WILD VISION

ADC 1208

ANALOGUE TO DIGITAL CONVERTE
FOR THE ARCHIMEDES SERIES OF
COMPUTERS

USER GUIDE

Wild Vision,
15 Witney Way, Boldon Business Park,
Boldon Colliery, Tyne & Wear NE35 9PE
England.

Tel: 091 519 1455
Fax: 091 519 1929

ADC 1208

ANALOGUE TO DIGITAL CONVERTER
FOR THE ARCHIMEDES SERIES OF
COMPUTERS

USER GUIDE

ISSUE 3.00
May 1992

WILD VISION
15 WITNEY WAY
BOLDON BUSINESS PARK
BOLDON COLLIERY
TYNE AND WEAR NE35 9PE
Tel : (091) 519 1455
Fax : (091) 519 1929

Copyright © WILD VISION 1991

The information provided in this guide is applicable to the use of the Archimedes range of computers as of December 1991.

No warranties are given, expressed or implied, that the material contained in this guide is free from error, or is consistent with any particular standard, or that it will meet the requirements for an application. The information given should not be relied on for solving a problem whose incorrect solution could result in injury to a person or loss of property. To do so is at the user's own risk. All liability for direct or consequential damage resulting from the use of material within this guide is disclaimed.

LIST OF CONTENTS

PREFACE

The following pages describe the Wild Vision ADC-1208 analogue to digital converter card for the Acorn Archimedes computer. Sections describing both hardware and software are included, as well as fitting instructions and a description of how to get started. Appendices include sections on calibration procedures, error numbers, and use with high level languages. A demonstration disc is supplied and contains several examples of how the card can be used in applications. Full instructions for using the demonstration software are supplied in text files on the disc. This manual is also included as a text file on the disc.

The manual should be carefully read before using the ADC-1208, as although the card is extremely easy to use, the number of possible methods of using the card is large. So that the user can make logical decisions on the choice of parameters and specific commands a decision flowchart is provided. To use this facility a general knowledge of the ADC-1208 is required.

INTRODUCTION TO THE ADC 1208 ANALOGUE TO DIGITAL CONVERTER

Computers are beginning to play an increasing role in the control and measurement of events that occur outside of the computer. This poses a problem because within the computer the processor deals with binary encoded numbers in the form of digital logic. These binary signals are stored in discrete memory locations, whereas the signals that are found outside the machine are often continuous and continuously variable. These analogue signals from the real world often need to be measured and controlled. A device must be provided to form the interface between the external environment of the computer and the discrete digital internal world of the processor. This is the role performed by the analogue to digital converter (ADC).

The ADC-1208 is a fast, accurate analogue to digital converter card for the Archimedes range of desktop personal computers. It has a wide range of uses in any application where a processor is required to make measurements on signals external to the computer. These applications include :-

Real-time signal analysis

Machine and industrial measurement and control

Remote sensing

Quality assurance and inspection

Biological measurements

Medical monitoring and analysis

Geological, seismological and geographical applications

Architectural and environmental acoustics

The ADC-1208 is a standard single-width expansion card which may be used in any available expansion slot on all versions of the Archimedes machine. It is a multi-layer board and connects to the expansion card backplane via a DIN4162 64-way edge connector. All the on-board power supply rails needed are obtained from the Archimedes power supply via this connector.

The ADC-1208 has been carefully designed so as to match the inherent high speed of the Acorn RISC processor thus optimising the performance of the combined system.
The expansion card can sample analogue voltages at rates up to 160000 samples per second and convert these voltages in to digital format with a resolution of 12 bits (1 part in 4096). The card also provides 8 digital input/output lines. The analogue input D connector on the rear of the card can accept 8 analogue signals in the voltage range -5 to +5 volts.

The ADC-1208 is the only system currently available for the Archimedes which allows accurate, high speed analogue measurement. Along with the on-board software it provides a flexible but simple basis for computerised signal measurement and analysis.
The ADC-1208 is supported by a range of software options.

GETTING STARTED

After installing the ADC-1208 in one of the backplane slots of the Archimedes computer (see Appendix 1), you can now switch on. No other interfacing to the host computer is required. All the driver software required to operate the ADC-1208 is contained in an on-board ROM which is automatically copied from the expansion card into the computer on power-up. Subsequent use of the ADC-1208 is made possible by access to this software by the use of SWI or SYS calls. Star commands (*) can also be used.

As a simple example of the use of the ADC-1208 the user can enter the following program in BASIC.

```
CLS
*CLOCK 2000
*CHANNEL 1
*ADC 1024 1
*DISPLAY 1024 1
```

or

```
CLS
SYS"WVAIO_Clock",2000
SYS"WVAIO_Channel",1
SYS"WVAIO_ADC",1024,1
SYS"WVAIO_Display",1024,1
```

If a signal is applied to the channel one input of the analogue input connector (see the next section for details) on the rear of the inserted card, and this program is RUN, the ADC-1208 will collect 1024 samples of the input signal on channel 1 over half a second and then display the results on the VDU screen. The test signal should have a frequency of less than 1000 Hertz for correct operation (consult a book on sampling techniques if this is not understood).

You can also run some of the simple demonstration programs on the utility disc provided. These elementary programs are provided as a simple introduction to the use of the software which is provided in the ADC-1208 ROM. They show how easy it is to control the ADC-1208 and should be used in conjunction with the sections which describe the ADC-1208 control commands. Programs for use under the BASIC interpreter are contained in a directory BASICDEMOS.

The LANGDEMOS directory also contains a series of example programs which show how the ADC-1208 can be controlled from the computer language C. These examples are discussed in more detail in the sections CALLING THE ADC-1208 from C, PASCAL and ASSEMBLER can also be used to access the card. The LANGDEMOS directory also contains a C library for incorporation into the users' own utilities.

Recent additions to the software, manual or any corrections are listed in the file !Readme which is an ASCII file.

THE CONTENTS OF THE ADC-1208 EXPANSION CARD: SYSTEM DESCRIPTION

The ADC-1208 contains :-

i) an eight channel analogue multiplexer. Here, the eight input channels converge into a single input channel to the analogue to digital converter. The choice of the current active input channel is made by sending data to this device.

ii) a sample and hold device. The analogue to digital converter requires a constant analogue input whilst it is performing the conversion of the analogue voltage into a digital format for the host computer. The sample and hold circuitry, which is inside the analogue to digital converter chip, captures the voltage provided by the multiplexer and presents this sampled version of the input to the analogue to digital converter circuitry.

iii) an analogue to digital converter: This unit performs the process of converting an analogue input voltage into a digital format that can be processed by the host computer. The converter is directed to initiate a conversion on receipt of a command signal from the Versatile Interface Adapter (VIA).

iv) a Versatile Interface Adapter (VIA). The VIA provides eight lines of digital input/output (I/0) to the expansion card, two digital control lines and also provides the control lines for the multiplexer and the analogue to digital converter. The I/O lines and the digital control lines allow the host processor to communicate with other computers or devices using digital logic signals. These lines can be used to synchronise the analogue to digital conversions to events external to the host computer and also allows the computer to provide timing signals for external devices. The VIA also provides the control signals for selection of the analogue input channel and the number of analogue to digital conversions per second (the clock frequency).

v) interrupt demand buffers: This circuitry allows the expansion card to tell the host computer that an event has occurred that requires servicing by the host processor. The choice of interrupt is selectable by links that can be changed by the user.

vi) a read only memory (ROM). The software to allow the host Archimedes computer to control and read the results of analogue to digital conversion is provided on a ROM which is plugged into the expansion card. The contents of the ROM are automatically transferred to the Archimedes on initialisation when the computer is operating under Risc-OS. The software is in the form of a relocatable module, which is loaded into the RMA workspace area of the host computer. The contents of the module can be listed by using the *HELP ADC-1208 command or by using *HELP COMMANDS. Each of the commands has further help described by using *HELP <command>.

