the ***ADwin*** system into a digital bit sequence, we use the instruction **ADC** in this example; at the same time the converted value is written into the global variable PAR_1.

Read the analog voltage

➢ Enter and compile the following program:



You should now see a value in the parameter window in the line PAR_1, which varies around 32,768. This value corresponds to a measured voltage of 0 V, which is correct in this case, because the analog input 1 is open. The variations result from noise at the input (the voltage is measured in each process cycle again).

With the instruction **ADC** you measure voltages at the analog inputs between -10 V and +10 V. This corresponds to the digital values of 0 to 65,535 (see also Excursus).

Contrary to the first program you can see in the display Busy that this program needs essentially more processing time, although with each process cycle again only one program line is executed. The relatively long execution time of the instruction **ADC** results from the fact that you have to wait for the necessary conversion time for each individual measurement.

Output an analog signal

Now extend your program by additionally outputting an analog signal via D/A converter, and measure it (as described above) with the A/D converter:

➢ For this purpose connect the analog output 1 with the analog input 1 (+) as well as the analog ground (AGND) with the input 1 (–).
The pin assignment is described in the hardware manual of your ***ADwin*** system.

You call the digital-to-analog converter with the instruction **DAC**, to whom you must transfer the number of the analog output and the value you want to convert:

```
EVENT:
  DAC(1, PAR_2)
  PAR_1 = ADC(1)
```

After having transferred the program to the ***ADwin*** system, you can read the measured value again in the global variable PAR_1. It varies around the value 0, because the D/A converter outputs a value of -10 V, which is equivalent to the value 0 of the global variable PAR_2. You may prove with an external multimeter that exactly -10 V are output at the analog output 1.

You can easily change the value, which the D/A converter outputs at the analog output.

➢ Click PAR_2 in the parameter window and overwrite the displayed value.
After having confirmed with [RETURN] you see that the value in PAR_1 has changed, too. The display of a connected multimeter should also change.

Functions generator

This two-line program can easily be made to a functions generator, which for instance outputs a ramp function. For this we combine the first example (the incrementation) with the last example (AD-DA conversion):

```
EVENT:
  DAC(1, PAR_2)
  PAR_1 = ADC (1)
  PAR_2 = PAR_2 + 1
  IF (PAR_2 > 65535) THEN     ' 16-bit converter
    PAR_2 = 0
  ENDIF
```

If you compile this program you will see that the values of the global variables PAR_2 (and PAR_1, too) change.

➢ Connect an oscilloscope (a multimeter is too slow) to the analog output 1 and watch the ramp function, equivalent to the increasing value of PAR_2.

The query with **IF** at the end of the program is used to reset the counter values of the global variables PAR_2 to 0 after reaching the maximum value of 65,535. This is necessary, because the function **DAC** corrects the values exceeding the maximum value of 65,535 (at 16-bit converters).

Excursus

Analog conversion with different converter resolutions

The input converters (ADCs) of the ***ADwin*** system are available as 12-bit, 14-bit or 16-bit converters, the output converters (DACs) only as 16-bit converters.

A 16-bit ADC converts the voltage into a value between 0 and 65,535 digits.
If the converter has a resolution of 12-bit or 14-bit, the voltage is converted using accordingly greater digit steps. In order to be able to compare the values with those of a 16-bit ADC, 2 zero-bits are added from the right to the 14-bit value and 4 zero-bits are added to the 12-bit value.

The result is that the measurement values of different converter resolutions are returned in different step sizes. The maximum measurement value (measurement range -10V…+10V) is just one step size smaller than 2^{16} digits:

	Step size	Max. measurement value
16-bit converter	1 digit	65,535 digits
14-bit converter	4 digits	65,532 digits
12-bit converter	16 digits	65,520 digits

3.2 *ADwin-Pro* Systems

Special characteristics of the Pro system

The flexible equipment of the ***ADwin Pro*** systems requires that you include at the beginning of each program certain files with the instruction **#INCLUDE**. If you forget to do so, the compiler sends an error message. Moreover, you must indicate at all hardware related instructions the address of the Pro module.

