

ADC-200 / 212 / 216 Manual v1.1

- 1. Introduction
- 2. Connecting to PC
- 3. Specifications
- 4. Principles of Operation



PicoScope Manual
(Oscilloscope/
Spectrum Analyser
software)



PicoLog
Manual
(DataLogging
software)



Software
updates

Writing your own software overview



C



Turbo
Pascal



C/C++
Win



Delphi



Visual
Basic



Excel



HP Vee



LabView



LabWindows



Linux

Other Information



Contact Pico



Technical
Support



Signal Conditioners



Safety



Legal



Print this
manual

Safety Warning

The maximum input voltage range of the ADC-200 is $\pm 20\text{V}$. Any voltages in excess of $\pm 100\text{V}$ may cause permanent damage to the unit.

The inputs to the ADC-200 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-200 (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-200 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

This manual covers the ADC-200, ADC-212 and ADC-216 products. Where information applies equally to all 3 product groups, the abbreviation ADC-2xx is used

The ADC-2xx products are high speed analog to digital converter with two input channels and software controlled input ranges. They can be used as a PC based oscilloscopes / spectrum analysers with the supplied PicoScope software or as dataloggers with the PicoLog software; alternatively, you can use the ADC2xx driver software to develop your own programs to collect and analyse data from the unit.

The ADC-2xx package contains the following items:

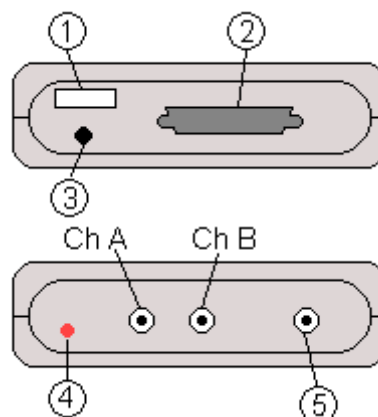
- ADC-2xx unit
- 25 way parallel port cable
- Power supply (12 Volt @ 500mA)
- Software CD
- Installation guide

Connecting to the PC

- Connect the D-connector on the ADC-2xx to the printer port on your computer using the cable provided
- Connect DC power by plugging the power adapter into a mains socket and plugging the DC power jack into the socket on the ADC-2xx. The red light should now be on, showing that the unit is powered. The light may switch off when data is not being processed.
- To check that the unit is working, start up PicoScope. 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.

The ADC-2xx has the same connectors as an oscilloscope, so you can use standard oscilloscope probes. The input impedance is also the same, so the x10 function on a scope probe works correctly. The BNC connector labeled 'E' has two functions; in normal use it is the external trigger input and accepts a TTL compatible signal. This connector can also be used as a simple (squarewave) generator. This signal generator can be used to compensate x10 scope probes.

1. BATCH number
2. D25 Parallel port connector
3. DC 12Volt@500mA power socket
4. 'Running' LED
5. External trigger/Signal generator



Specifications

Product	200/20	200/50	200/100	212/3	212/50	212/100	216
Resolution	8 bits	8 bits	8 bits	12 bits	12 bits	12 bits	16 bits
Input Channels	2 (BNC connectors with 1 M ohm impedance, AC/DC coupling)						
External Trigger	The EXT BNC connector can be used as a external TTL level trigger input OR as a square wave signal generator output						
Voltage ranges	+/-50mV to 20V in 1,2,5 steps (9 Ranges) ADC212/3 and ADC216 also have +/-10 and +/-20mV ranges						
Accuracy	+/-3%	+/-3%	+/-3%	+/-1%	+/-1%	+/-1%	+/-1%
Overload protection	+/-100V	+/-100V	+/-100V	+/-100V	+/-100V	+/-100V	+/-100V
Sampling rate							
(1 channel)	20MS/s	50MS/s	100MS/s	3MS/s	50MS/s	100MS/s	333kS/s
(2 channels)	20MS/s	50MS/s	50MS/s	1.5MS/s	50MS/s	50MS/s	166kS/s
Analog bandwidth	10MHz	25MHz	50MHz	1.5MHz	25MHz	50MHz	166kHz
Buffer Size	8K	16K	32K	32K	128K	128K	32K
Signal Generator	<250kHz TTL square wave						
Power supply	12V DC nominal at 500mA max. DC1.3mm connector (center positive)						
Dimensions	140 x 190 x 45mm						

Principles of Operation

This section explains how the ADC-2xx works. This information is intended for people writing their own software and is not required if you are only using the product with PicoScope or PicoLog software.