The ADC-1208 also contains buffer circuitry to enable it to be plugged into any slot in the Archimedes backplane.

The ADC has to be able to do three things, first it must convert an analogue voltage into a binary number that can be operated on by the computer. Secondly, because it cannot perform this conversion continuously, the ADC has to sample the analogue voltage at a particular time. Lastly, computers have to work on particular discrete memory locations which contain binary numbers, so

the ADC has to produce the results of the conversion in such a way that the host processor sees them as though it were dealing with computer memory.

The operation of the ADC can be specified in terms of the above requirements. The resolution of the conversion and the form of the coding into binary numbers can be described by the user. The accuracy and number of conversions per second can be determined and the way in which the processor sees the ADC as memory can also be defined.

Commands can divided into three groups:
Those concerned with setting up the converter prior to data acquisition-

- Viawrite
- Clock
- Channel
- NoChannels
- Trigger
- GetAdcInfo
- ViaRead

Those concerned with data acquisition-

- Adc
- ContAdc
- Adcstart
- AdcStop
- Oscillo

Those concerned with displaying the results of data acquisition-

- Display

SPECIFICATIONS OF THE ADC 1208

The ADC-1208 is a 12 bit linear converter. This means that the voltage resolution of the ADC is given by the analogue input voltage range divided by 2 raised to the power 12 (4096). For example, if the ADC can accept analogue inputs of +5 volts to -5 volts, the range of analogue input voltages is 10 volts and the quantal resolution of the ADC is 10/4096 volts which is 2.44 millivolts. The implication of this is that a voltage of 0 and a voltage of 2 millivolts at the input to the ADC will result in the same output from the ADC. Whereas an input of 2.5 millivolts would result in an output with a magnitude one greater than that produced by an input of 0 volts. Thus the outputs of the ADC can be described as being integer multiples of the input voltage divided by the analogue resolution of the ADC.

The output of the ADC-1208 is encoded as offset binary. An input voltage of -5 volts will result in a binary coded output of 000000000000, an input voltage of 0 volts will result in an output of 100000000000 and an input of +5 volts will cause an output of 111111111111. In more general terms the binary output can be described as being:

output = integer (analogue input volts + 5 volts)*4096/10

It is important to understand that this is not the same as the internal representation of numbers within the computer. Input voltages less than zero do not result in binary outputs from the ADC which are coded as negative numbers, (I.E THE OUTPUT IS NOT TWO'S COMPLEMENT BINARY). This output format does have some advantages however, in that it is extremely easy to use the output for direct display on the screen. This is treated in more detail later.

The conversion of analogue voltages into binary signals is not error free. The ADC-1208 like all ADCs may deviate from the ideal least significant bit step size. This differential non-linearity, the deviation of the staircase step width between increments in the binary code is less than 1 bit. This behaviour means that the ADC has monotonic properties, in that increasing the input voltage results in an increase in the binary output code of the ADC throughout the whole range of voltages.

The ADC can sample signals at rates up to 160000 samples per second. It takes 6 microseconds to perform the conversion of the sampled analogue signal into a binary format. The input signal does not have to be constant for the whole of that time however because a sample and hold device captures the input voltage during a 10 nanosecond window and maintains the voltage input to the ADC device throughout the conversion period. The sample rate is programmable on the ADC-1208 from 30 to 160000 samples per second. The number of samples converted by the ADC is also programmable from 1 to 800000 in a 4MByte machine (150000 on the 410 Archimedes). Thus the duration of sampling can be varied from 6 microseconds to over 7 hours.

The result of an analogue to digital conversion is a 12 bit binary number. On the ADC-1208 this result is stored in a buffer which is sited on the expansion card. The host computer can then access this intermediate result whilst the ADC starts the process of converting the next analogue sample.

The host computer has to `read' the buffer to find out the result of an analogue to digital conversion. The `read' can be initiated by the host or by the expansion card. The host processor can read the contents of the buffer at any time, but it makes most sense if access is synchronised to the sampling process on the expansion card. This can be done by either getting the expansion card to interrupt the host computer every time the sampling clock has initiated a new analogue to digital conversion, or by the host computer interrogating the expansion card to determine if a new sample

has been generated (by reading specific registers on the VIA): Both these sampling modes are supported by the software provided with the ADC-1208:

The contents of the ADC buffer have to be stored elsewhere, otherwise the result of an analogue to digital conversion will be overwritten by the next sample. The ADC software provides a means by which the contents of the buffer can be transferred to an area of memory which is termed the ADC data store:

THE MEMORY STRUCTURE OF THE ADC 1208

There are three areas of memory that are important to the ADC-1208 hardware and software. The first area is in the expansion memory of the Archimedes and these memory locations are defined by the hardware of the ADC-1208: The second area is the workspace of the ADC-1208, and contains all the current values of the control parameters of the hardware and the software: The third area defines the data storage area for the the results of ADC-1208 activity:

The Archimedes series computers can be fitted with a backplane that allows expansion cards to be inserted. These expansion cards are each allocated an area of memory such that reading from or writing to these locations is actually the reading of data from or sending data to the expansion card: Each backplane slot is allocated a series of memory locations. The actual memory locations used determines the timing of data transfers to and from the expansion card: The ADC1208 uses ' synchronous timing' of data transfers to and from the host computer and this defines the base address for each expansion card slot. The user does not need to know the base address as this information is automatically calculated by the system on power-up or after a re-initialisation of the ADC-1208. The base address of the ADC-1208 is stored in the workspace for the ADC-1208 module.

The hardware locations of the ADC-1208 are expressed as offsets from the base address of the expansion card, which is defined by the slot location used by the ADC-1208. The offsets from this base location are:-

offset &2000 = base location of the ADC-1208 VIA
offset &2100 = location of the program paging register
offset &2200 = location of the interrupt status register
offset &2300 = location of the ADC sample buffer

The ADC sample buffer can also be accessed by MEMC reads from any address within the MEMC area for that podule slot:

The user does not need to know these locations because the ADC-1208 software provides an easy to use interface between the computer and the hardware memory locations of the ADC expansion card: In fact direct peeking at these locations is not possible unless the programmer does so in an ' ARM supervisor mode'.

The workspace of the ADC-1208 is situated in the RMA workspace of the main host computer: The location of the workspace is therefore variable and depends on the content of the RMA area on initialization. The *AdcInfo command or the SWI "WVAIO_GetAdcInfo” return the pointer to the parameter block for the ADC-1208:

The parameter block contains the following information:-

parameter	block offset	
0	pointer to ADC data storage area	
4	pointer to second ADC data storage area	
8	current number of points	
12	current channel number	
16	current number of channels	
20	current sampling rate in Hertz	
24	current period of sampling in microseconds	
32	podule MEMC address	
60	podule base address	
64	direction of data on PA bus (always 255 - as always outputs)	
68	direction of data on PB bus	
72	current byte read from PB bus	
76	current trigger word	
80	base frequency of VIA	
88	Interrupt driven ADC flag	
92	mode of acquisition in continuous ADC sampling	
96	trigger mode in background signal acquisition	
448-759	must be preserved and not accessed in user applications	
	Single channel CONT_ADC	Multichannel CONT_ADC
760	reserved	address of data store
764	reserved	current channel
768	no points left	no points left
772	address data store	address data store
776	ADC address	ADC address
780	mask	mask
784	reserved	number of points expressed as words
1024-2047	RESERVED this area must be preserved:	
2048-	default data storage area **THIS AREA IS ONLY 8192 BYTES LONG**	

After an analogue voltage has been converted into binary format, the result of the conversion is stored in an intermediate buffer store on the ADC-1208 expansion card. This allows the ADC device to start another conversion whilst, in parallel with this next conversion, the host computer is able to read the contents of the buffer.