Include-file

Each of these include-files contains instructions so that you can access the hardware of a specified class of Pro modules. Which of the files you must include depends therefore on the instructions (and modules) you are using.

These include-files must always be included at the beginning of your source code.

You will find an assignment of the instructions to individual include-files in the documentation ***"ADwin Pro*** System Specification; Programming in ***ADbasic***". In the following example for the instruction **ADC** it its necessary to include the file ADWPAD.INC, and for the instruction **DAC** the file ADWPDA.INC.

Always indicate the complete path to the include-files or make sure that they can be found in the standard directory. In this example the files are supposed to be in C:\ADwin\ADbasic\Inc\.
You can adapt the standard directory with the help of the menu item Options/Settings; then set in the dialog window Settings, under Directory the option Include-Directory (with a "\" as last char).

Module address

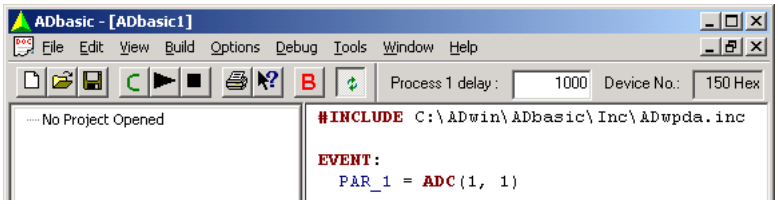
In order to access a certain Pro module always indicate the module address as first argument of an instruction. Therefore all instructions for *Pro* systems have always one argument more than the same instructions for the *Gold* or *light-16* systems. In the Pro hardware manual you will find information about how to set the module address.
In this example we use an A/D module Pro-AIn-x (with multiplexer) with the address 1 and D/A module Pro-AOut-x with the address 1.

Beginning of the example

In order to convert an analog voltage with the A/D converter of the ***ADwin*** system into a digital bit sequence, we use the instruction **ADC** in this example, which writes the converted value into the global variable PAR_1.

Read analog voltage

➢ Enter and compile the following program:



You should now see a value in the parameter window in the line PAR_1, which varies around 32,768. This value corresponds to a measured voltage of 0V, which is correct in this case, because the analog input 1 is open. The variations result from noise at the input (the voltage is measured in each process cycle again).

With the instruction **ADC** you measure voltages at the analog inputs between -10V and +10V. This corresponds to the digital values of 0 to 65,535 (see also "Excursus").

Contrary to the first program you can see in the display Busy that this program needs essentially more processing time, although with each process cycle again only one program line is executed. The relatively long execution time of the instruction **ADC** results from the fact that you have to wait for the necessary conversion time for each individual measurement.

Output an analog signal

Now extend your program by additionally outputting an analog signal via D/A converter, and measure it (as described earlier) with the A/D converter:

➢ For this purpose connect the analog output 1 with the analog input 1 (+) as well as the analog ground (AGND) with with the input 1 (–).
The pin assignment is described in the hardware manual of your ***ADwin Pro*** system.

You call the digital-to-analog converter with the instruction **DAC**, to whom you must transfer the module's number, the number of the analog output and the value you want to convert:

```
#INCLUDE ADwpad.inc
#INCLUDE ADwpda.inc
```

```
EVENT:
  DAC(1, 1, PAR_2)
  PAR_1 = ADC(1, 1)
```

After having transferred the program to the ***ADwin*** system, you can read the measured value again in the global variable PAR_1. It varies around the value 0, because the D/A converter outputs a value of -10 V, which is equivalent to the value 0 of the global variable PAR_2. You may prove with an external multimeter that exactly -10V are output at the analog output 1.

You easily change the value, which the D/A converter outputs at the analog output.

➢ Click PAR_2 in the parameter window and overwrite the displayed value. After having confirmed with [RETURN] you see that the value in PAR_1 has changed, too. The display of a connected multimeter should also change.