The ADC-2xx range includes both high-speed analog to digital converters (eg ADC-200/100) and high resolution converters (eg ADC-216). These devices take sequences of voltage measurements and feed the information into a computer.

Block sampling

When running at high speeds, the ADC-2xx can collect data much faster than the PC can read it, so the ADC-2xx reads in a block of data into internal memory, then transfers it to the PC once the block is completed. At very low speeds, it may be unacceptable to wait until the block is completed before being able to inspect the first few readings. The ADC-2xx therefore works in two modes- fast and slow.

Fast mode

In fast mode, the computer starts the ADC-2xx to collect a block of data into its internal memory. When the ADC has collected the whole block, the computer stops the ADC and transfers the whole block into computer memory.

The maximum number of values depends upon the size of the ADC-2xx memory. The unit can sample at a number of different rates which are the clock frequency divided by powers of two (half, quarter, eighth, etc). There are between 16 and 20 sampling rates, depending on the ADC-2xx model.

There is a separate buffer for each channel: on the faster models, one input can be routed into both buffers, thus doubling the effective sampling rate.

The ADC2xx driver normally performs a number of setup operation before collecting each block of data. This can take up to 50 milliseconds. If it is necessary to collect data with the minimum time interval between blocks, use the `adc200_set_rapid` option.

Slow mode

In **slow mode**, the ADC-2xx uses its internal memory as a FIFO: the computer can read values from the FIFO as soon as the readings are taken. In this mode, the ADC-2xx is running continuously, and there is no limit to the number of values that can be read.

Triggering

The ADC-2xx can either start collecting data immediately, or it can be programmed to wait for a trigger event to occur. The trigger event can occur when the channel A or B input crosses a threshold voltage, or on a change of state of the external (digital) trigger input. The trigger event can be either a rising or a falling edge.

The ADC-2xx can be programmed to place the trigger event at the beginning of the buffer, like an analogue scope, or at the end of the buffer (pre-trigger), or any point in between.

The external trigger input is the same as the signal generator output, so these two functions cannot be used at the same time.

Voltage ranges

It is possible to set the gain for each channel to give an input voltage range from $\pm 50\text{mV}$ to $\pm 20\text{V}$ ($\pm 10\text{mV}$ to $\pm 20\text{V}$ for the ADC-212/3 and ADC-216).

AC/DC operation

Each channel can be set to either AC or DC coupling. When AC coupled, any DC component of the signal is filtered out. For some older versions, there is a physical AC/DC switch for each channel on the front of the unit: for newer versions, it is controlled by software.

Oversampling

When the unit is operating at speeds below maximum, it is possible to **oversample**- to take more than one measurement during each time interval. This reduces the effects of aliasing, and increases the apparent resolution of the ADC.

Scaling

The ADC-200 is an 8-bit ADC, which returns a value between 0 and 255 to represent the currently selected voltage range. To facilitate software development, the numbers are adjusted so that 0 ADC counts corresponds to 0 volts. All values returned by the driver are scaled as if 16x oversampling is selected, so the maximum positive voltage in the selected range is represented by 2047 and the maximum negative voltage by -2047.

The ADC-212 is a 12-bit ADC, which returns a value between 2047 and -2047 regardless of the oversampling selected..

The ADC-216 is a 16-bit ADC, which returns a value between 32767 and -32767.

Signal Generator

The ADC-2xx has a built-in **signal generator**. It produces a selection of accurate frequencies from 1kHz to 250kHz. These are selected under software control. The waveform is approximately square at low frequencies, but it rounds off above about 100kHz.

The signal generator output is the same as the signal generator input, so these two functions cannot be used at the same time.

Multi-unit operation

It is possible to collect data using up to three ADC2xx units at the same time. Each ADC-2xx must be connected to a separate printer port. The routine `adc200_set_unit` select which unit the driver should access next.