The contents of the buffer are overwritten by the results of subsequent analogue to digital conversions, so the host computer must read the contents of the buffer and store the data elsewhere. The software provided with the ADC-1208 allows the user to define the location of this data storage area. However, if no data storage area is provided by the user then the default data storage area in the ADC-1208 relocatable module area workspace is used.

The ADC software reads the contents of the expansion card data buffer and then writes this data to specific memory locations within the main memory of the machine: It starts writing to the first memory location and the results of all subsequent data acquisitions are stored in successive memory locations within the ADC data store. The result is that the ADC data storage area contains a record of voltage (expressed in the offset binary code format) over time. Each data word in the storage area is a sample of the voltage on the input at a particular time.

THE SWI COMMANDS FOR THE ADC-1208

On the ADC-1208 expansion card there is an eprom which contains a set of software utilities that can be used from MOST computer languages for the control of the ADC-1208 hardware: These software utilities are termed the LEVEL 0 commands and are provided in the form of extensions to the operating system. There are two forms of the LEVEL 0 commands: There are SWI (software interrupt) and * (star) command versions: The utilities are automatically loaded into the host computer from the expansion card on power-up or a hard reset and are contained within a relocatable software module. The name of the module is "ADC-1208": Further information about the module can be obtained by using:

***HELP MODULES**
***MODULES**
***HELP COMMANDS**
***HELP ADC-1208**

On initialisation the software module performs the following :-

1. A claim is made for workspace for the module software: This workspace is used to hold the parameter block for the LEVEL 0 software and a default data storage area for the results of analogue to digital conversions performed by the ADC-1208. The parameter block contains a copy of all the current parameters used to control the ADC-1208 hardware. It also contains a number of data pointers and values which are of use in the manipulation of the results of analogue to digital conversion.

2. The module software resets specific hardware registers on the ADC-1208: For example, the default input multiplexer channel is set to be number 1; the interrupt enable registers are cleared; the clock is turned off.

3. Certain control parameters for the ADC-1208 expansion card are set to default values. For example, the number of channels is set to be 1; the data storage area is set to start at a location within the workspace allocated to the ADC-1208 module; the number of points for data acquisition is set to a default of 1024.

The module can be reinitialised using:

*RMREI NIT ADC-1208module

THIS DOES NOT MAINTAIN ANY PARAMETER VALUES FROM BEFORE THE REINITIALISATION.

The LEVEL 0 commands are described in detail, with examples, in the next section: The full list of commands is:-

SWI / SYS CALLS

GetAdcInfo	Channel
ADC	Display
ViaWrite	No_Channels
ContADC	Oscillo
ViaRead	Trigger
ADCStart	Clock
ADCStop	

These calls are case sensitive and they should all be prefixed by WVAIO__ to access them: e:g

```
SYS"WVAIO_Clock",clock_rate in BASIC
```

STAR (*) commands

Adcinfo	Channel
ADC	Display
Viaread	No_channels
ContADC	Oscillo
Viawrite	Trigger
ADCStart	Clock
Adcdata	ADCStop
Clockoff	

Star (*) commands are case insensitive:

WVAIO_GetAdcInfo

(SWI &80A80)

A COMMAND FOR ACCESSING THE WORKSPACE OF THE ADC-1208 MODULE

The workspace allocated to the ADC-1208 on initialization contains a block of memory which is used to store a number of useful parameters for the control of the ADC-1208. This parameter block also contains information that describes the data which is produced by analogue to digital conversions. The address of the parameter block (which is located at the start of the workspace allocated to the ADC-1208 module) is returned to the user by the use of *AdcInfo or the SWrWVAIO_GetAdcInfo" command. The returned value can be directed to be stored at a specific address or as the updated contents of a variable. If no pointer to a return parameter is given the parameter block address is printed on the current output stream.

Syntax

1. As a star command

***ADCINFO (location }**

where (location is the address for the result of the call to be stored at.

2. As a SWI call

"WVAIO_GetAdcInfo" {variable }

where (variable will contain the address of the module workspace:

3. From the C library

WV_GetAdcInfo(void);

and returns an integer pointer to the start of the workspace.

Examples of use

in BASIC

```
DIM BL 4
OSCLI("ADCINFO "+STR$(BL))
M%=~!BL
```

this would place the address of the parameter block in the memory starting at BL. It would then store the pointer to the parameter block in M%. By using offset addressing the contents of the parameter block can be read or written to. For example to read the current dock rate in BASIC the user could PRINT M%!20.

***ADCINFO**

this would immediately print the address, in hexadecimal format, of the start of the parameter block

SYS"WVAIO_GetAdcInfo" TO A%

would place the address of the start of the parameter block in variable A%.

WVAIO_ViaWrite

(SWI &80A81)

A COMMAND FOR WRITING A BYTE OF INFORMATION TO A REGISTER IN THE VERSATILE INTERFACE ADAPTOR

The ADC-1208 contains a 6522 versatile interface adaptor. This device controls the multiplexer and the eight digital input/output lines: This command can be used to write to any of the registers on the VIA. The data is sent to the VIA as a byte.

The user is free to program the CB1 and 2 control lines, and the PB bus directly, except that PB7 is used for all internal clocking of the ADC circuitry: The CA bus, CA1 and CA2 are reserved solely for internal use as they are used to program the channel multiplexer and for high speed interrupts: This places a small constraint on the use of some registers. The ADC-1208 module normally sets all the registers up correctly for use: However if the user wishes to directly manipulate the 6522 they should take care to note the following:

For correct operation of the ADC-1208, register 12, the peripheral control register, should always be &C <AND> any value the user wishes to reset this register to. This is the default: Registers 4 and 5 are used for internal dock generation: This is enabled by the use of register 11, the auxiliary control register, set to &C0. this is the default.

Register 14 is used to generate fast interrupts when loaded with &40. Extreme care should be taken when changing this register's contents. In particular, a fast interrupt routine must be present: This is performed by default when WVAIO_ContAdc is used.

Syntax

1. As a star command

   ***VIAWRITE {register } {byte_to_bewritten }**

2. As a SWI call

   SWI "WVAIO_ViaWrite",register,byte (register ,byte}

3. From the C library

   WV_Via_Write(int register,char byte);

Range of values

Register	0-15
Byte	0-255

Examples of use

in BASIC

***VIAWRITE 2 255 *VIAWRITE 0 31**

would set the external digital I/O bus of the ADC1208 to be 00011111.

OSCLI("VIAWRITE 2"+STR$(variable))

would output the contents of <variable> MOD 256 to register 2.

SYS"WVAIO_ViaWrite",13,variable

would set register 13 to the contents of <variable> to MOD 256.

SYS"WVAIO_ViaWrite",0,A% TO A%,B%

would output on the digital bus the value of A% MOD 256 and copy the byte written into the variable B%: A% would contain the register written to following the call: The registers of the VIA are described in appendix 1.

WVAIO_ViaRead

(SWI &80A82)

A COMMAND FOR READING THE INFORMATION FROM THE VIA

This command can be used to find out the current state of a register in the VIA of the ADC-1208: The result of the read operation is placed in a variable, stored at a specified location or if no return parameter is given then the result is sent to the current output stream.