Functions generator

This two-line program can easily be made to a functions generator, which for instance outputs a ramp function. For this we combine the first example (the incrementation) with the last example (AD-DA conversion):

```
#INCLUDE ADwpad.inc
#INCLUDE ADwpda.inc

EVENT:
  DAC(1, 1, PAR_2)
  PAR_1 = ADC(1, 1)
  PAR_2 = PAR_2 + 1
  IF (PAR_2 > 65535) THEN     ' 16-bit converter
    PAR_2 = 0
  ENDIF
```

If you compile this program you will see that the values of the global variables PAR_2 (and PAR_1, too) change.

➢ Connect an oscilloscope (a multimeter is too slow) the analog output 1 and watch the ramp function, equivalent to the increasing value of PAR_2.

The query with **IF** at the end of the program is used to reset the counter values of the global variables PAR_2 to 0 after reaching the maximum value of 65,535. This is necessary, because the function **DAC** corrects the values exceeding the maximum value of 65,535 (at 16-bit converters). See also the excursus on page 5.

4 Saving of Measurement Values

Large quantity of data in arrays

If you work with consecutive measurement values, for instance in a series of measurements, you can write them into an array. For this you define an array. There are global and local arrays (identical to variables); here the global array DATA_1 is used.

```
DIM DATA_1[1000] AS LONG
DIM i AS LONG

INIT:
  i = 1

EVENT:
  DATA_1[i] = ADC (1)
  i = i + 1
  IF (i > 1000) THEN
    i = 1
  ENDIF
```

First of all, in this example the global array DATA_1 is dimensioned with 1,000 array elements. These array elements are defined to store values of the data type **LONG**, that is integer values with 32-bit. Then you define the local variable i, which is to be used as index variable for accessing the data element.

In the section **INIT:** the value 1 is assigned to the variable i. This value is the index of the first data element in the global array. Consider that there is no array element with the index 0! The section **INIT:** is only called once during processing the program, directly after the start.

You find the essential program lines after the keyword **EVENT:**. Here the value of the analog channel 1 is read in first, and at the same time assigned to the element of the global array with the index i. Now the value of the index variable is incremented by 1. Afterwards, in the **IF** query the index variable is reset to its start value as soon as it has reached the value 1,000, in order to avoid an overflow of the array. Thus, always the last 1,000 measurement values are stored in the global array.

Display array values

If you compile and start the program, you will not find any changes in the parameter window. But this is not really astonishing, because none of the global variables (PAR_1 ... PAR_80 or FPAR_1 ... FPAR_80) is used in the program. Nevertheless, you can display the contents of the array with the add-in program TGraph of ***ADtools***.
ADtools is a collection of useful Windows programs, accessing directly the global variables and arrays (PAR, FPAR and DATA) of the ***ADwin*** system, in order to display and change them.

➢ You open the program TGraph in the start menu under Programs ▸ ADwin ▸ ADtools ▸ TGraph.

In this help program the values of the array `DATA_1` are graphically displayed. In the picture you see the read-in values of the AD converter at open input.

If you start `TGraph` for the first time, you should in any case have a closer look to the options, being called via the menu item `Options`. It is important to indicate the corresponding Device No. and the number of the specified data array (`DATA_1` in your program has the number 1, `DATA_10` the number 10).
The value range of the display can be adapted with the different buttons, but for the first test the automatic scaling (option `Autoscale` = `Yes`) will do.

5 Online Evaluation of Measurement Values

In this chapter we will show you how you can read-in measurement values in a program and process them fastly. You need an ***ADwin*** system with an analog input and an external, analog signal in the range of -10V…+10V.

➢ Connect the external analog signal with the analog input 1 (+) as well as the analog ground (AGND) with the input 1 (-).

For the pin assignment see your hardware manual of the ***ADwin*** system. Please consider the information about the measurement range and earthing of your device.

The program reads in 1,000 measurement values at the analog input 1 and determines the highest and lowest value. The minimum value is written in the variable `PAR_1`, the maximum value in `PAR_2`.

From here you need no longer type in the whole program examples. They can be found in the directory `C:\ADwin\ADbasic\samples_ADwin`. If you work with an ***ADwin-Pro*** system, pay attention to the notes in chapter 3.2 on page 5.

The first example is saved under the name `BAS_DMO1.BAS`. After compiling and starting of the program the minimum value in variable `PAR_1` and the maximum value in the variable `PAR_2` is shown.
Instead of calculating the minimum and maximum values you can also easily execute other calculations.

