DAS-4

DAS-4

Part Number: 24888

Revision A

Last Edit: July, 1987

WARRANTY INFORMATION

All products manufactured by Keithley MetraByte are warranted against defective materials and worksmanship for a period of one year from the date of delivery to the original purchaser. Any product that is found to be defective within the warranty period will, at the option of Keithley MetraByte, be repaired or replaced. This warranty does not apply to products damaged by improper use.

New Contact Information

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, OH 44139

Technical Support: 1-888-KEITHLEY
Monday – Friday 8:00 a.m. to 5:00 p.m (EST)
Fax: (440) 248-6168

Visit our website at http://www.keithley.com

Table of Contents

Section 1

INTRODUCTION

1.1 SUMMARY OF DAS-4 FUNCTIONS.

MetraByte's DAS-4 is a 8 channel 8 bit high speed A/D converter and digital I/O board for the IBM PC-PC/XT-PC/AT & PS2 Model 30[1] and other bus compatible computers e.g. A.T.& T. 6300 series, Zenith, Compaq, PC's Limited, Tandon, Televideo etc. The DAS-4 board is 5" long and can be fitted in a "half" slot. All connections are made through a standard 37 pin D male connector that projects through the rear of the computer. The following functions are implemented on the DAS-4:-

1. An 8 channel, 8 bit successive approximation A/D converter with sample/hold. The full scale input of each channel is +/-5 volts with a resolution of 0.03906 volts (39.06 millivolts). Inputs are single ended with a common ground and can withstand a continuous overload of +/-30 volts and brief transients of several hundred volts. All inputs are fail safe i.e. open circuit when the computer power is off. A/D conversion time is typically 20 microseconds (30 microseconds max.) and depending on the speed of the software driver, through puts of up to 30,000 channels/sec are attainable.

2. 7 bits of TTL digital I/O are provided composed of one output port of 4 bits and one input port of 3 bits.

3. 1 precision +5.00v (+/-0.2v) reference voltage output is derived from the A/D converter reference. This output can source/sink 5mA and is the voltage is slightly adjustable by the A/D Full Scale trim pot.

4. An external interrupt input is provided that can select any of the IBM P.C. interrupt levels 2-7 and allow user programmed interrupt routines to provide background data acquisition or interrupt driven control. The DAS-4 includes status and control registers that make interrupt handshaking a simple procedure. The interrupt input is positive edge triggered and may be connected to any external TTL compatible trigger source.

5. IBM P.C. bus power (+5, +12 & -12v) is provided along with all other I/O connections on the rear connector. This makes for simple addition of user designed interfaces, input signal conditioning circuits, expansion multiplexers etc.

The following utility software for DAS-4 is provided on a double sided 360K PC-DOS format 5-1/4" floppy disk compatible with DOS 2.0 and higher revisions (3-1/2" media available on request):-

1. A machine language I/O driver (DAS4.BIN) for control of A/D, and digital I/O channel functions via BASIC CALL. The I/O driver can select

---

1. Registered trademarks of International Business Machines Corporation.

multiplexer channels, set scan limits, perform software commanded A/D conversions, interrupt driven conversions and scans. The driver source listing, DAS4.ASM, is also provided on the disk.

2. Initial setup and installation aids.

3. Calibration and test programs.

4. Electronic strip chart program.

5. Slow speed data logging program.

6. Other examples and demonstration programs.

The DAS-4 is simple, inexpensive and easy to use. It is ideally suited to low precision data acquisition and control applications. 8 bits provides a span of 255 bits with -128 bits corresponding to -5 volts of input, and +127 corresponding to +4.96 volts. This provides better than 1% bipolar resolution which is often adequate for many measurement, control and graphing applications. The DAS-4 is hardware compatible with MetraByte's DAS-8 which provides higher resolution through the use of a 12 bit A/D.

To extend the capabilities of DAS-4 the following expansion modules can be connected via flat insulation displacement cable to the main 37 pin D I/O connector:-

1. SCREW TERMINAL CONNECTOR BOARD - All I/O lines on the rear connector are connected to miniature screw terminal connectors. The digital I/O port lines are monitored by L.E.D.'s and a small bread board area with +/-12v & +5v power is available for amplifiers, filters, and other user supplied circuits. The screw terminal connector board is MetraByte part number STA-08.

2. EXPANSION MULTIPLEXER AND INSTRUMENTATION AMPLIFIER - The EXP-16 multiplexes 16 differential inputs to a single analog output suitable for connection to any of the analog input channels of DAS-4. EXP-16 boards are cascadable so that up to 8 EXP-16 boards can be attached to a single DAS-4 providing a total of 128 channels. The expansion multiplexer board includes a low drift instrumentation amplifier with preselected switchable gains of 0.5, 1, 2, 10, 50, 100, 200 or 1000 (other gains can be resistor programmed). A cold junction compensation sensor is also included for software compensation of thermocouples which can be directly connected to EXP-16, although the DAS-4 resolution will not provide the fine resolution of a 12 bit A/D board when used for thermocouple measurements.

3. ISOLATION AMPLIFIER - MetraByte's model ISO-4 provides 4 isolation amplifier channels and can be connected directly to DAS-4. This is an excellent accessory for measuring off ground voltages (500v max. isolation) or eliminating ground loops or protecting input circuits. Up to 32 ISO-4 expanders may be connected to one DAS-4. The ISO-4 also includes cold junction compensation and can be used for thermocouple measurements. The ISO-4 provides gains ranging from 1 to 1000.

MetraByte also offers many optional software packages that can enhance the ease of use of the DAS-4. Their menu driven user interfaces eliminate programming and give fast results. Most include the capability of generating Lotus 1-2-3 compatible data files, immediate graphing of data and some also provide analytical capabilities as well. For a full description, see our catalog. All of our software that works with DAS-8 will also work with DAS-4 and includes LABTECH ACQUIRE, LABTECH NOTEBOOK, UNKELSCOPE, UNKELSCOPE JR., SNAPSHOT STORAGE SCOPE, TTOOLS and CTOOLS.

Section 2

INSTALLATION

2.1 BACKING UP THE DISK

The utility software supplied with DAS-4 is in DOS 2.0 (360K DSDD) format which is compatible with DOS 2.0 thru 3.3 revisions. If you need a 3-1/2" disk compatible with the PS2 Model 30 drives, please contact MetraByte, it is available at no charge. It is advisable to make a back up copy before using the software. For a direct back up, use the DOS DISKCOPY utility or alternatively COPY *.* to a pre-formatted disk. For a hard disk, simply use COPY *.* to transfer to a directory of your choice, the DAS-4 utility software is not copy protected. If for any reason you should misplace or damage the disk, please contact MetraByte for a free replacement.

