I/O PODULE

Guide

Acorn

Archimedes

The product described in this guide and products for use with it are subject to continuous development and improvement.

All information of a technical nature and particulars of the product and its use (including the information and particulars in this guide) are given by Acorn Computers in good faith. However, it is acknowledged that there may be errors or omissions in this guide or in the products it describes. Acorn Computers welcomes comments and suggestions relating to the product and this guide.

All correspondence should be addressed to:

Customer Support and Training,
Acorn Computers Limited,
Cambridge Technopark,
645 Newmarket Road,
Cambridge CB5 8PB.

All maintenance and service on the product must be carried out by Acorn Computers' authorised dealers. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service, maintenance or repair by unauthorised personnel. This guide is intended only to assist the reader in the use of this product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this guide, or any incorrect use of the product. Refer to the explicit instructions for installation.

First published 1987
Issue 1 August 1987
Published by Acorn Computers Limited
Part number 0476,200

CONTENTS

1 INTRODUCTION

The I/O Expansion Podule allows some existing BBC peripherals to be connected to the Archimedes Microcomputer. It is not suitable for the Teletext Adapter, Acorn 1MHz bus, Winchester Drive or IEEE 488 Interface due to software limitations.

The Input/Output interfaces provided are the 1MHz Bus, the User Port, and the Analogue to Digital Converter. All three interfaces use the same connectors as on existing BBC Microcomputers with the same pin-outs. However, due to the new processor and the higher speed of the Archimedes Microcomputer there are some incompatibilities between the two which are outlined below.

The following descriptions should be read in conjunction with the circuit diagram at the back of this guide.

1MHz BUS

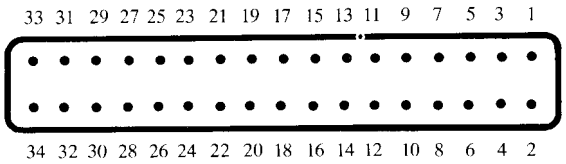
General

The 1MHz bus is a memory mapped I/O port which gives peripherals direct access to the BBC Microcomputer's 6502 data bus. It is accessed through two 256-byte pages (known as FRED and JIM) in the memory map, so the bottom eight address lines are fed to the 1MHz bus connector to give the offset in each page, and the page itself is determined by one of two control signals being active. These control signals are NotPageFC and NotPageFD, so named because these are the pages into which the 1MHz bus is mapped in the BBC Microcomputer. On the Archimedes, it is better to think of these two signals as FRED and JIM respectively.

On the BBC Microcomputer and the Archimedes, legal access is made to the 1MHz Bus by the use of four OSBYTE calls, numbers 146 to 149.