BAS_DMO1.BAS

```
'########################################################
'# The process BAS_DMO1 is searching for the maximum
'# and minimum values out of 1000 samples from ADC-#1
'# and writes the result to PAR_1 (min.) and PAR_2 (max).
'# After 1000 samples a flag (= PAR_10) is set.
'# PAR_1 = minimum value
'# PAR_2 = maximum value
'# Platform: ADwin-Gold or ADwin-light-16
'########################################################

#DEFINE limit 65535          'max. 16 bit ADC-value

DIM i1, iw, max, min AS LONG

INIT:
  i1 = 1                     'reset sample counter
  max = 0                    'initial maximum value
  min = limit                'initial minimum value
  PAR_10 = 0                 'init End-Flag
  GLOBALDELAY = 40000        'cycle-time of 1ms (ADSP)

EVENT:
  iw = ADC(1)                'get sample
  IF (iw > max) THEN max = iw 'new maximum sample?
  IF (iw < min) THEN min = iw 'new minimum sample?
  i1 = i1 + 1                'increment index
  IF (i1 > 1000) THEN        '1000 samples done?
    i1 = 1                   'reset index
    PAR_1 = min              'write minimum value
    PAR_2 = max              'write maximum value
    min = limit              'reset minimum value
    max = 0                  'reset maximum value
    ACTIVATE_PC              'only for use with TestPoint
    PAR_10 = 1               'set End-Flag
  ENDIF
```

6 Digital Proportional Controller

For the following example, a digital proportional controller, you need an ***ADwin*** system with an analog input and an analog output as well as an external system that is to be controlled.

With the global variable PAR_1 you set the controller's setpoint and with PAR_2 the gain. The actual value of the system you want to control is measured at the A/D converter input 1. The program calculates the actuating value and outputs this value at the D/A converter output 1. The gain must be adapted to the system that you want to control.

The external system to be controlled can for instance be a simple low-pass. In order to get a functioning controller, the signal "actual value" of the system, which is being controlled, must be connected with the analog input 1(+) as well as the analog ground (AGND) with the input 1(-). And a connection of the analog output to the actuating input of the controlled system must be established.

For the pin assignment see the hardware manual of your ***ADwin*** system. Please pay attention to the information about the measurement range and earthing of your device.

You will find the source code of the controller under the name BAS_DMO2.BAS (in the directory C:\ADwin\ADbasic\samples_ADwin). After you have compiled and started the program, you see in the parameter window the set-point to the right of 1: and the gain to the right of 2:. By changing the value of PAR_1 you modify the set-point and by changing the value of PAR_2 you modify the gain factor of the controller.

BAS_DMO2.BAS

```
'#################################################
'# The process BAS_DMO2 is a digital P-controller.
'# PAR_1 = setpoint
'# PAR_2 = gain
'# Platform: ADwin-Gold or ADwin-light-16
'#################################################

#DEFINE offset 32768          '0V for 16 bit ADC/DAC-systems

DIM cd, av AS LONG

INIT:
  PAR_1 = offset              'initial setpoint
  PAR_2 = 10                  'initial gain
  GLOBALDELAY = 40000         'cycle-time of 1ms (ADSP)

EVENT:
  cd = PAR_1 - ADC(1)         'compute control deviation (cd)
  av = cd * PAR_2 + offset    'compute actuating value (av)
  DAC(1, av)                  'output actuating value on DAC#1
```

7 Data Exchange with a Global Array

If you just want to exchange large quantities of data, the best solution is to use global arrays. For this example you need an ***ADwin*** system with an analog input and an external, analog signal in the range of -10V … +10V. Connect the system with the analog input 1(+). In addition you have to setup a connection between the analog ground (AGND) and the input 1(-). For the pin assignment see the hardware manual of your ***ADwin*** system. Please consider the information about the measurement range of the system.

The program measures the analog input 1 and informs the computer after 1,000 measurements. The computer is then ready to read these data. They are transferred with the help of a global array. The ***ADbasic*** program finishes after 1,000 measurements independently.