2.2 HARDWARE INSTALLATION

DAS-4 utilizes 4 consecutive address locations in I/O space. Some I/O addresses will already be used by internal I/O and your other peripheral cards, so to avoid conflict with these devices, DAS-4's I/O address can be set by the BASE ADDRESS D.I.P. switch to be on a 4 bit boundary anywhere in the I.B.M. PC decoded I/O space. The PC and PC/XT's expansion I/O address space extends from decimal 512-1023 (Hex 200-3FF) which is much larger than is ever likely to be fully occupied.The PC AT's I/O address space is larger and extends from 256-1023 (Hex 100-3FF), some XT compatibles also follow this expanded I/O space capability. Such a large space also allows use of more than one DAS-4 in a single computer. The reserved I/O addresses for standard IBM devices are detailed below:-.

ADDRESS(Hex)	DEVICE	ADDRESS(Hex)	DEVICE
000-1FF	Internal system	378-37F	LPT1:
200-20F	Game	380-38C	SDLC comm.
210-217	Expansion unit	380-389	Binary comm. 2
220-24F	Reserved	3A0-3A9	Binary comm. 1
278-27F	Reserved	3B0-3BF	Mono dsp/LPT1:
2F0-2F7	LPT2:	3C0-3CF	Reserved
2F8-2FF	COM2:	3D0-3DF	Color graphics
300-31F	Prototype card	3E0-3E7	Reserved
320-32F	Hard disk	3F0-3F7	Floppy disk
		3F8-3FF	COM1:

This covers the standard IBM I/O options (most compatibles are identical), but if you have other I/O peripherals e.g. special hard disk drives, special graphics boards, prototype cards etc. they may be making use of I/O addresses not listed in the table above. Memory addressing is separate

from I/O addressing so there is no possible conflict with any add-on memory that may be in your computer. Usually, a good choices is to put the DAS-4 at base address Hex &H300 or &H310 (Decimal 768 or 784). (Note if you have an IBM prototype board plugged in, it makes use of the Hex 300-31F address space and would conflict, &H330 or &H340 would be a good alternative in this case). As an aid to setting the base address D.I.P. switch, a graphical switch position display program, INSTALL.EXE, can be run from the DOS prompt. To run the installation aid program, type:-

A>INSTALL

When you get the "Desired base address?" prompt, type in your choice in decimal or &H--- format and press return. The program will round your address to the nearest 4 bit boundary, check for possible conflicts with standard IBM I/O devices (and warn you if so) and draw a picture of the correct positions of the toggles on the base address DIP switch. If you like to understand the details, see Fig. 2.1 for an explanation of base address switch settings. Set the DAS-4 base address dipswitch to correspond with your choice.

The only other setting on the DAS-4 is the choice of hardware interrupt level. If you are not going to use interrupts in your programming (mode 4 of the DAS4.BIN driver) the interrupt jumper can be left in the "X" (inactive) position. If you intend to use interrupts, set the jumper to a level that is not in use by any other peripheral board (see Section 3.5, mode 0 for more information).

To install the board, TURN OFF THE POWER on your computer and remove the case (See the "Guide to Operations" manual for your computer if you are not already expert at this maneuver). Remove a vacant back plate by undoing the screw at the top and plug the DAS-4 in and secure the backplate. The board will fit in any of the slots of the IBM PC, XT or Portable computer but should not be placed in the short slot next to the power supply on a PC/XT. This slot is intended for the expansion interface and has slightly different signals from the others, DAS-4 will not work properly in it although it will not cause any damage. On the PC/AT, DAS-4 can be plugged into any socket but it will not make use of the extended AT bus interface connector. Installation is now complete. You may plug any of the DAS-4/8 accessories or your own cable into the 37 pin D connector on the rear.

Remember, TURN OFF THE POWER whenever installing or removing any peripheral board including the DAS-4. Never try to install or remove any peripheral board with the power on as it can cause costly damage to the electronics of your computer and/or the DAS-4 board.

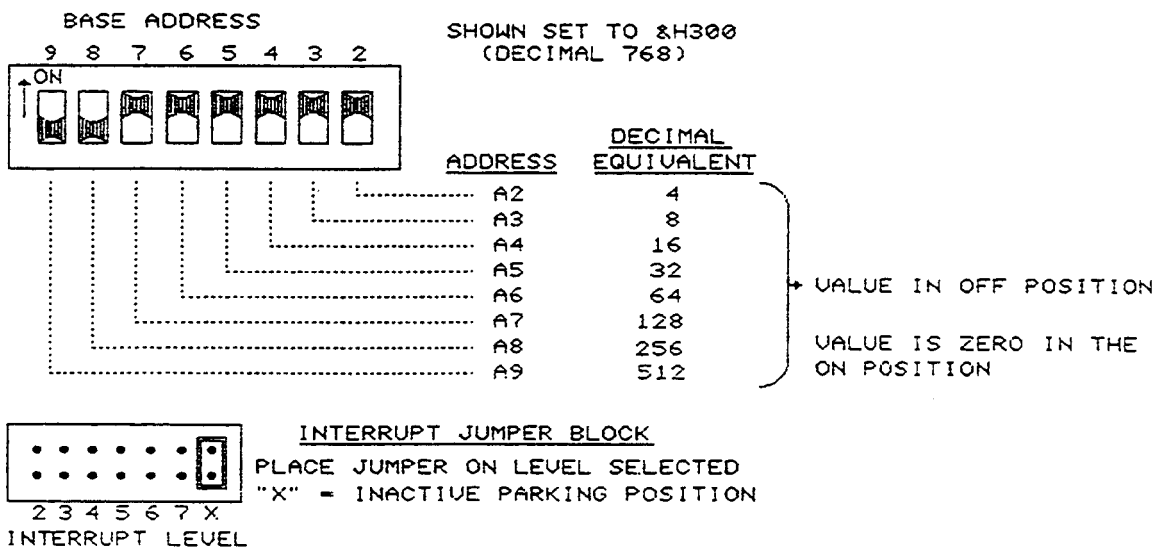


FIG. 2.1 BASE ADDRESS & INTERRUPT SELECTION

Section 3

PROGRAMMING

3.1 PROGRAMMING DAS-4

At the lowest level, DAS-4 can be programmed using I/O input and output instructions. In BASIC these are the INP (X) and OUT X,Y functions. Assembly language and most other high level languages have equivalent instructions (IN AL,DX and OUT DX,AL). Use of these functions usually involves formatting data and dealing with absolute I/O addresses. Although not demanding, this can require many lines of code and necessitates an understanding of the devices, data format and architecture of the DAS-4. To simplify program generation, a special I/O driver routine "DAS4.BIN" is included in the DAS-4 software package. This may be accessed from BASIC by a single line CALL statement and covers the majority of common operating modes. The various modes of the CALL routine select all the functions of the DAS-4, format and error check data, and perform frequently used sequences of instructions. An example is Mode 2 which performs a sequence of operations required to perform an A/D conversion, check A/D status, read data and increment the multiplexer.