Syntax

1. As a star command

***VIAREAD <register> {address }**

2. As a SWI call **SWI "WVAIO_ViaRead",register {register ,byte_read}**

this allows repeated reading of the same register

3. From the C library

WV_Via_Read(int register);

returns the value read as an integer.

Range of Values

Register 0-15

Examples of use

in Basic

```
DIM BL 4
OSCLI("Viaread 0"+STR$(BL))
```

would read the current state of register 0, that is the eight digital input lines, and store the result at the address of BL.

***Viaread 0**

would print the current state of the eight digital input lines on the currently selected output stream.

SYS"WVAIO_ViaRead",0 TO ,A%

would store the state of the digital input lines as the contents of variable A%.
The registers of the VIA are described in appendix 1.

WVAIO_Clock

(SWI &80A83)

A COMMAND TO SET THE SAMPLING RATE FOR ANALOGUE TO DIGITAL CONVERSIONS

The ADC-1208 uses a programmable 16 bit counter/timer as a clock for analogue to digital conversion timing: This command can be used to reset the dock to a new rate for data sampling: If no parameter is given the current clock rate is printed, immediately, on the currently selected output stream:

The dock generates a square wave signal on the digital bus line PB7. Therefore the internal clock can be monitored, or used to synchronise external events: This signal is also sent to the analogue to digital converter device on the ADC-1208 board. It is this signal which forces a conversion of an analogue signal to a digital format. Thus the clock must be running for analogue to digital conversions to take place. (An external clock can provide this signal by asserting high/low transitions on the PB7 line if PB7 is placed into an input only mode).

There is a maximum rate for analogue to digital conversions. In the ADC-1208 this rate is 160000 samples per second. This is because it takes just over 6 microseconds to convert a single voltage sample into a digital value.

It should also be noted that the clock frequency can only be set such that its period is a multiple of 1. 0 microseconds. This is because all the programmable clock rates are derived from a 2MHertz master clock. If the *clock command is used to set the sampling rate to a frequency which is not attainable by the hardware, THE NEXT POSSIBLE HIGHEST SAMPLING RATE IS USED. The current dock rate can be read by either using *clock without a command tail parameter or by using a SWI call:

Syntax

1. As a star command

   ***CLOCK (rate }**

2. As a SWI call

   SWI "WVAIO_Clock",sampling_rate {frequency_used }

3. From the C library

   WV_Clock_Set(int clock_rate);

this function returns the clock rate used as an integer.

Range of values

sampling rate	30-160000

Examples of use

in BASIC

***CLOCK 4096**

would set the sampling rate to 4096 analogue to digital conversions per second:

***CLOCK**

would print the current sampling rate used by the hardware

SYS"WVAIO_Clock",freq% TO rate

would set the hardware clock to a rate, which would be the closest possible sampling rate to freq%: If freq% has a period which is an integer multiple of 1.0 microseconds then rate will equal freq%.

WVAIO_Channel

(SWI &80A84)

A COMMAND FOR SELECTING THE CURRENT ANALOGUE INPUT CHANNEL

There are eight analogue input channels on the ADC-1208 expansion card. The channels are numbered 1 to 8. Only one channel may be active at at any one moment: In single channel acquisition, this command allows the selection of the specific channel: The default channel is number 1 and is selected on initial power-up or on a hard-reset. If no command tail parameter is supplied with the command then the currently selected channel is printed.

Syntax

1. As a star command

    ***CHANNEL {channel_number }**

2. As a SWI call

    SWI "WVAIO_Channel"{bold ,channel_no} {curr_chan }

3. From the C library

    WV_Channel_set(integer channel);

this function returns a void.

Range of Values

Channel number 1-8

Examples of use

in BASIC

***CHANNEL 1**

would set the analogue input channel to number 1.

***CHANNEL**

would print the current analogue input channel number on the current output stream

SYS"WVAIO_Channel",3

would select analogue input channel number 3.

SYS"WVAIO_Channel",0 TO C%

would return the current channel number as the contents of variable C.

WVAIO_Trigger

(SWI &80A85)

A COMMAND FOR SETTING THE TRIGGERING MODE FOR DATA ACQUISITION

The acquisition of a block of data can be synchronised to an external event or can provide a digital output which can signal to other devices that acquisition has started. This command can be used to determine the mode of triggering. Seven modes are available: If no parameter is supplied or the parameter is zero then the current trigger mode is printed on the currently selected output stream:

The trigger modes are:-

0 default or print current mode
1 no triggering protocol
2 a change in logic on CB1 on the digital input/output bus
3 a change in logic on CB2 on the digital input/output bus
4 PB0 must go high on the digital input/output bus
5 PB1 must go high on the digital input/output bus
6 a pulse is output on the CB2 line at the start of acquisition
7 a pulse is output on the PB0 line at the start of acquisition

SWI Via Write must be used to enable these events to occur.

Syntax

1. As a star command

***Trigger {mode }**

2. As a SWI call

SWI "WVAIO_Trigger",{mode } {current trigger }

3. From the C library

WV_Trigger(integer trigger);

this function returns a void.

Range of Values

Trigger mode 0-7

Examples of use

in BASIC

***Trigger**

would print the current trigger mode on the current print stream

***TRIGGER 7**

would cause a pulse on PB0 on the digital input/output line to be output at the beginning of every block of signal acquisition.

SYS "WVAIO_Trigger",2

would set CB1 to be the input trigger line. Only after a trigger event had occurred on this line would data acquisition start:

WVAIO_NoChannels

(SWI &80A86)

A COMMAND TO SELECT THE NUMBER OF ANALOGUE INPUT CHANNELS TO BE USED IN MULTIPLEXED ANALOGUE DATA SAMPLING

The input channels can be sampled successively by the analogue to digital converter hardware: This command allows the number of channels to be determined: The maximum number of channels is eight, but any number up to eight may be used: The sample rate per channel can be calculated by dividing the clock rate by the number of channels selected. Each channel is selected in turn and after all the channels have been selected, the channel list is reset to start from channel 1: The number of samples acquired is actually per channel. Thus if 1024 samples are specified on 6 channels, then the data area must be 1024*6*4 bytes long. THE DEFAULT DATA AREA IS 8192 BYTES LONG.

If no parameters are given, then the current number of multiplexed channels is printed on the output stream.

Syntax

1. As a star command
   ***NOCHANNELS {of channels}**

2. As a SWI call
   SWI "WVAIO_NoChannels",{no_channels } TO {no_channels }

3. From the C library
   WV_NoChannels(integer no_channels);

this function returns a void:

Range of Values

Number of channels 1-8

Examples of use

in BASIC

***NOCHANNELS 4**

would set the maximum number of channels to four. The hardware will then sample analogue input channels 1,2,3 and 4 successively.

***NOCHANNELS**

would print the current number of input channels that will be multiplexed by the software module:

WVAIO_ADC

(SWI &80A87)

A COMMAND FOR PERFORMING A SET NUMBER OF ANALOGUE TO DIGITAL CONVERSIONS

This command is used to initiate a set number of analogue to digital conversions on the ADC-1208 expansion card. The results of the analogue to digital conversion are stored in successive word aligned locations within the current ADC data storage area (as determined by *ADCDATA): The frequency of sampling is determined by the current value of the programmable dock, (as determined by *CLOCK). The command allows single channel acquisition and multiplexed channel acquisition: The single channel should have been previously selected using *CHANNEL. The number of multiplexed channels can be selected with this command or can be selected using *NO_CHANNELS: The start of the analogue to digital conversions can be triggered by a digital event either external or internal to the host computer.

