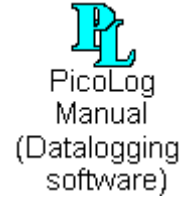


ADC10/12/40/42 Manual v1.0

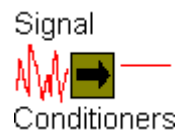
- 1. Introduction
- 2. Connecting to PC
- 3. Specifications
- 4. Technical Information



Writing your own software overview



Other Information



Safety Warning

The ADC-10 and ADC-12 are designed to measure voltages in the range 0 to 5V. Any voltages in excess of $\pm 30V$ may cause permanent damage to the unit.

The ADC-40 and ADC-42 are designed to measure voltages in the range -5V to 5V. Any voltages in excess of $\pm 30V$ may cause permanent damage to the unit.

The inputs to the ADC are not designed for use with mains voltage. To measure mains we recommend the use of an differential isolating probe specifically designed for such measurements.

The ground input of the ADC BNC outer shell) is connected directly to the ground of your computer (which for most computers is mains earth). This is done in order to minimise interference. As with most oscilloscopes, you should take care not to connect the ground input of the ADC to anything which may be at some voltage other than ground: doing so may cause damage to the ADC. If in doubt, use a meter to check that there is no significant AC or DC voltage.

For computers that do not have an earth connection (for example laptops), it must be assumed that the ADC is not protected by an earth. For such computers, we recommend that only class II (double insulated) oscilloscope probes should be used.

The unit contains no user serviceable parts: repair or calibration of the unit requires specialised test equipment and must be performed by Pico Technology Limited or their authorised distributors.

Introduction

The PICO ADC-10, ADC-12, ADC-40 and ADC-42 are medium speed, single channel, analog to digital converters.

All of these products can be used as virtual instruments (oscilloscope / spectrum analyser) with the PicoScope program, or as a data logger using PicoLog. There are also stand-alone programs to collect large blocks of data under DOS or Windows. Alternatively, you can use the driver software to develop your own programs to collect and analyse data from the unit (the same driver works for all units).

There are three differences between these products:

- the ADC10 and 40 are 8-bit converters, and the ADC-12 and 42 are 12-bit converters
- the ADC-10 and 12 accept 0 to +5 volts, whereas the ADC-40 and 42 accept -5 to +5 volts
- the ADC-40 and 42 have an input impedance of 1M Ohm, and so are suitable for use with x10 scope probes: the ADC-10 and ADC-12 have a lower input impedance, and will give misleading results if used with a x10 probe.

This manual describes the physical and electrical properties of the ADC-10/12/40/42, and explains how to use the DOS stand-alone program and the software drivers. Please refer to the documents listed below for further information about other items.

DOS drivers	adc10.txt file
PicoLog for DOS	PL.TXT file
PicoLog for Windows	Online help file
PicoScope for Windows	Online help file

Connecting to the PC

To use the ADC, you should connect it to the printer port on your computer, either directly or using a good quality extension cable. Next, connect a voltage source to the BNC connector. The ADC has the same connectors as an oscilloscope, so you can use standard oscilloscope probes.

To check that the unit is working, start up the PicoScope program. If you have connected the ADC to a printer port other than the port specified when you installed the software, you will need to go to the Setup panel and then change the port number to the appropriate value. PicoScope should now display the voltage that you have connected. If you are using scope probes, when you touch the scope probe tip with your finger, you should see a small 50Hz mains signal on the screen.

Specification

		ADC-10	ADC-12	ADC-40	ADC-42
Resolution	bits	8	12	8	12
Number of input channels		1			
Input range	V	0 to 5V		-5V to 5V	
Maximum sampling rate	ksps	20	15	20	15
Repeatability		±4lsb at 25 C			
Absolute accuracy		±1% typical at 25 C			
Overvoltage protection	V	±30			
Input impedance	ohms	200k		1M	
Input connectors		BNC			
Output connector		25 way male D-type to computer printer port			
Power requirements		No power supply required			
Environmental conditions		0 to 70 C 0 to 95% humidity NOT water resistant			

Scaling

The ADC-10 is an 8-bit unipolar analog to digital converter. This means that it produces values in the range 0 to 255 to represent voltages between 0 and 5 volts. To convert from ADC readings to Volts, you should multiply by 5 and divide by 255. Thus, an ADC reading of 132 represents $132 \times 5 / 255 = 2.588$ Volts.

The ADC-12 is a 12-bit unipolar analog to digital converter. This means that it produces values in the range 0 to 4095 to represent voltages between 0 and 5 volts. To convert from ADC readings to Volts, you should multiply by 5 and divide by 4095. Thus, an ADC reading of 132 represents $132 \times 5 / 4095 = 0.161$ Volts.