Using the DAS4.BIN driver saves a lot of programming time and has some other benefits as well. For instance, the driver supports data collection on interrupt from an external source. Note that BASIC has no interrupt processing functions and so called "background" data collection using these methods is only available using the CALL routine.

Both methods of programming using INP and OUT functions and the CALL routine achieve the same result, and you are free to choose either although usually the BASIC programmer will find the CALL routine much simpler to implement. If you need to perform specialized scans such as non-sequential channels, special interrupt routines etc. you can modify the DAS4.BIN driver to your requirements. The fully commented assembly source is supplied on the utility disk (DAS4.ASM) and can be re-assembled using the Microsoft Macro-Assembler (any version) and is a good starting point for assembly language programmers who wish to modify the standard driver. For the steps involved in generating a BLOADable DAS4.BIN file, follow the instructions in the file HOWTO.BIN file on the disk (enter TYPE HOWTO.BIN after the DOS prompt).

3.1.1 I/O ADDRESS MAP OF DAS-4

First of all let's take a look at the I/O address map of the DAS-4:-

ADDRESS			READ	WRITE
Base Address	+	0	Always zero	Start A/D conversion
	+	1	A/D data (8 bit)	Start A/D conversion
	+	2	Status register	Control register
	+	3	Status register (same as Base + 2)	-

The DAS-4 has an I/O address map that is a subset of the 12 bit DAS-8, it uses only the first 4 addresses, but the register format and functions are identical. This is to maintain hardware compatibility with MetraByte's model DAS-8, 12 bit A/D board, so that the DAS-4 can be used with existing software e.g. Labtech Notebook etc. that has drivers for the DAS-8.

3.1.2 STARTING THE A/D CONVERTER

An A/D conversion is initiated by writing to either location BASE ADDRESS + 0 or BASE ADDRESS + 1. The value of any data written to these locations is irrelevant and will be lost in any case. The ensuing A/D conversion will take about 20 microseconds to complete, its progress can be monitored from bit 7 of the status register.

The DAS-4 uses the AD7574 8 bit A/D converter. One of the design "features" of this converter is that it will not start another conversion until data from the previous conversion has been read. It is a good precaution to perform a read of the A/D and throw away the data before issuing a write to BASE ADDRESS to start a new conversion.

Starting an A/D conversion:-

```
        xxx10   OUT BASADR%, 0
or:-
        xxx10   OUT BASADR% + 1, 0
```

3.1.3 READING THE A/D DATA

After the end of conversion the data from the A/D may be read from location BASE ADDRESS + 1. The data format is:-

BIT POSITION	D7	D6	D5	D4	D3	D2	D1	D0
(BASE ADDRESS + 1)	B1 (MSB)	B2	B3	B4	B5	B6	B7	B8

Location BASE ADDRESS + 0 will always return zero when read, this is to maintain compatibility with the 12 bit A/D data of the DAS-8.

The A/D data bits B1-B8 correspond to an offset binary code:-

BINARY	HEX	DECIMAL	ANALOG INPUT VOLTAGE
0000 0000	00	0	-5.000 v (-Full scale)
0000 0001	01	1	-4.961 v
. .	.	.	.
. .	.	.	.
0100 0000	40	64	-2.500 v (-1/2 scale)
. .	.	.	.
. .	.	.	.
1000 0000	80	128	+/-0 v (zero)
1000 0001	81	129	+0.039 v
. .	.	.	.
. .	.	.	.
1100 0000	C0	192	+2.500 v (+1/2 scale)
. .	.	.	.
. .	.	.	.
1111 1111	FF	255	+4.961 v (+Full scale)

A sequence of BASIC instructions to read the data and turn it into volts would be:-

```
xxx10 INP(BASADR% + 1), D%      'read data in bits
xxx20 V = (D%-128)*5/128        'scale to volts
```

3.1.4 THE DAS-4 STATUS REGISTER

The status register provides information on the operation of DAS-4. It is a read only register at I/O location BASE ADDRESS + 2 (or BASE ADDRESS + 3) and has the following format:-

BIT POSITION	D7	D6	D5	D4	D3	D2	D1	D0
(BASE ADDRESS + 2)	EOC	IP3	IP2	IP1	IRQ	MA2	MA1	MA0

The bits have the following significance:-

EOC: End of Conversion. If EOC is high (Logic 1) the A/D is busy performing a conversion. Data should not be read in this condition as it will be invalid. Wait for the EOC to return to logic 0. signifying valid data available.

IP3 - IP1: These bits correspond to the three digital input port lines IP3, IP2 and IP1. They may be used for any digital input data.

IRQ: After generation of an interrupt to the processor IRQ is set to logic high (1). It is reset to logic low (0) by a write to the control register. This provides a means of acknowledging or "handshaking" DAS-4 interrupts.

MA2-MA0: These bits provide the current analog multiplexer channel address as follows:-

MA2	MA1	MA0	CHANNEL
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

3.1.5 THE DAS-4 CONTROL REGISTER

The control register sets the multiplexer (channel) address, enables and disables interrupts and provides output data to the 4 general purpose digital outputs OP1-OP4. The control register is a write only register located at I/O address BASE ADDRESS + 2 (same location as status register). The data format of the control register is:-

BIT POSITION	D7	D6	D5	D4	D3	D2	D1	D0
(BASE ADDRESS + 2)	OP4	OP3	OP2	OP1	INTE	MA2	MA1	MA0

The bits have the following significance:-

OP4-OP1: These bits correspond to the four general purpose digital output lines OP1 thru OP4. These lines can be used for external control functions e.g. driving an input sub-multiplexer to increase the number of analog input channels. A 16 channel mux. on each of DAS-4's 8 analog channels can expand the system to 128 channels.