Syntax

1. As a star command
   ADC <no of points> {of channels} {of data store} {source }

2. As a SWI call
   SWI"WVAIO_ADC", no_of_points, no_of_channels, data_store, trigger{TO no_points, no_channels, data_area, trigger}

3. From the C library
   WV_adc(integer no_points,integer no_channels,integer *data_store,integer trigger);
and returns a void.

Range of Values

Number of points	1- size of data block
No of channels	1-8
Trigger	1-7

Examples of use

in BASIC
***ADC 1024,2**

would capture 1024 data samples on two multiplexed input channels: It would use the current data store and trigger source.

***ADC 2048,1,&8F10,2**

would capture 2048 data samples on the currently selected input channel. It would store the data in

memory starting at location &8F10 and would not start the acquisitions until a change in logic state had occurred on the CB1 line on the digital input/output bus.

```
DIM BL 1024
SYS"WVAIO_ADC",256,1,BL,0
```

would capture 256 data samples on a single input channel and store the results in successive word aligned locations in memory starting at location BL.

WVAIO_ContADC

(SWI &80A88)

A COMMAND FOR SETTING UP THE ADC1208 TO PERFORM BACK-GROUND DATA ACQUISITION

The *ADC command is used to perform data acquisition as a foreground task: That is, the computer processor is dedicated to performing just one task all the time and that task is scanning the programmable clock to determine whether or not to read the contents of the ADC buffer: This technique is very useful if the extremely high sampling rates of the ADC- 1208 are to be used: However it can be inconvenient if the number of acquisitions is very high or the sampling rate is very low, as the processor will be held up for a long time: Because of the high speed of the host processor the percentage of time taken on data acquisition is low and so this form of acquisition can be inefficient. To circumvent these problems the command *CONTADC is provided: This command sets up a background task that operates under interrupt control: The acquisition of analogue data is initiated by the ADC-1208 expansion card and the processor responds by reading and storing the latest result from the ADC card. The command only sets up the protocol for interrupt driven acquisition, the actual acquisition can be started using *ADCSTART, (or stopped using *ADCSTOP): Once started the analogue to digital conversions take place in the background to the main task, which should not use fast interrupt protocols. On completion the software sets a flag in the ADCINFO parameter block. The number of acquisitions that have still to be performed under CONTADC control is also available in the ADCINFO parameter block: The mode parameter can be used to cause repeated acquisition of data. If it is set to zero then the acquisition of data continues until an ADCSTOP command is given:

As with other ADC protocols the acquisition of a block of data can be synchronised to internal or external events by using the trigger option.

The user should be extremely careful in using this command. Within the Archimedes range of computers there is provision for only one fast interrupt routine at a time. This means that there can be serious clashes between hardware which all require FIQs. For example econet can not be used simultaneously with background analogue to digital conversions.

Syntax

1. As a star command
   ***CONTADC <no of points)> {of channels} {storage{style bold off} area} {trigger } { mode }**

2. As a SWI call
   SWI "WVAIO_ContADC",no points,no channels,address pointer,trigger,mode

3. From the C library
   WV_ContADC(integer no points,integer no channels,integer *address pointer,integer trigger,integer mode);

Range of Values

Number of points	1- size of data block
Number of channels	1-8
Trigger	1-7
Mode	0-1

Examples of use in BASIC

***CONTADC 256 1**

would set up a protocol and software for performing 256 analogue to digital conversions on 1 channel as a background task. This task would start after a ***ADCSTART** command. The results of the task would be stored in the default ADC data storage area. The conversions would not be triggered by any external event:

```
DIM BL 16384 SYS"WVAIO_ContADC"
,1024,4,BL,0,1
```

would perform 1024 acquisitions on 4 channels and store the results in the data block BL: The task would be initiated by a ***ADCSTART** command.

WVAIO_ADCStart

(SWI &80A89)

A COMMAND TO START BACKGROUND ANALOGUE TO DIGITAL CONVERSIONS

This command is used in conjunction with the ***CONTADC** command and is used to initiate the background FIQ (fast interrupt) driven acquisition of analogue data:

Syntax

1. As a star command

   ***ADCSTART**

2. As a SWI call

   SWI "WVAIO_ADCStart"

3. From the C library

   WV_adcstart(void);

WVAIO_ADCStop

(SWI 80A8A)

A COMMAND TO STOP BACKGROUND ANALOGUE TO DIGITAL CONVERSIONS

This command is used in conjunction with the ***CONTADC** command and is used to stop the background FIQ (fast interrupt) driven acquisition of analogue data: It returns the control of the fast interrupt service to the default owner (usually Econet). A new ***CONTADC** command must be given before a new background acquisition task can be initiated.

Syntax

1. As a star command

   ***ADCSTOP**

2. As a SWI call

   SWI "WVAIO_ADCStop"

3. From the C library

   WV_adcstop(void);

WVAIO_Display

(SWI &80A8B)

A COMMAND FOR DISPLAYING THE SAMPLED DATA STORE ON THE SCREEN

This command can be used to display the results of analogue to digital conversion in a graphical format: It does not change the display mode or colours, so that the user can select these prior to calling the display command: It plots the results starting at either the currently selected address of the ADC data store or from an address supplied as a parameter to the command: The number of points displayed can be selected as can the number of channels. If one channel is chosen and the number of points selected is greater than that originally selected for analogue to digital conversion then it will still display the contents of the store. This is intentional so that multi-channel results can be displayed as one continuous line across the screen. However if the number of channels for display is greater than one, then the individual channels are superimposed: By changing the address parameter, any area of memory can be displayed! This is useful for window environments when it is necessary to scroll through different data sections. The vertical scale of the display can also be selected:

Syntax

1. As a star command

    ***Display {of points} {of channels} {scale } {star t}**

2. As a SWI call

    SWI "WVAIO_Display",no_points,no_channels,scale,data_store

Range of Values

Number of points	1- size of data block
Number of channels	1-8
Scale	1-4095

Examples of use

in BASIC

***DISPLAY 1024**

would display 1024 points across the screen. The data plotted would start from the current ADCDATA location and would be plotted at a vertical scale of 1 (the default).

***DISPLAY 256 2 2**

would display two channels of 256 points superimposed on the screen. The data from each channel would be multiplied by two before plotting on the screen.

DIM BL 2048 SYS"WVAIO_Display"512,1,2,BL

would display 512 points, starting with the data stored at location BL, with a vertical scale multiplier of 2.

WVAIO_Oscillo

(SWI &80A8C)

A COMMAND FOR CONTINUOUSLY DISPLAYING THE RESULTS OF ANALOGUE TO DIGITAL CONVERSION

The *Oscillo command makes the host computer act just like a digital oscilloscope: It acquires data from the analogue world and immediately displays the result. The data is retained in memory and can be accessed by other commands such as *DISPLAY etc: It can be used to display up to eight channels simultaneously: Because it writes to the display legally (i.e: it will work in any graphical display mode) it does not allow extremely fast acquisition modes: The acquisition can be triggered or synchronised to external digital events by setting the trigger option.