For example, to collect data from units on LPT1 and LPT3 at the same time:

```
adc200_open (1)
adc200_open (3)

adc200_set_unit (1)
..... set up unit 1
adc200_run

adc200_set_unit (3)
.... set up unit 3
adc200_run

ready =FALSE
while not ready
  adc200_set_unit (1)
  ready = adc200_ready
  adc200_set_unit (3)
  ready = ready & adc200_ready ()

adc200_set_unit (1)
```

```
adc200_get_values  
adc200_set_unit (3)  
adc200_get_values
```

Slow mode

Slow mode is used to collect samples at regular intervals over long periods.

It is not currently implemented. `adc200_get_single` may be used until slow mode is available: it performs the same functions, but the user is responsible for timing when to take samples.

Driver Routines

Formats

The drivers are available in five formats:

- as a DOS standard mode object file
- as a DOS protected mode DLL
- as a Windows 3.1/Windows 95 16-bit DLL
- as a Windows 95 32-bit DLL
- as a Windows NT DLL
- as a Linux driver

Routines

The driver contains the following routines:

Procedure	Description
adc200_get_driver_version	Determine the driver version
adc200_open_unit	Open an adc-200 unit
adc200_set_unit	Switch to the ADC200 on a different port (multi-unit operation only)
adc200_close_unit	Shut down an ADC-200 unit
adc200_set_dc	Set the AC/DC switch
adc200_set_range	Set the input voltage range
adc200_set_channels	Specify channels to use (A, B, Both)
adc200_set_oversample	Specify the oversample factor
adc200_set_timebase	Set the time interval between samples
adc200_set_trigger	Specify the triggering parameters
adc200_set_rapid	Enable rapid block repeat mode
adc200_max_samples	Find out how many samples can be taken, using current settings
adc200_run	Start the ADC-200 collecting data
adc200_ready	Find out whether the ADC-200 has collected some data
adc200_stop	Stop the ADC-200
adc200_get_values	Get a block of samples from the ADC-200
adc200_get_overflow	Determine whether an overflow occurred during the last adc200_get_values operation
adc200_get_single	Get a single value from each channel
adc200_get_unit_info	If open failed, get fault info If open succeeded, get unit details
adc200_has_relays	Find out whether the adc-200 has software-controlled AC/DC switches
adc200_get_status	Get the error code from the most recent adc200_open_unit operation
adc200_get_product	Find out what type of unit (200/212/216) is connected.
adc200_set_frequency	Controls the signal generator.

Sequence of calls

This is the procedure for reading and displaying a block of data:

open the ADC-200

select ranges until the required mV range is located
set AC/DC switches, channels, trigger and oversampling
select timebases until the required ns per sample is located
set the signal generator frequency (if required)
start the ADC200 running
wait till the ADC200 says that it is ready
stop the ADC200
transfer the block of data from the ADC
display the data

adc200_get_driver_version

```
unsigned short adc200_get_driver_version (void)
```

This routine returns a 16-bit code that identifies the driver version. If it is possible that your software might be used with other drivers, you can use this routine to determine whether the driver is more recent than the one that you used to develop the software.

The upper byte contains the major version, and the lower byte contains the minor version.

adc200_open_unit

```
unsigned short adc200_open_unit (unsigned short port)
```

This routine opens the ADC-200 on the specified port. It returns TRUE if successful. The initialisation process takes a couple of seconds.

port The number of the parallel port that the ADC-200 is connected to (1 for LPT1, 2 for LPT2 etc).

The driver can handle up to three ADC-200 units at the same time: if you wish to use more than one unit, call `adc200_open_unit` once for each unit, then call `adc200_set_unit` to select which unit to use next.

Note: for the Windows NT version, the ADC200 does not have access to the actual base addresses for the printer ports. It assumes that they are:

LPT1	0x378
LPT2	0x278
LPT3	0x3BC

If your computer does not conform to this standard, you should enter the port number corresponding to the actual port base address.

adc200_set_unit

```
unsigned short adc200_set_unit (unsigned short port)
```

The driver can handle up to three ADC-200 units at the same time: if you wish to use more than one unit, call `adc200_open_unit` once for each unit, then call `adc200_set_unit` to select which unit to access next.

adc200_close_unit

```
void adc200_close (unsigned short port)
```

This routine stops the specified ADC-200 and powers the unit down.

adc200_get_unit_info

```
short adc200_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-200 DLL, the Windows driver and the sampling rate.

Str - character string buffer for result
str_lth - length of buffer
line - 0 to 4: selects which line of text to return
port - the printer port number (1..3) to return information for

adc200_set_dc