Archimedes implementation

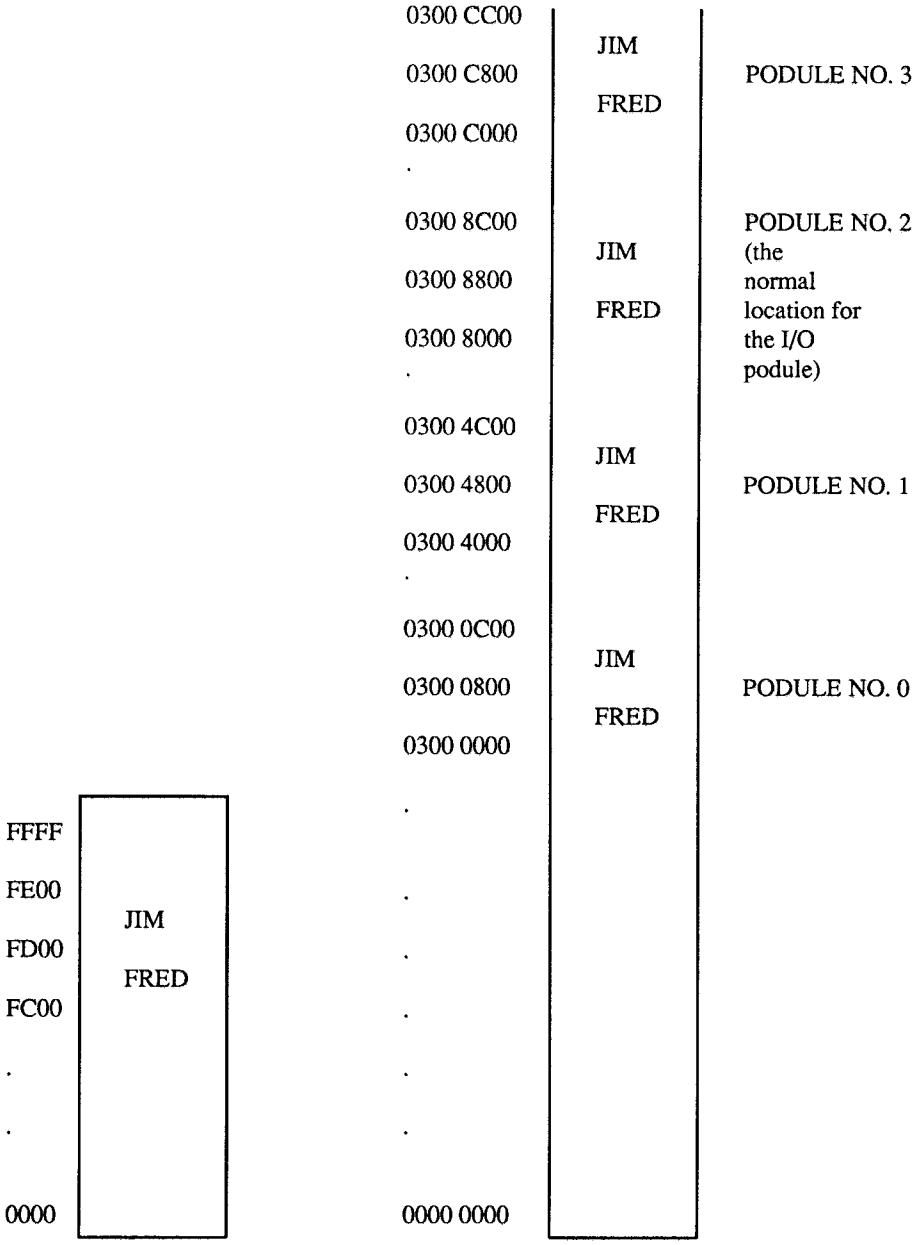
The pin-out of the 1MHz Bus 34-pin IDC connector is shown overleaf.



1 - 0v	2 - R/w	19 - D1	20 - D2
3 - 0v	4 - 1MHz	21 - D3	22 - D4
5 - 0v	6 - NM1	23 - D5	24 - D6
7 - 0v	8 - IRQ	25 - D7	26 - 0v
9 - 0v	10 - FC	27 - A0	28 - A1
11 - 0v	12 - FD	29 - A2	30 - A3
13 - 0v	14 - RST	31 - A4	32 - A5
15 - 0v	16 - NC	33 - A6	34 - A7
17 - 0v	18 - D0		

On the I/O Expansion Podule, the 1MHz bus is implemented in such a way that it can be accessed via the legal OSBYTE calls to the Arthur Operating System which read or write bytes to FRED and JIM. However, it cannot be assumed that FRED and JIM are mapped into memory at a specific location, so direct access to the 1MHz bus through writing to and reading from specific addresses does not work. This is true of all the interfaces provided by the I/O Podule. Of course, the two pages FC and FD into which the 1MHz bus is mapped on the BBC Microcomputer have no meaning in the context of the Archimedes implementation. Any code which peeks or pokes these locations does not work. It is better to forget about the specific locations and think only of the two memory areas FRED and JIM in which the 1MHz bus is mapped. The diagram on the following page shows the memory map of the Model B BBC Microcomputer compared with that of the Archimedes.

Memory maps of BBC Model B and Archimedes showing FRED and JIM



The diagram shows that the position of FRED and JIM in the BBC Microcomputer's memory map is fixed, whereas in the Archimedes the addresses of FRED and JIM depend on the Podule slot into which the Podule is plugged.