Syntax

1. As a star command

   ***Oscillo {of points} {of channels} {scale } {bold off} rate} {address } {trigger }**

2. As a SWI call

   SWI "WVAIO_Oscillo",no_points,no_channels,scale, sampling_rate, address, trigger
   {no_points ,no_channels,scale,rate,address,trigger}

Range of Values

Number of points	1- size of data block
Number of channels	1-8
Scale	1-4095
Sampling rate	30-160000
Trigger	1-7

Examples of use

in BASIC

***OSCILLO 128 3 2 2000**

would display 3 channels of 128 points with an increase in vertical display sensitivity of two. The sampling rate would be 2000 Hz:

*CLOCKOFF

A COMMAND TO TURN OFF THE PROGRAMMABLE SAMPLING RATE CLOCK

This command can be used to turn off the clock signal that initiates analogue to digital conversions: This is necessary if the full eight digital lines of the digital input/output bus are required: The clock signal uses line PB7: NOTE THAT ANALOGUE TO DIGITAL CONVERSIONS WILL STILL TAKE PLACE IF PB7 IS MADE TO CHANGE FROM LOGIC 1 TO LOGIC 0 BY AN EXTERNAL OR INTERNAL VOLTAGE:

Syntax
as a star command

***CLOCKOF**
F

THERE IS NO SWI VERSION OF THIS COMMAND. USE SWI "WVAIO_ViaWrite" TO TURN OFF THE CONTINUOUS SQUARE WAVE SIGNAL ON PB7

Examples of use

in BASIC

***CLOCKOFF**

would stop the generation of continuous square waves on digital line PB7: This stops the programmable clock being sent to the analogue to digital converter.

***VIAWRITE 11 0**

or

SYS"WVAIO_ViaWrite",11,0

can also be used to achieve this effect. A byte of value zero written to register 11 will stop the generation of a clock signal.

*ADCDATA

A COMMAND TO DIRECT THE STORAGE OF CONVERTED ANALOGUE DATA TO A SPECIFIC AREA OF MEMORY

The *ADCDATA command is used to change the start location of a memory area within which the results of analogue to digital conversions will be stored:

When the ADC-1208 is initialised, on power-up or on a hard reset, an area of memory is reserved for storing the data produced by analogue to digital conversions: In version 1:10 of the ADC-1208 module, this area is 1K words long: This is sufficient for storing the results of 1024 successive analogue to digital conversions on 1 input channel or just 128 multiplexed analogue to digital conversions on each of eight input channels: As this default area of memory may be too small for a specific user application, *ADCDATA can be used to redirect the data to another larger area of memory: Similarly the data acquisition SWI calls can be used to redirect the storage of data to the user's own memory area.

THE USER SHOULD VERIFY THAT THE LOCATIONS THAT WILL BE WRITTEN TO BY THE ANALOGUE TO DIGITAL CONVERTER ARE AVAILABLE FOR THAT USE. YOU ARE ADVISED TO CLAIM A SECTION OF MEMORY AND USE THE FACILITIES WITHIN THE SWI CALL OR *ADCDATA TO POINT TO THE FIRST WORD ALIGNED ADDRESS WITHIN THAT AREA. FAILURE TO TAKE THIS PRECAUTION WILL HAVE DISASTROUS CONSEQUENCES. If no command tail parameter is supplied with the command then the current start location of the ADC data memory is printed on the currently selected output stream:

Syntax

as a star command

***ADCDATA (location }**

there is no SWI version of this command. It is not needed as other SWI calls allow the location for data storage to be specified as one of the parameters for that SWI.

Examples of use

in BASIC

```
DIM BL 8192 OSCLI("ADCDATA "+STR$(BL))
```

would set aside 8192 bytes of contiguous memory and then allocate this memory to storage of the ADC data. This size of memory is sufficient for 2048 words of data.

```
*ADCDATA
```

would immediately print the address of the currently selected data block on the current output stream: It prints the address in hexadecimal format:

USING THE ADC 1208 IN C

There are two methods for accessing the ADC-1208 from the C language: The first is the use of a system() call and the second is the use of the SWI() call from the RISC_OS library: They correspond to the use of either star (*) commands or the use of SWI calls: There is however, a major advantage in using swi calls as the software can then directly communicate the results of a call to variables within the C program. The examples below show how these two methods can be used:

Using the System 0 function

```
#include <stdio.h>
#include <stdlib.h>
int main()
(result ;
result= system("clock 100000");
```

This sets the programmable clock to 100000 samples per sec.

```
result= system("channel 1");
```

This would set the analogue input to channel 1.

```
result= system("adc 512 1");
```

This causes 512 samples to be acquired and stored in the default adc data storage area: The conversions are performed on one channel

```
result= system("display 512 1 2");
```

The results of the acquisition are displayed with a scale of 2:

```
return 0;
}
```

The variable result will be 0 if the call to the system command line interpreter was successful and something else (this depends on the specific call) if an error occurs.

Using the SWI function

```
#include <stdio.h>
#include "os.h"

 #define WVAIO_swi 0x80A80

 int main()

 {no_points , no_channels, swi_number, clock,scale;
 os_regset myregs;
 int a[8192];

 no_points= 8192;
 no_channels= 1;
 clock = 2000;
 scale = 1;

 swi_number= WVAIO_swi+12;
 myregs.r[0] = no_points;
 myregss[1] = no_channels;
 myregs.r[2] = scale;
 myregs.r[3] = clock;
 myregs.r[4] = (int)a;

 os_swi(swi_number,&myregs];
```

this performs a WVAIO_Oscillo call to the expansion card

```
 return 0; }
```

It should be noted that the results of analogue to digital conversion can be directed straight into an array which can then be operated on by signal processing algorithms:
An even easier method is to use the C library provided on the accompanying disc:

USING THE ADC1208 FROM PASCAL

As in C, the SWI versions of the software calls to the expansion card can be used: Again, the SWI version is recommended because of the direct way in which the results of ADC-1208 activity can be returned to variable and arrays within the user's program: Examples of use are given:

To access the SWI calls the embedded ARM code facility must be used. For example to set the programmable clock a procedure could be included in the program which contained the following :-

```
CONST WVAIO_CLOCK = 16_80A83;

PROCEDURE clock(value : INTEGER);

VAR
        pointer 1 : INTEGER;

 BEGIN
         pointer 1 := ADDRESS(value);
*LDR_R1,pointer1;
*LDR_R0,[R1]
*SWI_WVAIO_CLOCK;
END;
```

USING THE ADC1208 FROM ASSEMBLER

The ADC-1208 can be controlled from assembler either by directly writing and reading the memory locations which are allocated to the ADC-1208 expansion card, or more conveniently by using the SWI versions of the software calls.

If direct access is required to the devices on the ADC-1208 then the processor must enter a supervisor mode before attempting to read or write to a location on the expansion card: The addresses of the devices can be found in the section 'The memory structure of the ADC-1208': THESE ADDRESSES ARE OFFSETS FROM THE BASE ADDRESS OF THE EXPANSION CARD WHICH DEPENDS UPON THE BACKPLANE SLOT CONTAINING THE CARD.

HOW TO CREATE A SIMPLE PROGRAM

To make a simple program that will control the ADC-1208 and capture a set number of samples from the analogue input lines a few decisions should be made before attempting to produce the code:

First the user should decide on the computer language which best suits their application: For simple interactive work BASIC is adequate for most needs and the star (*) commands provide a user-friendly method of accessing the ADC-1208: If high speed signal processing of analogue data is required, then the language C provides much more support for user applications: In this language the user is advised to use SWI calls to control the ADC-1208 rather than embedded machine code: If extremely high speed is required then the user will have to resort to machine code:

When the user is ready to program the ADC-1208 a few preliminary steps should be run through:-

1) How many samples will I need? Is the default area (8192 samples) adequate? If not then you need to claim an area of memory for the storage of the ADC data, the ADC software needs to be informed of this by using the *ADCDATA command or by stating the destination in the command tail of an acquisition command.

2) How many channels? The default is one. If you require more than one, then is the data area large enough? It should be no_points*no_channels*4 bytes long: Use Nochannels or the command tail of an acquisition command to set the no of channels.

3) How many samples per second do I need? Failure to set the clock is the commonest mistake. If the dock is zero then the computer will hang up when an acquisition command is initiated. Use *clock or its SWI call to set the sampling rate. The sampling rate can be set as a parameter in the OSCILLO command.