```
unsigned short adc200_set_dc (  
    unsigned short channel,  
    unsigned short dc)
```

This routine specifies the position of the AC/DC switch.

channel Use A200_CHANNEL_A or A200_CHANNEL_B.

dc 1 = DC, 0 = AC

adc200_set_range

```
unsigned short adc200_set_range (  
    unsigned short channel,  
    A200_GAIN gain)
```

This routine specifies the input voltage range for a channel. If the parameters are valid, it returns the voltage range in millivolts, otherwise it returns zero.

If you wish to find out all of the ranges, you can call this routine repeatedly and note the returned voltages until it returns zero.

channel Use A200_CHANNEL_A (0) or A200_CHANNEL_B (1).

is_slow This is TRUE if the ADC200 works in **slow mode** at this timebase (ie you can transfer readings from the ADC200 without stopping it).

timebase a code between 0 and 19 (not all codes are valid for all units- check the return value). Timebase 0 is the fastest timebase, Timebase 1 is twice the time per sample, Timebase 2 is four times, etc.

The time per sample is normally $\text{fastest_ns} * 2^{\text{timebase}} * \text{oversample}$.

For an ADC-200/50 (20ns fastest) with oversample 1, the timebases are

0	20ns
1	40ns
2	80ns
3	160ns
....	
18	5242880ns
19	10485760ns

For an ADC-212/3 (333ns fastest) with oversample 8,

0	2664 ns
1	5328 ns
2	10656ns
....	
15	87293952ns
16	174587904ns

adc200_set_trigger

```
unsigned short adc200_set_trigger (
    unsigned char enabled,
    A200_TSOURCE source,
    A200_TDIR direction,
    A200_TDELAY delay_percent,
    short threshold)
```

This routine defines a trigger event and specifies what data block to collect, with respect to the trigger.

enabled This is TRUE if the ADC200 is to wait for a trigger event, and FALSE if the ADC200 is to start collecting data immediately.

source 0 - A200_TSOURCE_A
 1 - A200_TSOURCE_B
 2 - A200_TSOURCE_E - use external logic input as trigger

direction 0 - A200_RISING
 1 - A200_FALLING

delay_percent This specifies the delay, as a percentage of the block size, between the trigger event and the start of the block. It should be in the range -100% to +100%. Thus, 0% means that the first data value in the block, and -50% means that the trigger event is in the middle of the block.

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

adc200_set_rapid

```
unsigned short adc200_set_rapid (
    unsigned short enabled)
```

This routine enables rapid repeat mode, where the driver initialises the adc200 only once, then several blocks can be collected in rapid succession. Block repeat rates of 200 per second are possible.
enabled This is TRUE to enable rapid repeat mode, FALSE to disable it.

The following example shows how to collect 50 blocks of 100 samples. Note that the first call to `adc200_run` will take 50-100ms longer than subsequent calls:

```
adc200_set_rapid (TRUE);
for (i = 1; i < 50; i++)
{
    adc200_run (100);
    while (!adc200_ready ())
        {};
    adc200_stop ();
    adc200_get_values (buffer, buffer, 100);
}
adc200_set_rapid (FALSE);
```

adc200_max_samples

```
unsigned long adc200_max_samples (void)
```

This routine returns the maximum number of samples that you can ask for. This is affected by a number of factors:

- ADC200 model
- channel mode (single/dual)
- oversampling factor
- trigger delay

You should therefore call this routine after you have selected the parameters listed above.

The adc200 operates so fast that it takes a couple of hundred readings to start and stop the converter.

If triggering is enabled, the pre- and post-trigger buffer is allocated in units of 512 bytes. Post-trigger is implemented by collecting data starting at the trigger event, but only returning data after the specified trigger delay, so for 100% post trigger about half of the buffer is unused.

For some 8-bit units, the unit can route the same channel to both memory banks, so the effective memory size is doubled.

The following approximate formula can be used:
 $\text{max sample} = (\text{buffer size} - 1000) / \text{oversample}$

adc200_run

```
unsigned short adc200_run (unsigned long no_of_values)
```

This routine tells the ADC200 to start collecting data. It returns TRUE if the ADC200 is started successfully.

`no_of_values` In fast mode, this is the number of data values that you require.

In **slow mode**, this can be zero: the adc200 will carry on collecting data until to call `adc200_stop`.

adc200_ready

