ADC11/22 Manual v1.0

PicoScope Manual (Oscilloscope/ Spectrum Analyser software)

PicoLog Manual (Data logging software)

Software updates

Writing you own software overview

C

Turbo Pascal

C/C++ Win

Delphi

Visual Basic

Excel

HP Vee

LabView

Direct Control

Linux

Other Information

Contact Pico

Technical Support

Signal Conditioners

Safety

Legal

Print this manual

Safety Warning

The ground inputs of the ADC-11 and ADC-22 are connected directly to the ground of your computer, in order to minimise electrical interference.

Do not connect the ground input of the ADC to anything which may be at some voltage other than ground, as you may risk damage to the ADC and your computer. Furthermore, if you connect the ADC ground to something which is ‘live’, your computer chassis may become live.

If in doubt, check by connecting a channel input to the doubtful ground point, and check that there is no significant AC or DC voltage.

Introduction

The PICO ADC-11 and ADC-22 are medium speed analog to digital converters with 11 and 22 analog input channels and one digital output. It can be used as a virtual instrument with the PicoScope program, or as a data logger using PicoLog. Alternatively, you can use the ADC-11/22 driver software to develop your own programs to collect and analyse data from the unit.

This manual describes the physical and electrical properties of the ADC- 11 and ADC-22, and explains how to use the Windows software drivers.

You should have the following items supplied with the package:

- ADC11/ADC22
- Diskettes or CD containing the software
- Installation manual

Connecting to the PC

To use the ADC11 or ADC22 you should connect the male connector on the ADC to the printer port on your computer.

To check that the unit is working start up the PicoScope program. You should immediately see a signal on screen (As the inputs are high impedance the signal will be noisy until a voltage is applied to the input). If you apply a DC source such as an AA battery you should see the signal jump to the corresponding voltage of the DC source (provided it is between 0 and 2.5 Volts!).

Specification

Product	**ADC-11/10**	**ADC-11/12**	**ADC-22**
Resolution	10 bits	12 bits	10 bits
Number of input channels	11	11	22
Input voltage range	0 - 2.5 V		
Typical sampling rate (measured on 100MHz Pentium)	15kS/s (15,000 samples per second)		
Linearity	±1 LSB at 25°C		
Accuracy (of FSD)	±1% of FSD	±0.5%	±1%
Input overvoltage protection	±30V		
Input impedance	>1MΩ		
Digital output voltage	typically 3-5 volts, depending on computer and load		
Digital output impedance	approx 1-3k ohms depending on type of computer		
Input connector	25 way female D-type		
Output connector	25 way male D-type (connects to PC printer port)		

Connections

ADC-11			ADC-22	
Pin	Function			Function
1	Digital output		1..22	Channels 1 to 22
2	Signal Ground		23..24	Signal ground
3..13	Channels 1 to 11		24	Signal ground
14	Auxiliary digital ouput see adc11_set_d02		25	Digital output
15..25	Unused			

The output pin can be used either as a digital output or as a voltage source. Note that the ADC-11 and 22 do not provide any additional protection for this output.

When using the pin as a voltage output, the output impedance will between computers from about 1 to 3kΩ, but will be consistent for a particular computer. It should be sufficient to power up to four 10kΩ thermistors, if you use a 30kΩ bias resistor for each thermistor. This output can also directly power the LM35 type of IC based temperature sensors.

The auxiliary digital output is not connected unless this feature is specifically requested when the unit was ordered.

Scaling

The ADC-11 and ADC-22 are 10-bit analog to digital converters. This means that they produce values in the range 0 to 1023 to represent voltages between 0 and 2.5 volts. To convert from ADC readings to Volts, you should multiply by 2.5 and divide by 1023. Thus, an ADC reading of 132 represents 132 x 2.5 / 1023 = 0.323 Volts.