4) Do I need triggered or synchronised acquisition? Use the *trigger command or its SWI call or set it as a parameter in the command tail of an acquisition command:

5) Do I want to have real time display of the results of data acquisition? If the sampling rate is low (typically below 2000 samples per sec per channel) then you can use OSCILLO or its equivalent, but for high rates the background fast interrupt method of acquisition must be used:

6) Do I require extremely high data acquisition sampling rates? If so, the *ADC command must be used to achieve high sampling rates. The highest speeds can only be achieved with single channel modes.

7) Do I want to perform other tasks, such as disc storage, signal processing, mathematical manipulation, text editing etc, whilst the ADC is collecting data? The only way to achieve this is to use the *CONTADC command or its SWI equivalent: The RISC-OS environment should be used to achieve true multi-tasking or great care should be exercised in the allocation of memory.

8) Do I want to manipulate the data correctly? Remember that the results of analogue to digital conversion are stored in offset binary format, they need to be converted to two's complement binary for correct internal use. This can achieved by subtracting 2048 from each data sample,.

INSTALLATION OF THE ADC-1208

Full use is made of the latest low-power fast HC and AC CMOS chip technologies giving low power consumption, high speed and reliability of design: However, the use of these devices means that great care should be taken in handling the card as static electricity can damage them: Although damage due to static discharge is unlikely on soldered cards it is not impossible: It is therefore recommended that full handling precautions should be taken when fitting the ADC-1208:

AVOID TOUCHING CIRCUIT COMPONENTS AND HOLD THE EXPANSION CARD ALONG ITS SIDES AND BY THE REAR PANEL ASSEMBLY ONLY.

To install the ADC-1208 expansion card into your Archimedes computer you should proceed as follows :-

1. Disconnect your computer from the mains supply, ensuring that the ON/OFF switch is in the 'OFF' position. Disconnect any peripherals which may be attached (e:g: printer, monitor etc:).

2. Unscrew the five case fixing screws to enable the removal of the top casing of the computer: (Three are located along the top edge at the rear of the computer and one on each side towards the front):

3. Remove the computer casing by carefully sliding it backwards until completely clear of the rest of the computer.

4. Remove a single blanking panel from the rear of the computer, ensuring that this position corresponds to a spare socket on the expansion card backplane. NOTE - If a backplane is not fitted then this should be done before your ADC-1208 can be installed - contact your local Acorn dealer:

5. The expansion card can now be plugged into the backplane. It is necessary to insert the card through the space left by the blanking panel at the rear of the machine: The ADC-1208 should be carefully but firmly pushed into the socket on the backplane and secured to the rear of the machine using the two fixing screws and washers used on the Archimedes blanking panel. With some early Archimedes machines the expansion card will protrude out at the rear of the computer by approximately 2mm. If this is the case then the two metal spacers supplied with the expansion card should be fitted between the back panel and the computer.

6. Replace the lid of your computer. The machine may now be re-connected to the mains supply and its peripheral devices.

CONNECTING EXTERNAL SIGNAL LINES TO THE ADC1208

The rear panel of the ADC-1208 has two sockets. The first is for analogue input and the second is for digital input/output. The pin-outs on these connectors are :-

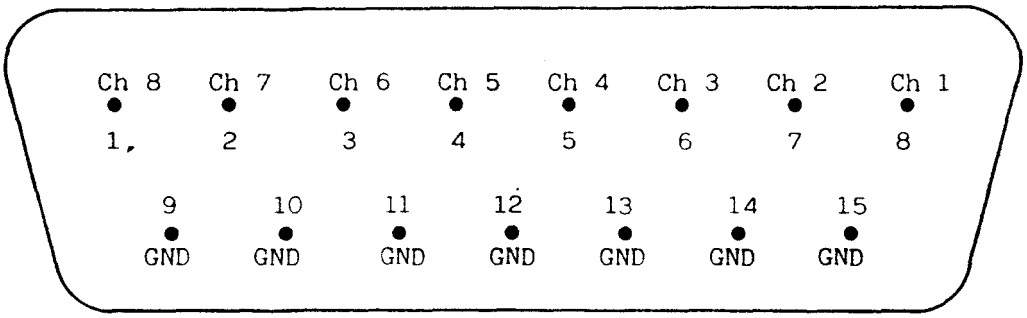
ANALOGUE IN **15 PIN D CONNECTOR**

pin	connection
1	CHANNEL 8
2	CHANNEL 7
3	CHANNEL 6
4	CHANNEL 5
5	CHANNEL 4
6	CHANNEL 3
7	CHANNEL 2
8	CHANNEL 1
9	GROUND
10	GROUND
11	GROUND
12	GROUND
13	GROUND
14	GROUND
15	GROUND

DIGITAL INPUT/OUTPUT 20 WAY IDC CONNECTOR

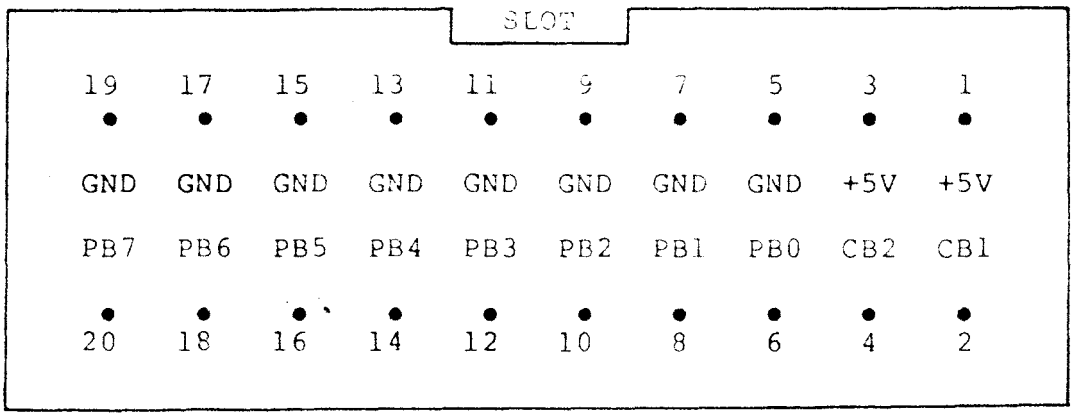
pin	connection
1	+5 volts
2	CB1
3	+5 volts
4	CB2
5	GROUND
6	PB0
7	GROUND
8	PB1
9	GROUND
10	PB2
11	GROUND
12	PB3
13	GROUND
14	PB4
15	GROUND
16	PB5
17	GROUND
18	PB6
19	GROUND
20	PB7

The digital input/output connector is pin compatible with the BBC micro series User port:



(Viewed looking into the socket on the back panel.)

'ANALOGUE IN' 15 WAY D-TYPE CONNECTOR PIN -OUT



(Viewed looking into the socket on the back panel.)

'DIGITAL INPUT/OUTPUT' USERS' PORT PIN-OUT

CALIBRATION

The ADC-1208 is fully calibrated at the factory: However, there is one factor that cannot be set in the factory: That is the precise clock frequency used by the computer it is going to be installed in: II you require extremely accurately timed measurements you should either use a known external clock or measure the exact frequency of the 2MHz clock on the backplane of your computer: This value should be placed in word 20 from the start of the module workspace after every power-on or hard-reset: Use SWI(WVAIO_GetAdcInfo) to find the start of the module workspace.