```
unsigned short adc200_ready (void)
```

In fast mode, this routine returns TRUE when the ADC-200 has collected a complete block of data.

In **slow mode**, this routine returns TRUE when there is one or more readings available.

adc200_stop

```
void adc200_stop (void)
```

Call this routine to stop the ADC-200. If you call it before a trigger event occurs, the ADC-200 may not contain valid data.

adc200_get_values

```
unsigned long adc200_get_values (  
    short huge *      buffer_a,  
    short huge *      buffer_b,  
    unsigned long     no_of_values)
```

This routine gets data from the adc200.

For the adc200 and 212, zero corresponds to zero volts: 2047 and -2047 correspond to the maximum and minimum voltage on the currently selected range.

For the adc216, zero corresponds to zero volts: 32767 and -32767 correspond to the minimum and maximum voltage on the currently selected range.

In fast mode, this routine reads in the whole block of data from the ADC200. In **slow mode**, it reads in as many readings as are available. In either case, it returns the number of readings put into the buffer.

buffer_a This contains pointer to the buffer to put data from channel A into. It is unused if the ADC is collecting only from channel B.

buffer_b This contains pointer to the buffer to put data from channel B into. It is unused if the ADC is collecting only from channel A.

no_of_values The maximum number of data values to transfer. If larger than the number of values available, the return value indicates the actual number of values transferred.

adc200_get_overflow

```
short adc200_get_overflow (  
    short channel)
```

This routine determines whether an overflow occurred (the input voltage went above or below the limits of the selected range) on the specified channel. It returns TRUE if an overflow occurred.

channel 0 - channel A
 1 - channel B

adc200_get_single

```
void adc200_get_single (  
    short far *    buffer)
```

This routine starts the adc200, collects a small number of samples and then returns the average of these samples. It is intended as a stopgap until slow sampling is implemented.

buffer a pointer to a buffer containing two integers. On return from this routine, the first entry contains a reading from channel A and the seconds entry contains a reading from channel B.

adc200_set_frequency

```
long adc200_set_frequency (  
    long frequency)
```

This routine controls the signal generator. If the frequency is zero, the signal generator is turned off. If the frequency is between 1 and 250,000, the driver starts the signal generator at the nearest available frequency. The returned value is the actual frequency.

Note: The signal generator stops if you call any routine other than `adc200_ready`.

frequency the required frequency, in Hz.

adc200_has_relays

```
short adc200_has_relays (void)
```

This routine determines whether the adc200 unit has relays to control the AC/DC switches. If it returns TRUE, the `adc200_set_dc` routine can be used to set the AC/DC switches.

adc200_get_status

```
short adc200_get_status (void)
```

This routine returns the status from the most recent call to `adc200_open_unit`. The codes are defined in ADC200.h:

- 0 - A200_OK
- 1 - A200_INVALID_PORT,
- 2 - A200_INVALID_HW_VERSION,
- 3 - A200_INVALID_SW_VERSION,
- 4 - A200_CONFIG_FAILED,
- 5 - A200_ADDR_READ_FAILED,
- 6 - A200_NVR_FAIL,
- 7 - A200_UNIT_NOT_FOUND,
- 8 - A200_INVALID_LENGTH,
- 9 - A200_DRIVER_NOT_FOUND,
- 10 - A200_OLD_DRIVER_VERSION

adc200_get_product

```
short adc200_get_product (void)
```

This routine returns a value which indicates what type of adc200 is attached. The options are

- 200 - ADC-200 8-bit converter (but the driver returns 12-bit values)
- 212 - ADC-212 12-bit converter
- 216 - ADC-216 16-bit converter

DOS Standard mode driver

The DOS standard mode driver is supplied as an object file, `adc200.obj`. It can be used in C

There is also a **protected-mode** version for DOS.

DOS protected-mode driver

There is also a protected-mode version `ADC200PM.DLL` which can be used with C and Pascal programs. This driver is available on request from Pico.

Windows 16-bit Driver

The windows 16-bit driver is the file `ADC20016.DLL`. It has been tested under Windows 3.11 and under Windows 95 and 98 with 16-bit applications.

The DLL has been tested with

- **Borland C version 4.52**
- **Delphi version 1**
- **Visual Basic version 3**
- **Excel version 5**

The driver requires `pico.386` to be active. This file is normally loaded when you install the software. To check that the following line occurs in your `system.ini` file:

```
[386Enh]
:
:
device=pico.386
:
:
```

Windows 95/98 32-bit Driver

The Windows 95/98 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 ADC20032.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 driver

The Windows NT/2000 driver, `adc200.sys`, is installed in `c:\winnt\system32\drivers`. This file is normally loaded when you install the software. To check that the driver is loaded,

- press the start button
- select settings
- select control panel
- select devices
- check that `adc200` is present and marked as started.

If not, check that the driver is present and then use the `regdrive.exe` program which is copied into the PICO directory. Type in

```
regdrive adc200
```

The Windows NT 32-bit driver is accessed using the file ADC20032.DLL: it is installed in `drivers\win32`. The DLL uses STDCALL linkage conventions, and undecorated names.

Note: The Windows NT driver does not have access to the actual base addresses for the printer ports. It assumes that they are:

```
LPT1  0x278  
LPT2  0x378  
LPT3  0x3BC
```

If your computer does not conform to this standard, you should enter the port number corresponding to the actual port base address in the `adc200_open_unit` call.

Linux driver

See the **man** information in the `adc200.tar` file for more information.

Pascal

It is no longer possible to support the ADC200 using the DOS standard mode driver, as Turbo Pascal cannot handle the initialised data items within the driver.

There is a protected mode driver which can be used with Turbo Pascal 7. This driver is available on request from Pico.

C (DOS)

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

#include the header file `adc200.h` into your program

If you are using an IDE, include the file `adc200.obj` in your project.

If you are using a command-line compiler, include the files `adc200.obj` in your linkfile.

See `a200.c` for an example of a simple DOS program.

The DOS driver has been tested with the following compilers:

Borland C++ V4.5

Microsoft Visual C V 1.5

Watcom C V10.0

C (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:

`a200test.c`

`a200test.rc`

either `adc20016.lib` (All 16-bit applications)

or `adc20032.lib` (Borland 32-bit applications)

or `adc200ms.lib` (Microsoft Visual C 32-bit applications)

The following files must be in the same directory:

`a200test.rch`

`adc200.h`

either `adc20016.dll` (All 16-bit applications)

or `adc20032.dll` (All 32-bit applications)

Delphi

The program `adc200.dpr` demonstrates how to operate the ADC200. The file `adc200.inc` contains procedure prototypes that you can include in your own programs. This has been tested with Delphi versions 1, 2 and 3.

Excel

Excel 5

Load the spreadsheet `adc20016.xls`

Select Tools | macro

Select `getadc200`

Select Run

Excel 7 (Office 95 etc)

Load the spreadsheet `adc20032.xls`

Select Tools | macro

Select `getadc200`

Select Run

Note: The Excel Macro language is similar to Visual Basic. The routines which return a TRUE/FALSE value, return 0 for FALSE and 1 for TRUE, whereas Visual basic expects 65535 for TRUE. Check for > 0 rather than =TRUE.

Visual Basic

Version 3

The WIN16 sub-directory contains the following files:

ADC200.mak
adc20016.frm - procedure definitions, form and program

Version 4 and 5

The WIN32 sub-directory contains the following files:

adc20032.vbp - project file
adc20032.bas - procedure prototypes
adc20032.frm - form and program

Note: The routines which return a TRUE/FALSE value, return 0 for FALSE and 1 for TRUE, whereas Visual basic expects 65535 for TRUE. Check for > 0 rather than =TRUE.

LabVIEW

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

The adc200.vi module in the DRIVERS\WIN32 sub-directory shows how to access these routines. To use this example:

- copy adc200.vi and adc20032.dll to your LabVIEW user.lib directory
- Use LabVIEW to open the adc200.vi
- Select the printer port that your adc200 is connected to
- Press RUN

Note: for the ADC216, it will be necessary to alter the scaling in frame 7 by replacing 2048 by 32768.

LabWindows

LabWindows is a protected mode DOS program that uses C linkage conventions.

The files adc200lw.obj and adc200.lvh are available on request from Pico.

HP-Vee

The example routine adc200.vee is in the drivers\win32 sub-directory. It uses procedures that are defined in adc200.vh. It was tested using HP-Vee version 5 under Windows 95.