INTE: DAS-4 generated interrupts are enabled onto any of the selected IBM P.C. interrupt levels 2-7 if INTE = 1 (logic high). Interrupts are disabled if INTE = 0 (logic low). Interrupts from the INT.IN input (pin 24) are passed through to the selected level and are positive edge triggered. It is the programmer's responsibility to set up an interrupt handling routine, interrupt vectors and initialize the 8259 interrupt controller on the IBM P.C. processor board. Writing to the control register will clear the IRQ bit of the status register.

MA2-MA0: These bits select the current analog multiplexer channel address as follows:-

MA2	MA1	MA0	CHANNEL
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

The multiplexer channel address can be determined at any time by reading the status register.

One further note about the control register. During power up of the IBM P.C. when the RESET line of the IBM P.C. is asserted, the DAS-4 control register is cleared. This insures that DAS-4 interrupts are disabled, sets digital outputs OP1-4 to zero and sets the multiplexer channel address to zero.

3.1.6 SOME BASIC PROGRAMMING TIPS

Some BASIC commands which you may not have used frequently in the course of ordinary programming, but may be useful with DAS-4 are:-

WAIT port, n[,m]

Data read at the port is exclusive-or'ed (XORed) with integer expression "m" and ANDed with "n". If the result is zero, BASIC loops back and tests the port again until a non-zero result is obtained. This is an excellent way of making your program wait until some desired external condition is attained.

ON TIMER (n) GOSUB line

This command is only available in DOS 2.0 and above. In effect it provides you with a pseudo-interrupt. After execution of each BASIC statement, BASIC checks the timer to see if the condition >n (1 < n < 86,400 seconds) is satisfied. If

it is, control passes to the subroutine, otherwise the next line is executed. This polling of the timer is called trapping and is activated by:-

```
TIMER ON
```

Trapping is disabled by:-

```
TIMER OFF
```

Note that trapping only occurs while your BASIC program is executing (unlike a true interrupt) and can be slightly delayed by statements that require a lot of execution time. If you need to sample at long intervals the ON TIMER command is a convenient way to do it.

If you wish to program an A/D conversion using BASIC INP and OUT rather than the more powerful CALL routine, the following sequence should be followed:-

```
xxx10 OUT BASADR% +1, 0                          'start A/D conversion
xxx20 IF INP(BASADR%+2)>=&H80 GOTO xxx20  'status test loop
xxx30 D%=INP(BASADR%+1)                          'read A/D data
```

Note that looping while the A/D converter is still busy (line xxx20) is not required in interpreted BASIC as the interpreter execution time greatly exceeds the A/D conversion time of 20 microseconds. However, if the program is subsequently compiled using the BASIC COMPILER, the execution time is reduced to a point where the status test of line xxx20 is essential to avoid returning erroneous data in line xxx30.

3.2 LOADING THE MACHINE LANGUAGE CALL ROUTINE "DAS4.BIN"

As the previous simple example shows, direct I/O using BASIC INP and OUT can be somewhat tedious to implement although a lot of the required programming could be handled in subroutines. Use of the CALL routine described in the this section avoids these problems, circumvents the execution time delays of interpreted or compiled BASIC, and also permits background data collection using interrupts.

In order to make use of the CALL routine "DAS4.BIN", it must first be loaded into memory. You must avoid loading it over any part of memory that is being used by another program e.g. BASIC (watch out for BASICA.EXE if you are using GW BASIC) print spoolers or "RAM-Disk". If you do interfere with another program's use of memory, the CALL routine will not work and most likely your PC will do strange things for a second or two and then hang up (No damage will result, but to get things going again you will have to turn off power and wait a few seconds before turning on again). ***Here is some advice, when you write programs with CALLs in them, SAVE them before you run them!*** Note that the information given in this section is general and would apply to loading any CALL or USR routine and supplements the limited information provided in the "I.B.M. BASIC MANUAL".

When you load the DAS4.BIN CALL routine, you have two options depending on the size of your available memory. Due to the design of Microsoft Basic, the maximum memory segment

that BASIC is able to use is 64K bytes. If BASIC is using its maximum 64K you will get the following message on power up or from DOS by entering A> BASIC(A) :-

```
BASIC       The IBM Personal Computer Basic
            Version D1.10 Copyright IBM Corp. 1981, 1982
            61807 Bytes free
```

The exact number of "Bytes free" varies with the version of BASIC(A) and DOS but is usually greater than 60000 bytes if an excess of memory over and above what BASIC can use is available. When the number of memory bytes free is less than 60000, this is an indication that your PC's memory is already fully utilized and BASIC is adjusting to this condition by using less than its possible 64K maximum workspace. If this is the case, you will have to load the CALL routine by further forced contraction of the BASIC workspace and loading the routine at the end of the newly defined workspace. DAS4.BIN will occupy about 1852 bytes but to keep things simple, let's clear a 2K (2048 byte) space for it.

Step 1 is to work out how much memory BASIC is able to use. First just load BASIC(A) from DOS:-

```
A> BASIC(A)
```

Note the number of bytes free in BASIC's greeting messages. Next do a SYSTEM command to return to DOS and this time load BASIC(A) with the optional /M parameter:-

```
A> BASIC(A) /M:WS  where WS is a number (not a variable)
```

Try setting the *WS* (workspace) parameter to 30000 or 40000 and then note the number of bytes free in the greeting message. The objective is to determine the workspace that must be specified to reduce the bytes free by at least 2048 bytes. Once this is determined you can either load BASIC(A) by specifying this workspace or include a CLEAR command right at the beginning of your program e.g.:-

```
xxx10 CLEAR, WS
```

Next, we need to know what segment BASIC is occupying in memory. In all versions of Microsoft derived BASIC, this can be found from the contents of absolute memory locations &H511 and &H510 which hold the current BASIC segment which we can call SG. SG can be determined as follows:-

```
xxx20 DEF SEG = 0        'define current segment = 0000
                          before reading absolute
                          addresses 0000:0510 & 0000:0511

xxx30 SG = 256*PEEK(&H511) + PEEK(&H510)
```

The segment address at which we can now load the CALL routine will simply be at the end of the working space i.e.:-

```
xxx40 SG = WS/16 + SG     'remember segment addresses
                           are on 16 bit boundaries
```

The routine can now be loaded as follows:-

```
xxx50 DEF SEG = SG
xxx60 BLOAD "DAS4.BIN",0   'loads routine at SG:0000
```

A BLOAD must be used as we are loading a binary (machine language) program. Once loaded, the CALL can be entered as many times as needed in the program after initializing the call parameters MD%, D%, FLAG% prior to the CALL sequence as follows:-

```
xxx70 DEF SEG = SG
xxx80 DAS4 = 0
xxx90 CALL DAS4 (MD%, D%(0), FLAG%)
```

Note that in interpreted Microsoft BASIC, DAS4 is a variable that specifies the memory offset of the starting address of the CALL routine from the current segment as defined in the most recent preceding DEF SEG statement. We have chosen DAS4 as a name, as it makes CALL DAS4 easy to associate with the device and would distinguish it from any other CALLs to other routines that might be in the same program. In compiled BASIC, Quick Basic or other compiled languages (FORTRAN, PASCAL etc.) the significance of DAS4 in the CALL is different. In this case, it is the name of the external routine that the linker will be looking for when linking after compilation. DAS4.OBJ is supplied on the software disk for this purpose (see Sections 3.18 & 3.19).