1MHz Bus signals

NotPageFC and NotPageFD tell the 1MHz Bus peripheral that the computer is reading or writing a byte to the 1 MHz bus and that there is a valid offset on the address bus. Of course, in the Archimedes, the memory addresses will not be pages FC or FD, but some other convenient locations. Think of these signals as FRED and JIM.

The Archimedes address lines A2 to A9 are available to the 1MHz bus on pins 27 to 34 of the connector. As the Archimedes is a 32-bit machine and the lMHz bus uses only the least significant byte of each 32-bit word, the lowest address line is A2 and not AO as on the BBC Microcomputers.

The low 8 bits of the data bus DO to D7 are fed to the 1MHz bus on pins 18 to 25 of the connector. The Read/Write line determines the data direction.

Incompatibilities between the BBC 1MHz Bus and the Archimedes implementation

These are as follows:-

- The interrupt inputs IRQ amd NMI are not normally supported. These interrupts may be connected to the Archimedes interrupt system by means of a jumper option marked PL4 on the circuit diagram. This gives the option of connecting to either of two interrupts, IRQ or FIQ. The Archimedes receives the interrupt when enabled via CA2 of the 6522. The Archimedes can also read these interrupts from DO and D2 respectively of the data bus to determine which Podule caused the interrupt.

- The analogue input to the sound circuit of the BBC Microcomputer is not supported.

USER PORT

General

The User Port consists of 8 data lines and two control lines from half of a 6522 Versatile Interface Adapter chip (VIA). The VIA contains 16 internal registers and these are mapped into memory.

On the BBC Microcomputer and on the Archimedes, legal access to these registers is made by using the two OSBYTE calls which read and write to SHEILA, numbers 150 and 151.

The signals available on the connector are the 8 data lines PBO to PB7 on pins 6, 8, 10, 12, 14, 16, 18 and 20 respectively, and the two interrupt/handshake/shift register control lines CB 1 and CB2 on pins 2 and 4 respectively.

When used for data transfer using handshaking, the CB2 signal is a `data ready' output to the peripheral, and the CB 1 signal is a `data taken' input from the peripheral.

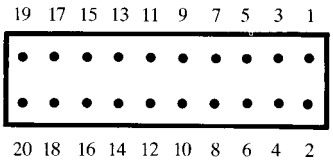
When used in interrupt mode, CBI and CB2 cause the IRQ line of the VIA to go low. However, interrupts from the VIA are not normally supported. See following page.

Serial data can be shifted into or out of the CB2 pin under control of either an internal timer or from an external clock applied to CB 1.

Archimedes implementation

The pin-out of the User Port 20-pin IDC connector is shown below.

User Port connector pin-out



1 - +5v	2 - CB1	11 - 0v	12 - PB3
3 - +5v	4 - CB2	13 - 0v	14 - PB4
5 - 0v	6 - PB0	15 - 0v	16 - PB5
7 - 0v	8 - PB1	17 - 0v	18 - PB6
9 - 0v	10 - PB2	19 - 0v	20 - PB7

The User Port is implemented as on the BBC Microcomputer using half of a 6522 VIA chip. As on the BBC Microcomputer, the VIA registers are memory mapped, and control is exercised in the same way through OSBYTE calls 150 and 151 which read from and write to the I/O page SHEILA. It cannot be assumed that SHEILA is mapped into memory at a specific location, so direct access to the User Port through writing to or reading from specific addresses does not work.

Incompatibilities between the BBC User Port and the Archimedes implementation

These are as follows:-

- The VIA chip on the expansion podule is running at 2MHz instead of the BBC Microcomputer's 1MHz device. This means that the internal timers of the VIA are running twice as quickly as expected. If the shift registers are being used under control of the internal timer then these too run twice as fast.
- Power which may be taken from the User Port must not exceed the difference between that allocated to the podule slot and that taken by the podule. See the section entitled Specification and timing.
- The interrupt signal from the VIA IC3 pin 21 is not normally supported. It may be connected to the Archimedes interrupt system by means of a jumper option marked PL4 on the circuit diagram. See the section on 1MHz Bus interrupts above for more details.