APPENDIX 1:THE VERSATILE INTERFACE ADAPTOR REGISTERS

The VIA is a standard 6522 device. It incorporates two 8 bit bidirectional I/O ports, one of which is made available on the external connector and the other is used to control the multiplexer: The Via contains two 16 bit programmable timer/counters, a serial data port and also has two CMOS-compatible peripheral control lines. The registers within the VIA and their use on the ADC1208 expansion card are :-

register	**description**	
	read	**write**
0	Input register B	Output register B
1	no meaning	Multiplexer channel selection
2	Data direction register B	
3	Multiplexer control - must be set to 255	
4		Low byte of current clock
5		High byte of current clock
6	low byte of clock	low byte of clock latch
7	high byte of clock	high byte of clock latch
8	timer 2 low byte latch	timer 2 low order counter
9	timer 2 high byte counter	
10	shift register	
11	auxiliary control register	
12	peripheral control register	
13	interrupt flag register	
14	interrupt enable register	
15	multiplexer channel selection	

The full details of the 6522 can be found in a data sheet for this particular device:

APPENDIX 2 : TRIGGERING THE ADC1208 AND SYNCHRONISATION SIGNALS

There are three commands in the ADC-1208 software module that use triggering or synchronisation protocols: These are the *ADC, *Oscillo and the *CONTADC command, and they all have the same trigger codes.

For these commands the trigger word in the command tail of the command has the following meaning :-

Trigger word = 1	no triggering
Trigger word = 2	a change in logic on CB1 on the digital I/O bus is required to initiate the acquisition of a block of data:
Trigger word = 3	a change in logic on CB2 on the digital I/O bus is required to initiate the acquisition of a block of data
Trigger word = 4	the logic level on PB0 must be high to initiate the acquisition of a block of data
Trigger word = 5	the logic level on PB1 must be high to initiate the acquisition of a block of data
Trigger word = 6	a pulse appears on CB2 at the start of acquisition
Trigger word = 7	a pulse appears on PB0 at the start of acquisition

It should be noted that the *VIAWRITE command or its SWI equivalent "WVAIO_ViaWrite" should be used to set the particular conditions necessary to allow the digital I/O bus to be used for trigger signals to be read or generated. For example, if CB2 is to be the trigger source then register 12 in the VIA can be used to determine the type of signal on CB2 which will effect a triggered acquisition. e.g.

***VIAWRITE 12 &60**
would result in data acquisition starting only after a positive edge occurring on the CB2 line:

***VIAWRITE 12 &20**
would result in data acquisition starting only after a negative edge occurring on the CB2 line:

***VIAWRITE 12 &10**
would result in data acquisition starting only after a positive edge on the CB1 line.

***VIAWRITE 12 &0**
would result in data acquisition starting only after a negative edge on the CB1 line:

If the PB0 or PB1 line are used in triggering or for the generation of a sync pulse then the data direction register 2 should be set to correspond to either input (for trigger mode) or to output (for sync pulses). This must be done by the user.

APPENDIX 3 : USING AN EXTERNAL CLOCK

An external clock can only be used with the **ADC** and **CONTADC** commands:

If an external dock is required then a simple sequence of commands must be used before the external clock pulses are used:

The internal clock must be disabled using *CLOCKOFF:

The external dock (which must be TTL compatible) should be connected to the PB7 line of the digital I/O bus:

The PB7 line must be set to be input only by using *VIAWRITE: Set bit 7 in register 2 to zero.

All the ADC commands will now operate correctly if the parameter block is written to with the frequency of the external clock:

APPENDIX 4 : THE SWI CALLS

ENTRY AND EXIT PARAMETERS

LIST OF SWI CALLS AND THEIR SWI NUMBERS

WILD ADC1208 SWI BASE IS &80A80

SWI NUMBER IN HEX	SWI NAME
80A80	WVAIO_GetAdcInfo
80A81	WVAIO_ViaWrite
80A82	WVAIO_ViaRead
80A83	WVAIO_Clock
80A84	WVAIO_Channel
80A85	WVAIO_Trigger
80A86	WVAIO_No Channels
80A87	WVAIO_ADO
80A88	WVAIO_ContADC
80A89	WVAIO_ADCStart
80A8A	WVAIO_ADCStop
80A8B	WVAIO_Display
80A8C	WVAIO_Oscillo

ENTRY AND EXIT PARAMETERS

80A80 swi "WVAIO_GetADCInfo"
on entry
R0 = address to store pointer to adcinfo block if zero then result is printed to current output stream
on exit
R0 = address of adcinfo parameter block

80A81 swi "WVAIO_ViaWrite"
on entry
R0 = register within VIA for writing to
R1 = byte to be written
on exit
R0 = register written to
R1 = byte written

80A82 swi "WVAIO_ViaRead"
on entry
RO = register in VIA to be read
on exit
RO = register read from
R1 = byte read

80A83 swi "WVAIO_Clock"
on entry
R0 = sampling rate in Hertz. If zero then current clock rate sent to current output stream
on exit
R0 = sampling rate used by VIA

80A84 swi "WVAIO_Channel"
on entry
R0 = Input channel for subsequent single channel acquisition If zero then current channel sent to current output stream on exit
R0 = Current input channel

80A85 swi "WVAIO_Trigger"
on entry
R0 = Trigger mode for subsequent acquisition
If zero then current trigger mode sent to current output stream on
exit
R0 = Current trigger mode

80A86 swi "WVAIO_No_Channels"
on entry
R0 = Number of channels to be used in subsequent acquisitions.
If zero then current no of channels printed on current output stream on
exit
R0 = Current no of channels

80A87 swi "WVAIO_ADC"
on entry
R0 = Number of data samples to be acquired
R1 = Number of channels to be used, starting at channel 1
R2 = pointer to adc data storage area
R3 = trigger word
on exit
R0 = Number of data samples acquired
R1 = Number of channels used
R2 = pointer to data area used to store results
R3 = trigger mode used

80A88 swi "WVAIO_ContADC"
on entry
R0 = Number of data samples to be acquired
R1 = Number of channels to be used, starting at channel 1
R2 = pointer to adc data storage area
R3 = trigger word
R4 = mode
on exit
-

80A89 swi "WVAIO_ADCStart"
on entry
-
on exit
-

80A8A swi "WVAIO_ADCStop"
on entry
-
on exit
-

80A8B swi "WVAIO_Display"
on entry
R0 = Number of points to be displayed
R1 = Number of channels to be displayed
R2 = vertical scaling multiplier
R3 = start address of data area to be displayed on
exit
R0 = Number of points displayed
R1 = Number of channels used
R2 = current vertical scaling multiplier
R3 = pointer to data area displayed

80A8C swi "WVAIO_Oscillo"
on entry
R0 = Number of points to be acquired
R1 = Number of channels to be used, starting at channel 1
R2 = vertical scaling multiplier to be used in display
R3 = sampling rate to be used during acquisition
R4 = pointer to adc data storage area
R5 = trigger word
on exit
R0 = Number of samples acquired
R1 = Number of channels used
R2 = current vertical scaling multiplier
R3 = Current sampling rate
R4 = pointer to data storage area
R5 = trigger mode used

ERRORS

The ADC-1208 software can handle a number of incorrect situations that can arise during the use of the ADC-1208 hardware: The error handlers will detect potentially incorrect use of the hardware and return an error message and error number: The use of SWI/SYS calls with an 'X' prefix, (for example SYS "XWVAIO_ViaRead",16) will suppress any error messages: However, the error will still have been handled correctly and the command will not have been executed fully.

The base error number is &802300:

ERROR NUMBER	ERROR TYPE
&802300	ADC-1208 ROM access error - failure to read ROM
&802301	ADC-1208 ROM not writable
&802302	Unable to claim workspace - RMA error
&802303	Unknown SWI/SYS call
&802304	Parameter out of range - too large
&802305	No clock - The acquisition of data requires a dock