The ADC-40 is an 8-bit bipolar analog to digital converter. This means that it produces values in the range 0 to 255 to represent voltages between -5 and 5 volts. To convert from ADC readings to Volts, you should subtract 128, multiply by 5 and divide by 128. Thus, an ADC reading of 132 represents $(132-128) \times 5 / 128 = 0.156$ Volts.

The ADC-42 is a 12-bit bipolar analog to digital converter. This means that it produces values in the range 0 to 4095 to represent voltages between -5 and 5 volts. To convert from ADC readings to Volts, you should subtract 2048, multiply by 5 and divide by 2048. Thus, an ADC reading of 132 represents $(132-2048) \times 5 / 2048 = -4.677$ Volts.

DOS stand-alone program

The program ADC10.exe read in a block of data values from the ADC-10 and ADC-40 and writes them to a file. The program ADC12.exe does the same for the ADC-12 and ADC-42. The readings are recorded in ADC counts: see [Scaling](#) for conversion to volts.

To run the program, type in

```
adc10 [-options] filename
```

where options are as listed below and filename is the name of the file to write the data to.

```
adc10 @control
```

where control is the name of a control file containing a number of options, one per line.

The following table defines the options:

Option	Example	Meaning
-p	.-p1	Get data from LPT1
-i	-i100	wait 100us between each reading
-n	-n1000	read in 1000 values
-t	-tn -tm -tr100	no trigger manual trigger (press a key to start) trigger on value rising above 100
-b	-b	binary file (ie not text)

Direct control of ADC

The following example files show how to drive the converters directly:

ADC10/40	ADC12/42
adc10a.c	adc12a.c
adc10a.pas	adc12a.pas
adc10.bas	adc12.bas

Drivers

The ADC-10, 12, 40 and 42 are supplied with driver routines that you can build into your own programs. Drivers are provided for the following operating systems:

DOS
Windows 3.x
Windows 95/98
Windows NT/2000

Once you have installed the software, the DRIVERS sub-directory contains the drivers and a selection of examples of how to use the drivers. It also contains a copy of this help file in text format.

The driver routine is supplied as object files for DOS, and as Dynamic Link Libraries for Windows 3.1, 95/98 and NT/2000

Note that there are a number of differences between the DOS driver and the Windows drivers. See adc10.txt and adc12.txt for information about the DOS driver.

The Windows DLLs can be used with C, Delphi and Visual Basic programs: they can also be used with programs like Microsoft Excel, where the macro language is a form of Visual Basic.

The driver is capable of supporting up to three units (one each on LPT1, LPT2 and LPT3). The units can be any mixture of ADC-10, ADC-12, ADC-40 and ADC-42.

The following table specifies the function of each of the routines in the Windows drivers:

Routine	Function
adc10_get_driver_version	Check that this is the correct driver
adc10_open_unit	Open the driver to use a specified printer port
adc10_set_unit	Select which ADC-10 unit to use
adc10_close_unit	Close the specified printer port
adc10_get_value	Get a single reading
adc10_get_value_and_time	Get a single reading and the time
adc10_set_trigger	Set a trigger event
adc10_set_interval	Set the time interval for the next call to adc10_get_values, or adc10_get_times_and_values
adc10_get_values	Get a block of readings at fixed intervals
adc10_get_times_and_values	Get a block of readings and their times, at fixed intervals
adc10_get_unit_info	Get information about an ADC10 unit

The driver offers the following facilities:

- specify the printer port that is connected to the ADC
- take a single reading
- specify a trigger event (only available in block mode)
- collect a block of samples at fixed time intervals

You can specify a sampling interval from 50us to a second. If you specify an interval that is shorter than your computer can manage, the driver will tell you how long it will actually take to collect the specified number of samples. After allowing for this, the timing accuracy under DOS is better than 1% for a block of 1000 samples at all sampling rates.

Under Windows, the sampling may be affected by Windows activities. At the least, there will be gaps in the data every 55 milliseconds due to the Windows timer function. There will be additional gaps if you move the mouse, or have other programs running. We therefore recommend using the `adc10_get_values_and_times` routine, so that you can determine the exact time that each reading was taken.

The normal calling sequence to collect a block of data is as follows:

- Check that the driver version is correct
- Open the driver
- Set trigger mode (if required)
- Set sampling mode (channels and time per sample)

- While you want to take measurements,
 - Get a block of data
- End While
- Close the driver

adc10_get_driver_version

```
PREF1 short PREF2 adc10_get_driver_version (void);
```

This routine returns the version number of the ADC-10 driver. You can use it to check that your application is used only with the driver version that it was designed for use with.