ANALOGUE TO DIGITAL CONVERTER

General

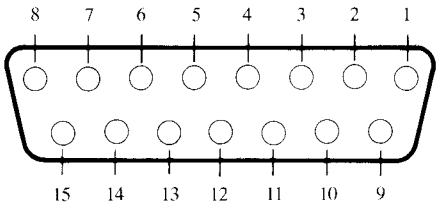
The Analogue to Digital Converter IC is a 10-bit integrating converter, but in this implementation the accuracy can be relied on only to 8 bits. Its output can be between 0 and 65520, but only 8 bits are significant so accuracy is to the nearest multiple of 256. Its two voltage references are OV and Vref. OV corresponds to 0 and Vref corresponds to 65520, so any applied voltage between OV and Vref will generate a number in direct proportion.

On the BBC Microcomputer and the Archimedes, legal access is made to the ADC either by using the BASIC keyword ADVAL, or using the OSBYTE calls 16 17 128 188 189 and 190. Access to the registers can also be gained using the two OSBYTE calls which read and write to SHEILA, numbers 150 and 151.

Archimedes implementation

The pin-out of the ADC 15-way D-type connector is shown opposite.

ADC **connector pin-out**



1 - +5	5 - AGND	9 - LPSTB	13 - FIRE0
2 - AGND	6 - AGND	10 - FIRE1	14 - VREF
3 - AGND	7 - CH1	11 - VREF	15 - CH0
4 - CH3	8 - AGND	12 - CH2	

The ADC is implemented using the same IC as in the BBC Microcomputer, the uPD7002.

The ADC can be accessed using both the legal methods of ADVAL from BASIC and OSBYTE calls.

Incompatibilities between the BBC ADC **and the Archimedes implementation**

None, if legal software is used.

MUSICAL INSTRUMENTS DIGITAL INTERFACE

This interface is available separately, and consists of another small board which connects to the I/O Expansion Podule and some ICs which plug into empty sockets on the I/O Podule board. Full instructions are supplied with the interface.

2 INSTALLATION

For installation of the Podule in the Archimedes Microcomputer see the separate Podule installation leaflet.

3 USING THE INTERFACES

LEGAL COMMANDS

All three interfaces must be used through the legal BASIC and Machine Operating System commands. Any software which tries to access specific memory locations in the earlier BBC Microcomputer I/O space will not work.

BASIC keyword ADVAL works in the same way on both the BBC Microcomputer and the Archimedes.

OSBYTE calls, however, use the 6502 registers on the BBC Microcomputer and so are implemented slightly differently on the Archimedes.

In general, parameters passed in A on the BBC Microcomputer are passed in the least significant byte of RO on the Archimedes. Those passed in X are now passed in the LSB of R1, and those passed in Y are now passed in the LSB of R2.

*FX commands still work as on the BBC Microcomputer, the parameters being passed in the correct registers automatically.

The legal commands are as follows.

1MHz Bus

OSBYTE 146 Read a byte from FRED
OSBYTE 147 Write a byte to FRED
OSBYTE 148 Read a byte from JIM
OSBYTE 149 Write a byte to JIM

On entry:
RO contains the OSBYTE number
R1 contains the offset in either FRED or JIM
R2 contains the byte to be written (for the two write commands).

On exit:
R2 contains the byte which was read (for the two read commands).

User Port

OSBYTE 150 Read a byte from SHEILA
OSBYTE 151 Write a byte to SHEILA

The 16 VIA registers which are memory mapped to the SHEILA I/O space have offsets &60 to &6F hex (96 to 111 decimal).

On entry:
RO contains the OSBYTE number
R 1 contains the offset in SHEILA
R2 contains the byte to be written (for the write command).

On exit:
R2 contains the byte which was read (for the read command).

Analogue to Digital Converter

Legal commands are as follows:-

- ADVAL Read the ADC channel value

  ADVAL is a BASIC function which takes a single parameter, the channel number (0 to 4).

  If the parameter is 0, ADVAL returns a 2-byte number. The low byte will give the status of the two `fire buttons` as follows.

Button	Status
0	No buttons pressed
1	Left side fire button pressed
2	Right side fire button pressed
3	Both fire buttons pressed

  If the parameter is between 1 and 4, ADVAL returns a 2-byte number which is the value of that ADC channel. This value is in the range 0 to 65520 in steps of 16 (in 12-bit mode) and steps of 256 (in 8-bit mode). However, accuracy is only to the nearest multiple of 256 in either mode, because in this implementation only the high byte is guaranteed accurate.