The source code is saved under the name `BAS_DMO3.BAS`. The program includes an instruction for the communication with TestPoint. If you have not installed TestPoint on your computer, delete the line from the example program.

In order to test the ***ADbasic*** program, you can read out the data of the global array `DATA_1` with ghe program `TGraph` and display them on the computer.

`BAS_DMO3.BAS`

```
'#########################################################
'# The process BAS_DMO3 samples ADC-#1 1000 times,
'# then stops and informs the PC to fetch the samples.
'# Data is transferred using a standard DATA array.
'# DATA_1 = series of samples
'# PAR_10 = End-flag
'# Platform: ADwin-Gold or ADwin-light-16
'#########################################################

DIM DATA_1[1000] AS LONG
DIM index AS LONG

INIT:
  index = 0                     'reset array pointer
  PAR_10 = 0                    'init End-flag
  GLOBALDELAY = 40000           'cycle-time of 1ms (ADSP)

EVENT:
  index = index + 1             'increment array pointer
  IF (index > 1000) THEN        '1000 samples done?
    ACTIVATE_PC                 'only for use with TestPoint
    PAR_10 = 1                  'set End-Flag
    END                         'terminate process
  ENDIF
  DATA_1[index] = ADC(1)        'acquire sample and save in array
```

8 Continuous Data Transfer with FIFO

If you want to continuously transfer large quantities of data, you may use the data structure FIFO. You need an ***ADwin*** system with an analog input and an external, analog signal in the range of -10V…+10V.

The data structure FIFO is a global array, where data are easily and automatically managed. With a FIFO you make sure that the data being written into the array first will also be read out first: First In, First Out = FIFO.

The program continuously measures the analog input 1. An application program on the computer can continuously read the data from the FIFO array. But here you have to make sure that you read the data faster than they are written into the FIFO by the ***ADwin*** system. If this is not the case the FIFO will "overflow" and data will get lost.

You have to connect the external signal with the analog input 1(+) as well as the analog ground (AGND) with the input 1(-).

For the pin assignment see the hardware manual of your ***ADwin*** system. Please pay attention to the information about the measurement range and earthing of your device.

You find the source code under the name `BAS_DMO4.BAS`. The program includes an instruction for the communication with TestPoint. If you have not installed TestPoint on your computer, delete the line from the example program.

Before compiling start the help program `TFifo` with ***ADtools***; the program now waits that data are written in the FIFO array. After compilation and transfer of the ***ADbasic*** program `TFifo` reads out the data and stores them on the hard disk for further processing. How to use `TFifo` is described in the integrated online help.

BAS_DMO4.BAS

```
'#######################################################
'# The process BAS_DMO4 continuously samples ADC-#1.
'# After 1000 samples the PC will be informed to
'# fetch the samples.
'# Data is transferred using a FIFO array.
'# DATA_1 = series of samples
'# Platform: ADwin-Gold or ADwin-light-16
'#######################################################

DIM DATA_1[4000] AS LONG AS FIFO
DIM index AS LONG

INIT:
  index = 0                     'reset sample counter
  PAR_10 = 0                    'init End-flag
  GLOBALDELAY = 40000           'cycle-time of 1ms (ADSP)

EVENT:
  index = index + 1             'increment sample counter
  IF (index > 1000) THEN        '1000 samples done?
    ACTIVATE_PC                 'only for use with TestPoint
    PAR_10 = 1                  'set End-flag
    index = 0                   'reset sample counter
  ENDIF
  DATA_1 = ADC(1)               'acquire sample and save in FIFO
```

9 Digital PID Controller

For this program, a PID controller, you need an ***ADwin*** system with an analog input and an analog output, as well as an external system that is to be controlled.

You configure the PID controller with several variables. At the A/D converter input 1 the actual value of the system you are controlling is measured. The program calculates the actuating value and outputs it at the D/A converter output 1. You have to adapt the gain to the system you are controlling.

In order to test the system you must connect the external signal with the analog input 1(+) as well as the analog ground (AGND) with the input 1(-).

For the pin assignment see the hardware manual of your ***ADwin*** system. Please pay attention to the information about the measurement range and earthing of your device.