Generally speaking, new driver versions will be fully backward compatible with earlier versions, though the converse is not always true, so it should be safe to check that the driver version is greater than or equal to the version that it was designed for use with.

The version is a two-byte value, of which the upper byte is the major version and the lower byte is the minor version.

adc10_open_unit

```
PREF1 short PREF2 adc10_open_unit (short port, short product);
```

This routine opens the ADC-10 driver.

For DOS and the 16-bit Windows driver, it checks the BIOS printer address table and gets the address of the specified printer port. This is not possible in the Windows 32-bit driver, so it assumes that the printer ports 1..3 are at 0x378, 0x278 and 0x3BC.

It then calibrates the timing functions for the computer. It returns `TRUE` if successful. If it is not successful, you can call `adc10_get_unit_info` to find out why it failed.

port	The number of the parallel port that the ADC-10 is connected to (1 for LPT1, 2 for LPT2 etc).
Product	This identifies the type of product that you wish to use.
	10 - adc10
	12 - adc12
	40 - adc40
	42 - adc42

adc10_close_unit

```
PREF 1 short PREF2 adc10_close_unit (short port);
```

This routine closes the ADC-10 driver.

port The number of the parallel port

adc10_set_unit

```
PREF1 short PREF2 adc10_set_unit (short port);
```

This routine is used to select the unit to use for subsequent operations. It is only necessary to use this function if you wish to have more than one unit open at the same time.

adc10_get_value

```
PREF 1 short PREF2 adc10_get_value (void);
```

This routine reads the current value of one channel. Depending on your computer, it will take approx 100µs to take one reading.

See also [adc10_get_value_and_time](#), which reports the exact time at which the reading was taken

adc10_get_value_and_time

```
PREF 1 void PREF2 adc10_get_value_and_time (  
    unsigned long * sample_time,  
    short * value);
```

This routine reads the current value from the currently selected adc10, and the time in microticks at which the reading was taken. Depending on your computer, it will take approx 100µs to take one reading.

sample_time is the time in microticks for the reading. There are 2^{32} microticks per hour or 1,193,046 per second. The sample time will therefore wrap around once an hour.

Value is scaled in [ADC counts](#).

adc10_set_trigger

```
PREF1 void PREF2 adc10_set_trigger ( unsigned short enabled,  
                                   unsigned short auto_trigger,  
                                   unsigned short auto_ms,  
                                   unsigned short dir,  
                                   unsigned short threshold,  
                                   unsigned short delay);
```

This routine defines a trigger event for the next block operation, and specifies the delay between the trigger event and the start of collecting the data block. Note that the delay can be negative for pre-trigger.

enabled this is `TRUE` if the ADC-10 is to wait for a trigger event, and `FALSE` if the ADC-10 is to start collecting data immediately.

auto_trigger this is `TRUE` if the ADC10 is to trigger after a specified time (even if no trigger event occurs). This prevents the computer from locking up, if no trigger event occurs.

auto_ms specifies the time in ms after which `auto_trigger` will occur.

dir the direction can be rising or falling.

threshold this is the threshold at which a trigger event on channel A or B takes place. It is **scaled** in ADC counts.

delay This specifies the delay, as a percentage of the block size, between the trigger event and the start of the block. Thus, 0% means the first data value in the block, and -50% means that the trigger event is in the middle of the block.

adc10_set_interval

```
PREF1 unsigned long PREF2 adc10_set_interval (  
                                   unsigned long us_for_block,  
                                   unsigned long ideal_no_of_samples);
```

This routine specifies the time interval per sample and the channels to be used for calls to `adc10_get_values` or `adc10_get_times_and_values`.

us_for_block target total time in which to collect `ideal_no_of` samples, in micro seconds.

ideal_no_of_samples specifies the number of samples that you intend to collect. This number is only used for timing calculations: you can actually collect a different number of samples when you call `adc10_get_values`.

An example of a call to this routine to set for 100 readings in 10000 us is:

```
actual = adc10_set_interval (10000, 100);
```

The routine returns the actual time to collect this number of samples. This actual time may be greater than the target time if you specified a sampling interval that is faster than your computer can manage. If the specified sampling rate was too fast, you have the following choices:

- if the total time is important, collect fewer than the ideal number of samples so that the total block time is correct

- if the number of samples is important, collect the same number of samples then allow for the fact that they took longer to collect.

adc10_get_values

```
PREF 1 unsigned long PREF2 adc10_get_values (  
                                unsigned short HUGE * values,  
                                unsigned long no_of_values);
```

This routine reads in a block of values. It collects readings at intervals and from channels specified in the most recent `adc10_set_interval` call.