- OSBYTE 16 Select ADC channels which are to be sampled

  On entry:
  RO contains 16
  R1 contains the number of channels to be sampled (0 to 4)
  If R1 contains 0 then sampling is disabled

- OSBYTE 17 Force ADC conversion

  On entry:
  RO contains 17
  R1 contains the channel number to be forced (0 to 4)
  If R1 contains 0 then no conversion is forced

- OSBYTE 128 Read ADC channel value and fire button status

  On entry:
  RO contains 128
  RI contains channel number to be read (0 to 4)

  On exit:
  If R1 contained 0 on entry then the two lowest bits (bits 0 and 1) of R1 indicate the status of the "fire buttons", and R2 contains the number of the channel which was last used for ADC conversion, or 0 if no conversion has been completed. If Rl contained 1 to 4 on entry then RI (low) and R2 (high) contain the 16-bit value for that channel.

- OSBYTE 188 Read current ADC channel

  On entry:
  RO contains 188

  On exit:
  Rl contains the current ADC channel number

- OSBYTE 189 Read maximum ADC channel number

  On entry:
  RO contains 189

  On exit:
  R1 contains the maximum channel number to be used (0 to 4)
  This maximum number is set by OSBYTE 16

- OSBYTE 190 Read whether 12-bit or 8-bit conversion

  On entry:
  RO contains 190

  On exit:
  If R1 contains 0 or 12 then the conversion is 12-bit
  If Rl contains 8 then the conversion is 8-bit

  Note that conversion is guaranteed only to 8 bits due to the implementation.

EXAMPLES

MHz Bus

Use of the lMHz Bus consists solely of reading from and writing to the two 1/O memory areas called FRED and JIM. Each memory area contains 256 locations, and each read or write to a location is a read or write direct to the peripheral.

From BASIC use the SYS command.

```
10 REM read a byte from offset 200 in JIM
20 osbyte%=6 :REM SYS 6 is equivalent to OSBYTE
30 readbyte%=148 :REM osbyte number for Read byte from JIM
40 offset%=200 :REM offset in JIM from which byte is to be read
50 result%=0 :REM place to put result
60 SYS osbyte%,readbyte%,offset%,result% TO „result%
70 PRINT result%
```

The above example will, on entry, place the contents of readbyte% in R0, the contents of offset% in Rl, and the contents of result% in R2. On exit, the byte which is read will be placed in result%.

```
10 REM write a byte to offset &OB in FRED
20 writebyte%=147 :REM osbyte number for Write byte to FRED
30 offset%=&OB :REM offset in FRED to which byte is to be written
40 byte%=236 :REM byte to be written
50 SYS "OS Byte",writebyte%,offset%,byte%
```

The above example will write 236 to offset &OB in FRED.

Note that we have used a string "OS Byte" in the SYS command instead of defining the variable osbyte%. Either of these methods is legal.

A **very simple way to achieve the same result would** be

For the same example in assembly language use the SWI instruction.

```
 10 REM write a byte to offset &0B in FRED
 20 osbyte%=6 :REM SWI 6 is equivalent to OSBYTE
 30 writebyte%=147 :REM osbyte number for Write byte to FRED
 40 offset%=&0B :REM offset in FRED to which byte is to be written
 50 byte%=236 :REM byte to be written
 60 DIM code% 100
 70 P%=code%
 80 [
 90 STMFD R13!,{R0-Rl2,R14} \ save registers on stack
100 MOV R0,#writebyte%      \ put osbyte number in R0
110 MOV R1,#offset%         \ put offset in R1
120 MOV R2,#byte%           \ put byte to be written in R2
130 SWI osbyte%             \ execute osbyte call
140 LDMFD R13!,{R0-R12,PC} \ pull registers from stack
                            \ and return to BASIC
150 ]
160 CALL code%
```

In the above example, all the BASIC variable assignments are the same.

In the assembly language section, line 110 puts the osbyte number in RO, line 120 puts the offset in Rl, and line 130 puts the byte to be written in R2.