Direct control of ADC

The following example files show how to drive the ADC-11 and ADC-22 directly:

adc11a.c	adc22drv.c
adc11a.pas	adc22drv.pas
adc11.bas	adc22drv.bas

Drivers

The ADC-11 and ADC-22 are supplied with driver routines that you can build into your own programs. Drivers are supplied 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 and NT

Note that there are a number of differences between the DOS driver and the Windows drivers. See adc11.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-11 and ADC-22.

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

Routine	**Function**
adc11_get_driver_version	Check that this is the correct driver
adc11_open_unit	Open the driver to use a specified printer port
adc11_set_unit	Select which ADC-11 unit to use
adc11_close_unit	Close the specified printer port
adc11_set_do	Set the digital output
adc11_set_do2	Set the auxiliary digital output
adc11_get_value	Get a single reading from one channel
adc11_set_trigger	Set a trigger event from a specified channel
adc11_set_interval	Set the channels and time interval for the next call to adc11_get_values, or adc11_get_times_and_values
adc11_get_values	Get a block of readings at fixed

	intervals
adc11_get_times_and_values	Get a block of readings and their times, at fixed intervals
adc11_get_unit_info	Get information about an ADC11 unit

The driver offers the following facilities:

specify the printer port that is connected to the ADC-11
take a single reading from a specified channel
specify a trigger event from a specified channel (only available in block mode)
collect a block of samples at fixed time intervals from one or more channels

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 adc11_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

adc11_get_driver_version

```
PREF1 short PREF2 adc11_get_driver_version (void);
```

This routine returns the version number of the ADC11/22 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.

adc11_open_unit

```
PREF1 short PREF2 adc11_open_unit (short port, short product);
```

This routine opens the ADC-11 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 adc11_get_unit_info to find out why it failed.

port The number of the parallel port that the ADC-11 is connected to (1 for LPT1, 2 for LPT2 etc).
Product The type of product that you wish to use
11 - adc11
22 - adc22

adc11_close_unit

```
PREF 1 short PREF2 adc11_close_unit (short port);
```

This routine closes the ADC-11 driver.

port The number of the parallel port

adc11_set_unit

```
PREF1 short PREF2 adc11_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.

adc11_set_do

```
PREF 1 void PREF2 adc11_set_do (short do_value);
```

This routine sets the state of the digital output pin for the currently selected ADC. Any non-zero value will turn the digital output on, zero will turn it off.

adc11_set_do2

```
PREF 1 void PREF2 adc11_set_do2 (short do_value);
```

This routine sets the state of the auxiliary digital output pin for the currently selected ADC. Any non-zero value will turn the digital output on, zero will turn it off.

The auxiliary digital output is pin 14: it is not connected unless this feature is specifically requested when the unit was ordered.

adc11_get_value

```
PREF 1 short PREF2 adc11_get_value (short channel);
```

This routine reads the current value of one channel from the currently selected ADC-11. Depending on your computer, it will take approx 200µs to take one reading. The `channel` number is 1 for channel 1 through to 11 for channel 11 (max 22 for the ADC22).

See also adc11_get_value_and_time, which reports the exact time at which the reading was taken

adc11_get_value_and_time

```
PREF 1 void PREF2 adc11_get_value_and_time (
            short channel,
            unsigned long * sample_time,
            unsigned short * value);
```

This routine reads the current value of one channel from the currently selected ADC-11, and the time in microticks at which the reading was taken. Depending on your computer, it will take approx 200µs to take one reading.

The `channel` number is 1 for channel 1 through to 11 for channel 11 (max 22 for the ADC22).

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 the adc value, scaled as 0 to 1023.

adc11_set_trigger

```
PREF1 void PREF2 adc11_set_trigger ( unsigned short enabled,
                                     unsigned short auto_trigger,
                                     unsigned short auto_ms,
                                     unsigned short channel,
                                     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-11 is to wait for a trigger event, and `FALSE` if the ADC-11 is to start collecting data immediately.

`auto_trigger` this is `TRUE` if the ADC11 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.

`channel` specifies which channel is to be used as the trigger input. The `channel` number is `1` for channel 1 through to 11 for channel 11.