Returning to interpreted BASIC, DAS4 is the offset (actually zero) from the current segment as defined by the last DEF SEG statement that tells your BASIC interpreter where the CALL routine is located. Be careful that you do not inadvertently redefine the current segment somewhere in a program before entering the CALL. If you are using DEF SEG in other parts of your program, it is good practice to immediately precede the CALL statement by the appropriate DEF SEG statement (the same one you preceded your BLOAD with) even at the cost of duplication. This precaution can save a lot of wasted time and frustration from crashing your computer!

Another important detail to understand is that CLEAR sets working space from the bottom of the BASIC working area up whereas we must set aside space for our subroutine from the top of **available** memory down. If we attempt to CLEAR more space than is actually available, we will end up loading our routine over the end of the BASIC program, data space and stack and will hang up the computer. Be careful this does not happen inadvertently if you are memory limited and later load BASIC with DEBUG or some other memory resident program without making a compensating reduction in the workspace (WS) declaration in the CLEAR statement. When memory is limited, setting up a workspace that is a considerable amount less than what is available is a simple precaution. Another precaution in this regard occurs on computers (other than the IBM P.C.) that do not contain most of the BASIC in ROM. These machines load BASIC completely into RAM in the form of a BASICA.COM and a BASICA.EXE file. BASICA.COM, DOS and the BASIC program space load from the bottom of memory up and BASICA.EXE loads from the top of memory down. The free memory ends up in the middle and some contraction of the BASIC program space will automatically take place as soon as the total memory minus that used by DOS and BASICA.EXE becomes less than 64K. This can easily happen on a machine with less than 128K of memory in which case further forced contraction of the workspace is the only way of loading the CALL routine. These considerations all sound a little on the complex side, but like all things they are not once you understand what you are doing. To further assist you, run and list the file EX00.BAS. This gives you an examples of loading and using the CALL routine and ready made loading and initializing code that you can use as a "front end" to your own programs.

The second method of loading the driver is somewhat simpler to follow and applies when you have plenty of memory (generally 256K or more) and you have the luxury of loading the CALL routine outside the BASIC workspace. In this case choose a segment that has 2K bytes clear at its beginning. For example we might choose &H6000 which is at 384K. Then proceed as follows:-

```
xxx10 DEF SEG = &H6000        'Sets up load segment
```

```
xxx20 BLOAD "DAS4.BIN",0    'Loads at 6000:0000
xxx30 DAS4 = 0
xxx40 DIM D%(3)
      ''''
      ''''
      ''''
xxxxx DEF SEG = &H6000
xxxyy CALL DAS4 (MD%, D%(0), FLAG%)
xxxzz etc.
```

An example of this approach is contained in file EX0.BAS. Before you try loading outside the workspace, be sure you really do have an unused 2K of memory at 384K. You can change the DEF SEG statements in EX0.BAS and experiment with loading the CALL routine at other locations. Usually any clash with another program's use of the same memory results in obliteration of some of the routine code and a failure to exit and return from the routine. The computer hangs up, and the only cure is to switch off, wait a few seconds and turn on the power again. Note that memory resident programs such as Borland's Sidekick will raise the loading floor in memory for additional programs such as BASIC(A). In this case even machines with large amounts of memory may run out of space or require loading the DAS4.BIN even higher in memory than the example above. Also higher revisions of DOS with additional features use more resident space in memory than earlier versions and may require loading the CALL routine higher up.

3.3 FORMAT OF THE CALL STATEMENT

If you are new to using CALL statements, this explanation may assist you in understanding how the CALL transfers execution to the machine language (binary) driver routine. Prior to entering the CALL, the DEF SEG = SG statement sets the segment address at which the CALL subroutine is located. The CALL statement for the DAS4.BIN driver must be of the form:-

```
xxxxx  CALL DAS4 (MD%, D%(0), FLAG%)
```

As explained in the previous section, DAS4 is the address offset from the current segment of memory as defined in the last DEF SEG statement. In all of our examples, we have chosen to define the current segment to correspond with the starting address of the CALL routine, therefore this offset is always zero and DAS4 = 0.

The three variables within brackets are known as the CALL parameters. On executing the CALL, the addresses of the variables (pointers) are passed in the sequence written to BASIC's stack. The CALL routine unloads these pointers from the stack and uses them to locate the variables in BASIC's data space so data can be exchanged with them. Three important format requirements must be met:-

1. The CALL parameters are referenced by position. The subroutine knows nothing of the names of the variables, just their locations from the order of their pointers on the stack.

   If you write:-

   ```
   xxxxx CALL DAS4 (D%(0), MD%, FLAG%)
   ```

you will mix up the CALL routine, since it will interpret D%(0) as the mode number, the mode MD% as the data etc.. *The parameters must always be written in the correct order:-*

(mode, data, errors)

2. The CALL routine expects its parameters to be integer type variables and will write and read to the variables on this assumption. If you slip up and use a non-integer (real single or double precision) variable in the CALL parameters, the routine will not function correctly. No error checking is done in the CALL on the variable type, so be careful since you may crash the computer!

3. You **cannot** perform any arithmetic functions within the parameter list brackets of the CALL statement e.g:-

   CALL DAS4 (MD% + 2, D%(0) * 8, FLAG%)

   is illegal and will produce a syntax error.

4. You cannot use constants for any of the parameters in the CALL statement. The following is illegal:-

   CALL DAS4 (7, 2, FLAG%)

   This must be programmed as:-

```
xxx10 MD% = 7
xxx20 D%(0) = 2
xxx30 CALL DAS4 (MD%, D%(0), FLAG%)
```

Apart from these restrictions, you can name the integer variables what you want, the names in the examples are just convenient mnemonics. Strictly, you should declare the variables before executing the CALL. If you do not, the simple variables will be declared by default on execution, but array variables cannot be dimensioned by default and must be dimensioned before the CALL to pass data correctly if used as a CALL parameter. Many modes of the DAS4.BIN CALL routine require multiple items of data to be passed in an array. For this reason, D%(0) is specified as the data variable so that the CALL routine can locate the whole array from the position of its initial element.