You find the source code under the name `BAS_DMO6.BAS`. The program includes an instruction for the communication with TestPoint. If you have not installed TestPoint on your computer, delete the line from the example program.

After having transferred the program to your ***ADwin*** system you can change the controller settings with the help of the global variables in the parameter window.

The calculated control deviation is continuously written into the array `DATA_1`. By visualizing these data with the program `Tgraph` you can optimize the control parameters.

BAS_DMO6.BAS

```
'#######################################################
'# The process BAS_DMO6 is a digital PID-controller.
'# The controller operates with float variables whose
'# coefficients have to be computed on the PC and then
'# transferred to the ADwin-system.
'# PAR_1  = setpoint in digits
'# FPAR_2 = P (gain, proportional factor)
'# FPAR_3 = I (integration time)
'# FPAR_4 = D (derivative time)
'# PAR_5  = buffer index for the control deviation
'# PAR_6  = cycle-time in 25ns steps
'# PAR_9  = flag for new setpoint definition by PC
'# Attention: Prior to start the program the controller
'#            parameters have to be set to the correct value!
'#            Make sure that FPAR_3 is not zero!!!
'# Platform: ADwin-Gold or ADwin-light-16
'#######################################################

#DEFINE offset 32768          '0V output

DIM DATA_1[4000] AS LONG
DIM av, cd, cdo, sum AS LONG
DIM diff AS FLOAT

INIT:
  sum = 0                     'initial value of integral part
  cd = ADC(1)                 'initial value of control
                              'deviation (cd) & MUX to Ch-#1
  PAR_5 = 1                   'reset array index
  IF (FPAR_3 < 1E3) THEN FPAR_3 = 1E3 'check min. of
                              'integration time
  IF (PAR_6 < 4E4) THEN PAR_6 = 4E4 'allow cycle-times >=1ms
  GLOBALDELAY = PAR_6         'set cycle-time

EVENT:
                              'compute actuating value
  av = FPAR_2*(cd + sum/FPAR_3 + diff*FPAR_4)
  START_CONV(1)               'start conversion ADC-#1
  DAC(1, av + offset)         'output actuating value at DAC#1
  cdo = cd                    'keep control deviation in mind
  WAIT_EOC(1)                 'wait for ADC's end of conversion
  cd = PAR_1 - READADC(1)     'compute control deviation
  FPAR_9 = FPAR_9*0.99+cd*0.01'mean value of control deviation
  sum = sum + cd              'calculate integral
  IF (sum > 2E6) THEN sum = 2E6 'positive limit of integral
  IF (sum < -2E6) THEN sum = -2E6 'negative limit of integral
  diff = (cd - cdo)           'calculate deviation difference
  DATA_1[PAR_5] = cd          'write control deviation
                              'in a buffer
  INC PAR_5                   'increment buffer index
  IF (PAR_5 >= 4000) THEN     '4000 samples done?
    ACTIVATE_PC               'only for use with TestPoint
    PAR_10 = 1                'set End-Flag
    PAR_5 = 1                 'reset buffer index
  ENDIF

FINISH:
  DAC(1,offset)               'analog output #1 to 0V
```

Help for Error Handling

In this chapter we will inform you about how to handle the errors described below:

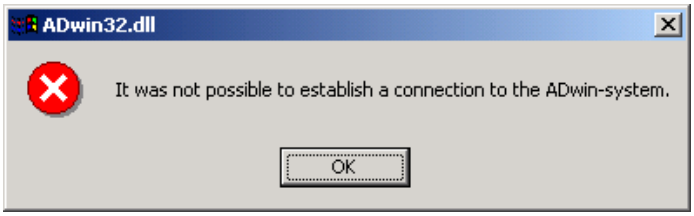
– Errors Occurring during the Boot Process

– The Communication is Interrupted

A-1 Errors Occurring during the Boot Process

Error message

If the ***ADwin*** operating system has not been successfully loaded to your system, you get (one or) two errors messages. Confirm both messages with `OK`, in order to being able to continue working with ***ADbasic***.



This error message displays that the ***ADwin*** system does not respond to any communication setups.