The SWI instruction in line 140 executes the osbyte call which writes the byte to the 1MHz Bus.

Line 180 calls the routine.

The instruction in line 100 saves the contents of 14 of the 16 registers RO to R12 and R14 on the stack. This is because the routine may corrupt some of these registers and they must be restored before returning to BASIC. R14 is the link register, and the contents of this register form the address to get back to BASIC. This is done by putting the contents of R14 into R15, the program counter.

Line 150 pulls the saved contents of the registers from the stack prior to returning to BASIC R0 to R12 are restored to their original values. R15, the program counter (PC), is loaded with the original contents of R14, the link register, which gives the address of the next BASIC operation.

User Port

The User Port is controlled via the 16 registers of the VIA chip which are mapped into the 1/O space SHEILA at offsets &60 to &6F.

For example, to write &FF to DDRB (data direction register B).

```
 10 osbyte%=6 :REM SYS 6 is equivalent to osbyte
 20 writebyte%=151 :REM osbyte number for write byte to SHEILA
 30 offset%=&62 :REM offset in SHEILA of DDRB
 40 byte%=&FF :REM byte to put in DDRB
 50 SYS osbyte%,writebyte%,offset%,byte%
```

The same example in assembly language is shown below.

```
 10 REM write &FF to User Port DDRB
 20 osbyte%=6 :REM SWI 6 is equivalent to OSBYTE
 30 writebyte%=151 :REM osbyte number for Write byte to SHEILA
 40 offset%=&62 :REM offset in SHEILA of DDRB
 50 byte%=&FF :REM byte to put in DDRB
 60 DIM code% 100
 70 P%=code%
 80 f
 90 STMFD R13!,{R0-R12,R14} \ save registers on stack
100 MOV R0,#writebyte%      \ put osbyte number in R0
110 MOV R1,#offset%         \ put offset in R1
120 MOV R2,#byte%           \ put byte to be written in R2
130 SWI osbyte%             \ execute osbyte call
140 LDMFD R13!,{R0-R12,PC}  \ pull registers from stack
                            \ and return to BASIC
150
160 CALL code%
```

The BASIC keyword ADVAL takes a parameter which is the ADC channel number.

The ADVAL function performs an OSBYTE 128 call, reading the value on the specified channel.

```
 10 REM read value of ADC channel 2
 20 REM read fire button status and last channel used
 30 REM using ADVAL
 40 :
 50 AtoD%=ADVAL(2) :REM convert on channel 2 and read result
 60 firebutton%=ADVAL(0) :REM fire buttons and channel status
 70 PRINT AtoD% :REM print channel 2 ADC value
 80 PRINT firebutton% AND 3 :REM print fire button status
 90 channel$=firebutton% DIV 256 :REM shift right channel status
100 PRINT channel% :REM and print it
```

In the above example, line 10 reads the value of ADC channel 2 into the BASIC variable AtoD%. Line 20 reads the fire button status and the ADC channel last used. Lines 30 to 60 print out the three pieces of information: the ADC channel 2 value, the fire button status, and the last channel to perform a conversion (which will be 2).

Below is the same example performed using OSBYTE call 128 and the BASIC SYS command.

```
 10 REM read value of ADC channel 2
 20 REM read fire button status and last channel used
 30 REM using OSBYTE calls via SYS
 40
 50 osbyte%=6 :REM SYS 6  is equivalent to osbyte
 60 readADC%=128 :REM osbyte number for read ADC value
 70 channel$=2 :REM ADC channel required
 80 status$=0 :REM parameter 0 to read status
 90 SYS osbyte%,readADC%,channel% TO ,low%,high%
100 SYS osbyte%,readADC%,status% TO ,firebutton%,channel%
110 AtoD%=low%+high%
120 PRINT AtoD%
130 PRINT firebutton%
140 PRINT status%
```

Line 90 reads ADC channel 2. Line 100 reads the status of the two fire buttons and the last channel used.

General

There is a software interrupt SWI instruction which returns the absolute location of the 1/O Podule in the memory map.