You can use some elegant techniques with the CALL parameters. Let's say we wanted to record or output a whole series of data in a FOR . . . NEXT loop. You can dimension your D% array as a two or more dimension array, for example:-

```
xxx00 DIM D%(4,100)
xxx10 DEF SEG = SG
xxx20 FOR I = 0 to 100
xxx30 CALL DAS4 (MD%, D%(0,I), FLAG%)
xxx40 NEXT I
```

Likewise any of the other CALL parameters may be integer array variables if required, and you can name any number of different integer data arrays for output and input. It is O.K. to dimension arrays with more elements than will be used by the CALL, unused elements will be unchanged and for example could be used for tagging data with time, date or other information.

MetraByte has chosen to use the same CALL structure for selecting any of the functions of the board. This makes the CALL structure easy to remember and helps to avoid errors compared to CALLs with variable offsets and number of parameters. One consequence of this is that not all functions or modes use all of the data array D%(*). In practice it is dimensioned to D%(3) to allow for the needs of modes 4 & 7 which use all four elements, but most of the other modes only use a few elements. In the interests of clarity, our example programs use inline code with each CALL written out separately, but you can just as readily put the CALL into one subroutine for the whole program and save a lot of lines of code, as in the following example. After BLOADing the DAS4.BIN driver (see EX0.BAS) proceed as follows (this program logs channels 2 - 5 to disk every 10 seconds):-

```
xx100 DIM D%(3)
xx110 MD% = 0
xx120 D%(0) = &H300 : D%(1) = 2
xx130 GOSUB 10000      'initialize
xx140 MD% = 1
xx150 D%(0) = 2 : D%(1) = 5
xx160 GOSUB 10000      'set scan limits
xx170 OPEN "MYFILE.DAT" FOR OUTPUT AS #1
xx180 TNOW = TIMER  'get system time (in seconds)
xx190 MD% = 2
xx200 GOSUB 10000      'do 1 A/D conversion
xx210 PRINT #1, D%(0)*5/128          'save data to disk scaled in volts
xx220 IF INKEY$<>"" THEN CLOSE #1:END    'finish if key pressed
xx230 IF TIMER > TNOW + 10 THEN GOTO xx180 ELSE GOTO xx220
                                   'get next channel at 10 second intervals

10000 CALL DAS4 (MD%, D%(0), FLAG%)     'subroutine for all calls
10010 IF FLAG%<>0 THEN ?"Error # ";FLAG%;" in mode ";MD%:STOP
10020 RETURN
```

3.4 EXAMPLES OF THE USE OF THE CALL ROUTINE

The following subsections give detail information and examples of the use of the CALL routine in all 10 modes. The modes are selected by the MD% parameter in the CALL as follows:-

MODE		FUNCTION
0	...	Initialize, input DAS-4 base address, interrupt level & check hardware.
1	...	Set multiplexer low & high scan limits.
2	...	Perform a single A/D conversion. Return data and increment multiplexer address. (Programmed conversion). Speed up to 200Hz, Operation - foreground.
3	...	Perform an N conversion scan after trigger. Conversions initiated by an external input. Data is transferred to an integer array. Speed up to 3KHz. Operation - foreground.
4	...	Perform an N conversion scan after trigger into a memory buffer area. Conversions initiated by an external input. Data transferred by interrupt. Speed up to 3KHz. Operation - background.
5	...	Analog trigger function similar to a scope trigger (specify channel, level & slope)
6	...	Transfers data from memory buffer to an integer array either as a whole block or piece by piece (used after mode 4).
7	...	Return status. Reports next channel number to be converted, whether interrupt is active or finished, interrupt level and remaining number of conversions in interrupt mode 4.
8	...	Read digital inputs IP1-3.
9	...	Write digital outputs OP1-4.

3.5 MODE 0 - INITIALIZE

Before using any other mode of the CALL routine, you must provide the I/O location, or BASE ADDRESS, of the DAS-4 board and the hardware interrupt level that you intend to use. If you fail to provide this information before accessing any other mode of the CALL, error flag (FLAG% = 1, driver not initialized) will be obtained and none of the other modes will execute. Calling mode 0 need only be done once in the initialization section of your program usually straight after loading the DAS4.BIN routine. For examples of loading DAS4.BIN and initializing, run and list EX0.BAS and EX00.BAS.

On entry the following parameters should be assigned:-

```
MD% = 0                          (mode)

D%(0) = &H300    'for example    (I/O address)

D%(1) = 2        'for example    (interrupt level, 2 thru 7)

D%(2) thru D%(3)                 (value does not matter)

FLAG% = X                        (value does not matter)
```

then:-

```
CALL DAS4 (MD%, D%(0), FLAG%)
                  ^
```

Note that specifying the first element of the array will pass all other required array parameters.

On return the variables contain data as follows:-

```
MD% = 0              (unchanged)

D%(0) thru D%(3)     (unchanged)
```

The following error codes apply to mode 0:-

```
FLAG% =  0    (no error, o.k.)
      =  2    (mode number out of range, <0 or >9)
      =  3    (hardware failure)
      =  10   (base address out of range <256 or >1008)
      =  11   (interrupt level <2 or >7)
```

The standard IBM PC or PC/XT allows use of I/O addresses between 512 and 1008 (Hex 200 - 3F0), the PC/AT allows use of addresses between 256 and 1008 (Hex 100 - 3F0), all machines use I/O addresses below 256 (Hex 100) for internal devices, hence the driver will flag any attempt to assign an I/O address in the range 0 - 255 (Hex 0 - FF) by returning error code #10 in the FLAG% variable.

Error #11 will occur if you have specified a non-valid interrupt level. The available levels on the P.C. expansion bus correspond to 2 thru 7. Certain of these levels may be in use by other

peripheral devices (especially level 6 used by floppy disk drive). A list of the standard IBM interrupt assignments is:-

Level 2	-	Reserved (but not used) by Color Graphics adapter
Level 3	-	Serial I/O - used if COM2: installed.
Level 4	-	Serial I/O - used if COM1: installed.
Level 5	-	Printer - may be used by LPT2: if installed.
Level 6	-	Always in use by disk drives
Level 7	-	Printer - may be used by LPT1: if installed.

If you do not have a particular device installed, it is safe to assume that level is available for use by DAS-4. The lower the level number, the higher the interrupt priority. ***Note that the interrupt will not be enabled unless you enter mode 4 which requires interrupts for operation.*** If you are ***not*** going to make use of interrupts any level can be chosen e.g. D%(1) = 2.