`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.

adc11_set_interval

```
PREF1 unsigned long PREF2 adc11_set_interval (
                                unsigned long us_for_block,
                                unsigned long ideal_no_of_samples,
                                short * channels,
                                short no_of_channels);
```

This routine specifies the time interval per sample and the channels to be used for calls to `adc11_get_values` or `adc11_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 `adc11_get_values`.

`channels` The address of an array, listing the channels to be used.

`no_of_channels` specifies the number of channels used.

An example of a call to this routine using channels 2, 3 and 5 is:

int channels [3];
channels [0] = 2;
channels [1] = 3;
channels [2] = 5;

adc11_set_interval (10000, 100, channels, 3);

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.

adc11_get_values

```
PREF 1 unsigned long PREF2 adc11_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 `adc11_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.

adc11_get_times_and_values

```
PREF1 unsigned long PREF2 adc11_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 adc11_open_unit or adc11_set_port call. It takes readings at nominal intervals specified in the most recent `adc11_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.

adc11_get_unit_info

```
PREF1 short PREF2 adc11_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-11 DLL, the Windows driver and the sampling rate.

DOS

From DOS, it is possible to acess the ADC-11 in C and Pascal using the DOS driver.

It is not possible to call the ADC-11 driver directly in BASIC, however the sample program ADC11.bas demonstrates how to driver the ADC11 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.

See ADC11.txt for more information about the DOS driver.

Windows 3.x

In Windows 3.1 it is possible to use the 16-bit Windows driver, or to access the ADC-11 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 `ADC1116.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 95/98

In Windows 95/98, you can use the 16-bit or the 32-bit driver: it is also possible to access the ADC-11 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
- Borland C 5
- 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 95/98 32-bit driver is accessed using the file `ADC1132.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/2000

The Windows NT/2000 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 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 `ADC1132.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 you program, you should take the following steps:

`#include` the header file `adc11.h` into your program
If you are using an IDE, include the file `adc11drv.obj` in you project.
If you are using a command-line compiler, include the file `adc11drv.obj` in you linkfile.

See `adc11b.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:

adc11tes.c
adc11tes.rc
either adc1116.lib (All 16-bit applications)
or adc1132.lib (Borland 32-bit applications)
or adc11ms.lib (Microsoft Visual C 32-bit applications)

The following files must be in the same directory:

adc11tes.rch
adc11w.h
either adc1116.dll (All 16-bit applications)
or adc1132.dll (All 32-bit applicaitons)

C++

C++ programs can access all versions of the driver. If adc11.h or adc11w.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 `adc11.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}`.

`adc11.pas` includes the driver using the `{$L adc11drv.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 adc11.bas shows how to drive the ADC-11 directly.

Delphi

`adc11pr.dpr` is a complete program which opens the driver and reads values from channel 1.

The file `ADC11fm.inc` contains a set of procedure prototypes that you can include into your own 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 `adc11xx.dll` to read in a set of data values. The Excel Macro language is similar to Visual Basic.

The example `ADC11xx.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, `ADC11.mak`.

ADC1116.MAK
ADC1116.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:

ADC1132.VBP
ADC1132.BAS
ADC1132.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 adc11.llb library in the `DRIVERS\WIN32` sub-directory shows how to access these routines.

To use these routines, copy `adc11.llb` and adc1132.dll to your LabVIEW user.lib directory. You will then find two sub-vis to access the ADC-11 and ADC-22, and some example sub-vis which demonstrate how to use them.

You can use one of these sub-vis for each of the channels that you wish to measure. The sub-vi accepts the port (1 for LPT1) and channel (1 to 11 or 1 to 22, depending on converter type) and returns a voltage.

HP-Vee

The example routine adc11.vee is in the drivers\win32 sub-directory. It was tested using HP-Vee version 5 under Windows 95.

The example shows how to collect a block of data from the adc-11.

Linux

A Linux driver is currently under development: please check the drivers section of the Pico Technology web site (http://www.picotech.com/drivers.html) for availability.