The SWI argument can be either name or number.

```
SWI "IO PODULE HARDWARE"
```

or

```
SWI &403C0
```

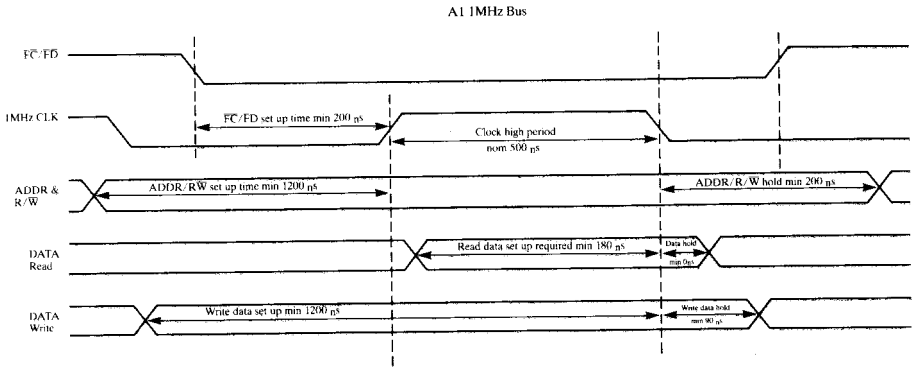
On exit, R1 contains the base address of the podule hardware. All other registers are preserved.

4 SPECIFICATION AND TIMING

TECHNICAL SPECIFICATION

	Min	Typ	Max	Units
Podule				
Operating voltage	4.5	5.0	5.5	V
Supply current to Podule		220	500	mA
	+ current supplied from User Port			
Analogue to Digital Converter				
Input voltage range (Vref typically 1.8V)	0		Vref	V
Accuracy	8		10	bits
Conversion time	8.5		15	ms
Input impedance		1000		Mohm
User Port				
Output drive capability			1	TTL i/p
Input load			1	TTL i/p
Output current (+5V)			500	mA
1MHz Bus				
Output drive capability			1	TTL i/p
Input load			1	TTL i/p