If a key is pressed while collecting, the routine will return immediately. The return value will be zero if a key was pressed, and the total time in micro-seconds if a block was successfully collected.

adc10_get_times_and_values

```
PREF1 unsigned long PREF2 adc10_get_times_and_values (  
                                long HUGE * times,  
                                unsigned short HUGE * values,  
                                unsigned long no_of_values);
```

This routine reads a block of values from the unit in the most recent `adc10_open_unit` or `adc10_set_port` call. It takes readings at nominal intervals specified in the most recent `adc10_set_interval` call, and returns the actual times for each reading.

If a key is pressed while collecting, the routine will return immediately. The return value will be zero if a key was pressed, and the total in micro-seconds if a block was successfully collected.

adc10_get_unit_info

```
PREF1 short PREF2 adc10_get_unit_info (char * str, short str_lth, short line, short  
port);
```

If the specified unit failed to open, this routine returns a text string which explains why the unit was not opened.

If the specified unit is open, The routine returns version information about the ADC-10 DLL, the Windows driver and the sampling rate.

Str - character string buffer for result
str_lth - length of buffer
line - 0 to 3: selects which line to return
port - the printer port number (1..3) to return information for

DOS

From DOS, it is possible to access the ADC-10 in C and Pascal using the DOS driver.

It is not possible to call the ADC-10 driver directly in **BASIC**, however the sample program `ADC10.bas` demonstrates how to driver the ADC10 directly.

The driver uses PASCAL linkage conventions.

The DOS driver does not support huge memory, so the data buffer must be less than 64k bytes.

For more information about the DOS driver, see `adc10.txt` for the ADC-10 and ADC-40, and `adc12.txt` for the ADC-12 and ADC-42.

Windows 3.x

In Windows 3.1 it is possible to use the 16-bit Windows driver, or to access the ADC-10 directly.

When running under Windows 3.x, an application is not in complete control- Windows can interrupt at any time. Interruptions occur every 55 milliseconds, and are also caused by mouse and keyboard input. As a consequence, the driver cannot always take readings at fixed time intervals. To deal with this, the driver returns the time at which each reading was taken.

The Windows 16-bit driver is called PICO.386, and is installed in windows\system. It is loaded using a reference in system.ini:

```
[386enh]
.....
.....
device=pico.386
```

The driver is accessed using the file ADC1016.DLL: this is installed in the drivers\win sub-directory: for some applications (eg Visual Basic), it is necessary to copy the DLL to c:\windows\system.

The DLL uses PASCAL linkage conventions, and uses HUGE pointers to data items, so that C and Delphi programs can access arrays larger than 64k bytes.

Examples are provided for C, Delphi, Visual Basic and Excel.

Windows 95

In Windows 95, you can use the 16-bit or the 32-bit driver: it is also possible to access the ADC-10 directly.

It is necessary to use the driver that matches the application. The following applications require 16-bit driver:

- Visual Basic 3
- Excel 5
- Delphi 1
- Microsoft C version 1.5
- Borland C 4

The following applications require 32-bit driver:

- Visual Basic 4 and above
- Excel 7 and above
- Delphi 2 and above
- Microsoft C version 2 and above.
- LabVIEW version 4 and above

The 16-bit and 32-bit drivers do not interfere with each other, so it is possible to install both drivers on the same system, as long as, for any given unit, only one driver is using it at once.

The Borland C 4 and above the Watcom C 10 and above compilers can produce either 16-bit or 32-bit applications.

When running under Windows 95, an application is not in complete control- Windows can interrupt at any time. Interruptions occur every 55 milliseconds, and are also caused by mouse and keyboard input. As a consequence, the driver cannot always take readings at fixed time intervals. To deal with this, the driver returns the time at which each reading was taken. Generally speaking, the 16-bit driver gives higher sampling rates, but the 32-bit driver is less prone to large gaps in the data.

The Windows 95 32-bit driver, PICO.VXD, is installed in windows\system, It is loaded using a reference in system.ini:

```
[386enh]
.....
.....
device=pico.VXD
```

The Windows 95 32-bit driver is accessed using the file ADC1032.DLL: it is installed in drivers\win32. The DLL uses STDCALL linkage conventions, and undecorated names.

The 32-bit DLLs for Windows 95 and Windows NT use the same calling conventions, so a 32-bit application will run without modifications on either system. Note, however, that the two operating systems require different versions of the DLL file.