Mode 0 performs several other initializing functions. Default scan limits of channels 0 & 7 are set. Mode 0 also performs a simple read/write test and a check on the busy signal from the A/D as a check on the presence and function of the DAS-4 hardware. If you obtain error 3, it is either indicative of a hardware fault in the DAS-4 or more commonly a discrepancy between the base address specified in D%(0) and the actual switch setting on the board.

3.6 MODE 1 - SET MULTIPLEXER SCAN LIMITS

Mode 1 is used to set the scan limits of the multiplexer to other than the default limits provided by mode 0. This is done prior to performing conversions in modes 2, 3 & 4. Two limits are passed. D%(0) contains the lower limit and D%(1) the higher limit. If the default limits of mode 0 are o.k. for you, it is not necessary to enter mode 1.

To illustrate the action of mode 1 assume we set D%(0) = 3 and D%(1) = 6. The first conversion (commanded by modes 2, 3 or 4) would be performed on channel 3, data returned and the channel incremented to 4. The next conversion would be performed on channel 4, data returned and the channel incremented to 5 etc. After the conversion on channel 6 has been performed, the software counter in the driver controlling the mux will automatically re-load to the start of scan limit to repeat the sequence. Scanning of channels would always be stepped between and including channels 3 and 6 as follows:-

3-4-5-6-3-4-5-6-3- etc.]

If you specify the lower limit greater than the higher limit i.e. D%(0) > D%(1) you will receive error code 4 as this is an illegal setup condition.

If you wish to perform continuous conversions on one channel, then set the low and high limits equal to each other and the desired channel number e.g. setting D%(0) = 1 and D%(1) = 1 would perform continuous conversions on channel 1.

Note that after exit from mode 1, the starting channel will always be D%(0), the lower limit.

On entry the following parameters should be initialized:-

MD% = 1	(mode number)
D%(0) = 0 thru 7	(lower scan limit)
D%(1) = D%(0) thru 7	(upper scan limit)
D%(2 thru 3)	(value does not matter)
FLAG% = X	(value does not matter)

then:-

CALL DAS4 (MD%, D%(0), FLAG%)

On return the variables contain data as follows:-

MD% = 1	(unchanged)
D%(0 thru 3)	(unchanged)

The following error codes apply to mode 1:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)
	=	4	(if limits reverse order, D%(0) > D%(1))
	=	10	(if lower channel limit D%(0) <0 or >7)
	=	11	(if upper channel limit D%(1) <0 or >7)

An example of the use of mode 1 will be found in EX2.BAS as well as several of the other examples using the A/D converter.

3.7 MODE 2 - DO ONE A/D CONVERSION AND INCREMENT MUX

Mode 2 performs one A/D conversion by software command. The mux is automatically incremented after the conversion through software routines in the driver. Data is returned as follows:-

D%(0) - A/D data (-128 to +127 bits)

D%(1) - Channel number

Data is transferred to D%(0) in 2's compliment form (standard integer) with -128 bits corresponding to -Full Scale of -5v and +127 corresponding to +Full Scale of +4.961 volts. Zero volts corresponds to zero bits. This minimizes processing of the data after exit from the CALL.

D%(1) contains the channel from which the data is derived. This information can be used or ignored as required.

The A/D will perform conversions on channels in accordance with the scan limit conditions set in mode 1. If mode 1 has not been entered prior to mode 2, conversions will cycle between channel 0 and channel 7.

On entry the following parameters should be initialized:-

MD% = 2	(mode number)
D%(0 thru 3) = X	(value does not matter)
FLAG% = X	(value does not matter)

then:-

```
CALL DAS4 (MD%, D%(0), FLAG%)
```

On return the variables contain data as follows:-

MD% = 2	(unchanged)
D%(0) = A/D data	(-128 to +127 bits)
D%(1) = Channel number of data (0 - 7)	
D%(2 thru 3)	(unchanged)

The following error codes apply to mode 2:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)
	=	3	(No EOC from A/D, time out indicative of hardware failure)

An example of the use of mode 2 will be found in EX2.BAS.

If you wish to perform a series of A/D conversions using mode 2, be aware that the speed becomes limited by the program execution time which for interpreted BASIC is slow (several milliseconds per line of code). A tight loop such as the one below will perform around 200 conversions/sec. on a standard IBM P.C. (4.77MHz clock):-

```
xxx10 DIM X%(10000)
xxx20 MD% = 2
xxx30 FOR I% = 0 TO 10000
xxx40 CALL DAS4 (MD%, D%(0), FLAG%)
xxx50 X%(I%) = D%(0)
xxx60 NEXT I%
        etc.
```

If this is compiled using the BASIC compiler, a conversion rate of about 4000 samples/sec. will be obtained.

3.8 MODE 3 - DO N A/D CONVERSIONS DIRECT TO ARRAY

Mode 3 performs N A/D conversions and transfers data directly into a BASIC integer array. N may be any number of conversions up to 32,767 although in practice it is impossible to dimension an integer array with 32,767 elements and leave any workspace for the program (30,000 is a more practical limit). Since the CPU is performing the A/D polling and data transfers as a "foreground" operation, exit from the CALL will ***not*** occur until all conversions have been completed. To provide an escape route, hitting any key on the keyboard while data is being gathered in mode 3 will abandon further conversions and produce an immediate return to your BASIC program. You will receive error code 5 as a warning that you have aborted the mode in this case. If you do not want to wait for data to be collected, mode 4 can be used to gather the data as a "background" operation so that your program is able to process data and collect it at the same time.

The A/D will perform conversions on channels in accordance with the scan limit conditions set in mode 1. If mode 1 has not been entered prior to mode 3, conversions will start on channel 0 and the upper scan limit will be channel 7.

In mode 3, each A/D conversion is initiated by a positive edge on the interrupt input (pin 24) although this mode does not make use of hardware interrupts. Triggering may be held off by holding IP1 low, as soon as IP1 goes high A/D conversions will commence and will continue until the full conversion count even if IP1 goes low again.

On entry the following parameters should be initialized:-

MD% = 3 (mode number)

D%(0) = VARPTR(ARRAY%(M)) - array pointer
Conversions may be loaded starting at the M'th. position in an array or at the start if M = 0.