This error message describes more precisely which kind of error occurs. The message depends on the kind of communication, just being used between the computer and the ***ADwin*** system; here you see the message of an Ethernet network.

Remedy

Solve this problem by checking the following items and setup the communication to the ***ADwin*** system again:

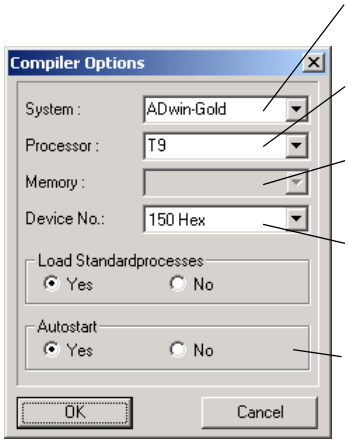
Mechanical Check-Up

Check-up	Remedy
Is the system connected to the power supply and switched on?	Make the necessary connections from the system to the power supply and switch it on.
Is the data line between system and computer connected correctly?	Either connect the system via USB cable to the computer or connect it to the Ethernet network.

Device No.

Check-up	Remedy
Has the correct Device No. been set in *ADconfig*? For this, call the program in the Windows start menu.	Set in *ADconfig* the relevant Device No. (Standard `150h`).

Configuration in *ADbasic*

Check-up	Remedy
Is the system configured correctly in ***ADbasic***? Call the menu item `Options` ▸ `Compiler` in ***ADbasic***. Check here, if the settings correspond to those in your system (for more details, see below).	Set the relevant data in the dialog box. Please consider that the option `Autostart` is set to `Yes`.
	Your ***ADwin*** system. The processor type of your ***ADwin*** system: T9 or T10. The memory size is of no importance for the processors T9 and T10. The device no. of your ***ADwin*** system, you have set with *ADconfig*. This option should be set to `Yes`.
Is the correct path indicated in ***ADbasic*** for the operating system file? To check this, select the menu item `Options` ▸ `Setting` and there `Directory`.	Enter the path name in `BTL-Directory` (with a "\" at the end). Upon a standard installation you must find here the path `C:\ADwin\`.

If this error handling is not successful, then probably an error has occured upon installation. Try to verify once more the installation in the manual ***"ADwin*** Driver Installation".

If nevertheless the error cannot be avoided, please call our support: Phone +49 (6251) 96320.

A-2 The Communication is Interrupted

Recognize the error

It may happen that the communication between the ***ADwin*** system and the computer is interrupted. This is the case when

- the color of input fields changes to dark-grey: in the parameter window, in the process window as well as the input field `Process n delay`,
- the values in the parameter and process windows do not change any more,
- no values are displayed in the status line,
- the button `Enable cyclic update` in the tool bar is deactivated  .

An interrupted communication frequently means that the processor on the ***ADwin*** system doesn't have enough time to communicate with the computer. This doesn't mean that your program doesn't run on the system, but that the computer has interrupted the communication.

Setup the communication again

You can only solve this problem only by rebooting your ***ADwin*** system. But this process also deletes your program in the system memory; you have to recompile and restart it (button).

Remedy

But you have not definitely avoided the error by booting and compiling. First of all try to remove the error cause and then go on with your program.

There are many reasons for an interruption of the communication; in most cases the processor has reached its maximum workload due to a high-priority process, so that it cannot respond in time to communication requests of the computer. Below you will find some hints for debugging.

Check-up	Remedy
Is there a problem with the data line? If meanwhile the connection functions again, the communication can be set up again.	Click `Enable cyclic update` in the tool bar. If this does not have any effect, boot your system.
Does the high-priority process start with a too short processdelay? Call the menu item `Options` ▸ `Process` in ***ADbasic*** and check the setting `Initial Processdelay`.	Set a higher value for the `Initial Processdelay` and restart the program.
Have you accessed array elements in the program which are not in the declared range?	Correct your program and restart it.
Is the execution time of the program section **`EVENT:`** always (!) shorter than the processdelay?	Increase the processdelay or Move program parts, which are not time-critical to a low-priority process.
Are several high-priority processes active, which influence each other?	Adjust the execution times and processdelays of the process with each other.