Windows NT

The Windows NT driver, PICO.SYS, is installed in `windows\system32\drivers`. The operating system must be told that the driver is available: this is normally done automatically by the setup program, but can also be done manually using the `regdrive.exe` program which is copied into the PICO directory. Type in

```
regdrive pico
```

The Windows NT 32-bit driver is accessed using the file `ADC1032.DLL`: it is installed in `drivers\win32`. The DLL uses STDCALL linkage conventions, and undecorated names.

The 32-bit DLLs for Windows 95 and Windows NT use the same calling conventions, so a 32-bit application will run without modifications on either system. Note, however, that the two operating systems require different versions of the DLL file.

C

DOS

To link the driver into your program, you should take the following steps:

- #include the header file `adc10.h` into your program

- If you are using an IDE, include the file `adc10drv.obj` in your project.

- If you are using a command-line compiler, include the file `adc10drv.obj` in your linkfile.

See `adc10b.c` for an example of a simple DOS program which uses the driver.

Windows

The C example program is a generic windows application- ie it does not use Borland AppExpert or Microsoft AppWizard. To compile the program, create a new project for an Application containing the following files:

- `adc10tes.c`

- `adc10tes.rc`

- either `adc1016.lib` (All 16-bit applications)

- or `adc1032.lib` (Borland 32-bit applications)

- or `adc10ms.lib` (Microsoft Visual C 32-bit applications)

The following files must be in the same directory:

- `adc10tes.rch`

- `adc10w.h`

- either `adc1016.dll` (All 16-bit applications)

- or `adc1032.dll` (All 32-bit applicaitons)

C++

C++ programs can access all versions of the driver. If `adc10.h` or `adc10w.h` are included in a C++ program, the `PREF1` macro expands to **extern "C"**: this disables name-mangling (or decoration, as Microsoft call it), and enables C++ routines to make calls to the driver routines using C headers.

Pascal

The program `adc10.pas` can be compiled either as a stand-alone program `{ $DEFINE MAIN }` or as a unit which can be linked into other programs `{ $UNDEF MAIN }`.

`adc10.pas` includes the driver using the `{ $L adc10drv.obj }` command: it also provides pascal prototypes for each of the routine in the driver.

This program has been tested with Borland Turbo Pascal V6.0.

Basic

The DOS driver does not work with DOS Basic, however the example program `adc10.bas` shows how to drive the ADC-10 directly.

Delphi

`adc10pr.dpr` is a complete program which opens the driver and reads values from channel 1.

The file `ADC10fm.inc` contains a set of procedure prototypes that you can include into your programs.

Excel

The easiest way to get data into Excel is to use the Picolog for Windows program.

However, you can also write an Excel macro which calls `adc10xx.dll` to read in a set of data values. The Excel Macro language is similar to Visual Basic.

The example `ADC10xx.XLS` reads in 20 values from channels 1 and 2, one per second, and assigns them to cells A1..B20.

Use 16-bit driver for Excel version 5, and the 32-bit driver for Excel version 7 and above.

Note that it is usually necessary to copy the `.DLL` file to your `\windows\system` directory.

Visual Basic

Version 3 (16 bits)

The `DRIVERS\WIN16` sub-directory contains a simple Visual Basic program, `ADC10.mak`.

```
ADC1016.MAK
ADC1016.FRM
```

Note that it is usually necessary to copy the `.DLL` file to your `\windows\system` directory.

Version 4 and 5 (32 bits)

The `DRIVERS\WIN32` sub-directory contains the following files:

```
ADC1032.VBP
ADC1032.BAS
ADC1032.FRM
```

LabVIEW

The routines described here were tested using LabVIEW for Windows 95 version 4.0.

While it is possible to access all of the driver routines described earlier, it is easier to use the special Labview access routines if only single readings are required. The `adc10.lib` library in the `DRIVERS\WIN32` sub-directory shows how to access these routines.

To use these routines, copy `adc10.lib` and `adc1032.dll` to your LabVIEW `user.lib` directory. You will then find four sub-vis to access the ADC-10, ADC-12 ADC-40 and ADC-42, and some example sub-vis which demonstrate how to use them.

You can use one of these sub-vis for converter that you wish to use. The sub-vi accepts the port (1 for LPT1) and returns a voltage.

HP-Vee

The example routine `adc10.vee` is in the `drivers\win32` sub-directory. This example uses the driver definitions in `adc10.vh`. It was tested using HP-Vee version 5 under Windows 95.

The example shows how to collect a block of data from the `adc-10`. It would be necessary to alter the scaling for use with the ADC12, 40 or 42.