1MHz Bus timing diagram



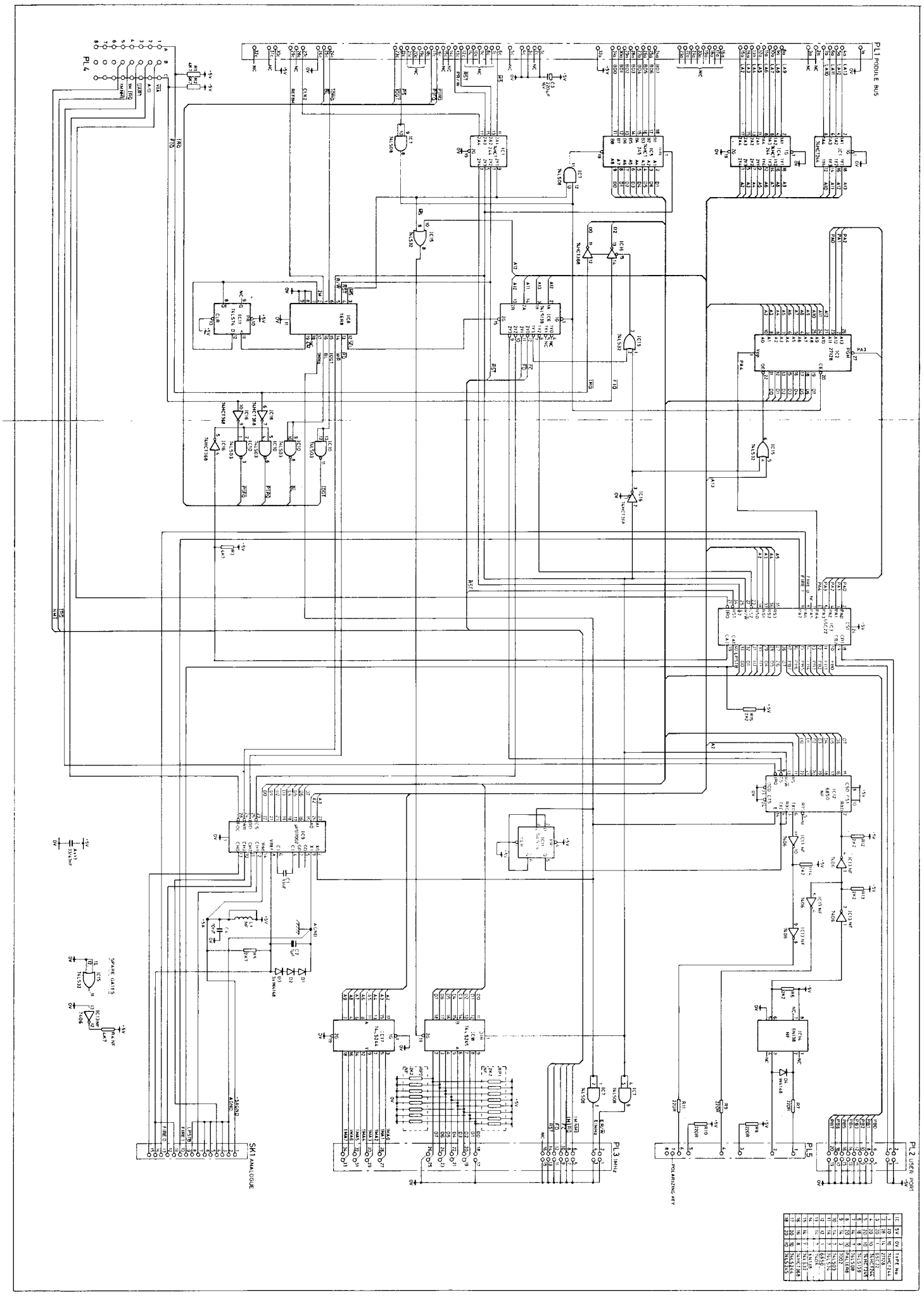
5 APPENDIX

I/O EXPANSION PODULE ADDRESS ALLOCATION

Slot	Device	Address space
0	1MHz Bus FRED	0300 0000 to 0300 03FF step 4 (256 bytes)
	1MHz Bus JIM	0300 0800 to 0300 OBFF step 4 (256 bytes)
	ADC	0300 1000 to 0300 1000 step 4 (4 bytes)
	ROM IC2	033C 0000 to 033C 1FFF step 4 (2 Kbytes)
	User Port (VIA)	033C 2000 to 033C 203C step 4 (16 bytes)
1	1MHz Bus FRED	0300 4000 to 0300 43FF step 4 (256 bytes)
	1MHz Bus JIM	0300 4800 to 0300 4BFF step 4 (256 bytes)
	ADC	0300 5000 to 0300 500C step 4 (4 bytes)
	ROM IC2	033C 4000 to 033C 5FFF step 4 (2 Kbytes)
	User Port (VIA)	033C 6000 to 033C 603C step 4 (16 bytes)
2	1MHz Bus FRED	0300 8000 to 0300 83FF step 4 (256 bytes)
	1MHz Bus JIM	0300 8800 to 0300 8BFF step 4 (256 bytes)
	ADC	0300 9000 to 0300 9000 step 4 (4 bytes)
	ROM IC2	033C 8000 to 033C 9FFF step 4 (2 Kbytes)
	User Port (VIA)	033C A000 to 033C A03C step 4 (16 bytes)
3	1MHz Bus FRED	0300 C000 to 0300 C3FF step 4 (256 bytes)
	1MHz Bus JIM	0300 C800 to 0300 CBFF step 4 (256 bytes)
	ADC	0300 D000 to 0300 DOOC step 4 (4 bytes)
	ROM IC2	033C 0000 to 033C DFFF step 4 (2 Kbytes)
	User Port (VIA)	033C E000 to 033C E03C step 4 (16 bytes)

Note: these addresses are given only as guides and software should not access these locations directly. Always use the legal OSBYTE calls to access the I/O ports.

I/O EXPANSION PODULE CIRCUIT DIAGRAM



REFERENCES

Archimedes Backplane Installation Guide (Acorn)

1 MHz Bus Application Note (Acorn)

6522 VIA Data Sheet (Rockwell)

uP7002 ADC Data Sheet (NEC)

INDEX