D%(1) = Number of conversions required (Word count).
Range 1 to N where N-1 <= array dimension

D%(2 thru 3) - (value does not matter)

FLAG% - (value does not matter)

then:-

```
CALL DAS4 (MD%, D%(0), FLAG%)
```

On return the variables contain data as follows:-

MD% = 3 (unchanged)

D%(0 thru 3) (unchanged)

ARRAY%(M) = 1st. data word
ARRAY%(M+1) = 2nd. data word
ARRAY%(M+2) = 3rd. data word

```
. . .
etc.
```

The following error codes apply to mode 3:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)
	=	3	(hardware error - A/D not converting)
	=	5	(mode aborted by keyboard)
	=	11	(number of conversions D%(1)<0 or >32767)

An example of the use of mode 3 will be found in EX3.BAS.

A number of precautions apply to the reliable use of mode 3. First you must dimension a receiving array that has at least as many elements as the number of conversions specified in D%(1). No checks are made by the CALL routine on whether you are performing more conversions than the array will hold. If you do, some of the data area of BASIC will be overwritten which may destroy descriptors and other variable data and cause strange effects as a result. ***Do not overrun the array limits.***

Second, after assigning the pointer to the receiving array, do not introduce any new simple variables before entering the CALL. For example, this is O.K.:-

```
xxx10 DIM D%(3), X%(999)          'declare D%(*), X%(*)
xxx20 MD% = 3                     'mode #
xxx30 D%(0) = VARPTR(X%(0))       'pointer to array
xxx40 D%(1) = 1000                'number of conversions
xxx50 CALL DAS4 (MD%, D%(0), FLAG%)
```

But this will cause a problem:-

```
xxx10 DIM D%(3), X%(999)          'declare D%(*), X%(*)
xxx20 MD% = 3                     'mode #
xxx30 D%(0) = VARPTR(X%(0))       'pointer to array
xxx40 D%(1) = 1000                'number of conversions
xxx50 NEW_VARIABLE = X            'new variable
xxx60 CALL DAS4 (MD%, D%(0), FLAG%)
```

The problem arises because of the way BASIC stores array variables. They are located in memory above the data area for simple (non-array) variables which in turn is located above the program storage area. If you introduce a simple variable that has not been used before, BASIC makes room for this variable by re-locating all the array variables upwards in memory. If the pointer (using VARPTR) is assigned to the receiving array **before** a new variable is introduced and then the CALL is entered, the actual location of the array will have changed, and the CALL routine writes data to the old array location causing strange effects.

ALSO NOTE that exit from mode 3 ***cannot*** occur until IP1 has been taken high ***and*** a sufficient number of trigger pulses to perform the desired number of conversions have been supplied. If these conditions are not met, your computer may give the erroneous appearance of being hung up. To abort mode 3, hit any key on the keyboard.

Conversion rates in excess of 2000 samples/sec are attainable in this mode. As interrupts in the computer (mainly the timer interrupt) may divert the CPU away from attending to transferring data from the A/D for several hundred microseconds, data may be lost above 3000 samples/sec.

3.9 MODE 4 - DO N A/D CONVERSIONS AND TRANSFER TO MEMORY ON INTERRUPT

Mode 4 performs N A/D conversions through a special interrupt handler that is installed by this mode. Each time an interrupt is invoked by a positive edge on the Interrupt In (pin 24), the handler performs an A/D conversion, moves the data to a memory buffer area (up to 64K bytes) and increments the multiplexer ready for the next interrupt. The handler keeps track of the total number of conversions performed. When the number reaches N, as specified by D%(0), interrupts are disabled if in the non-recycle mode (D%(3)=0), or the process is repeated continuously to the same segment of memory if D%(3) specifies the re-cycle mode. Note that once mode 4 has enabled interrupts, conversions continue regardless of what other programs the user may be running (although they should not interfere either with the location of the DAS4.BIN driver or the A/D data area). For this reason it is termed a background operation. About 3000 samples/sec. are possible in mode 4 although at this speed so much processing is taking place in the background, you may notice a significant reduction in speed of foreground operations.

To return data to a BASIC integer array when mode 4 is operating or has finished operation, use mode 6. To assess the progress of an operation initiated by mode 4, use mode 7. To abort an interrupt operation initiated by mode 4, re-enter mode 4 but with D%(2) = 0. For detail descriptions of the features of these other modes, see the following sections.

The A/D will perform conversions on channels in accordance with the scan limit conditions set in mode 1. If mode 1 has not been entered prior to mode 4, conversions will start on channel 0 and the upper scan limit will be channel 7.

The A/D triggering occurs indirectly through the interrupt handler. On a 4.77 MHz standard 8088 based PC, there is typically an 80 microsecond latency between generation of the interrupt and start of A/D conversion. This latency varies according to other interrupt activity especially the internal time of day interrupt on level 0. IP1 acts as a start gate for the operation, holding IP1 low will hold off triggering. To start triggering IP1 should be taken high for at least 100 microseconds after which its subsequent state will have no effect. Hitting any key of the keyboard will also start conversions while IP1 is held low, in this case error code 5 is produced as a reminder that the keyboard initiated conversions.

On entry the following parameters should be initialized:-

MD% = 4 (mode number)

D%(0) = Number of conversions required (Word count).
Range -32768 to +32767
Note: BASIC only provides signed integers, data is passed as an unsigned integer (see Appendix C). e.g. -1 corresponds to 65,535 conversions.

D%(1) = Segment of memory to receive data. Segments are on paragraph (16 bit) boundaries e.g. D%(1) = &H7000 would start loading A/D data at 448K. Be sure to choose an empty area of memory for a data buffer (always outside BASIC workspace)

D%(2) - Enables/disables interrupt:-

D%(2) = 1 : Enables interrupt. Interrupts will automatically disable when the conversion count is reached in non-recycle mode.

D%(2) = 0 : Disables interrupt. Used to abort an active interrupt or stop further interrupts when operating in the recycle mode.

D%(3) - Single cycle/Re-cycle operation:-

D%(3) = 0 : One cycle. After completion of the number of conversions specified, interrupts are disabled, setting the operation status to zero.

D%(3) = 1 : Re-cycle. In this case data is continuously written to the same memory. D%(0) corresponds to the memory "buffer" length. The status of the operation is 1 = active until stopped by re-entry with D%(2) = 0.

FLAG% = X (value does not matter)

then:-

CALL DAS4 (MD%, D%(0), FLAG%)

On return the variables contain data as follows:-

MD% = 4 (unchanged)

D%(0 thru 3) (unchanged)

The following error codes apply to mode 4:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)
	=	5	(if mode initiated by keyboard)
	=	12	(if D%(2) neither 0 or 1)
	=	13	(if D%(3) neither 0 or 1)

An example of the use of mode 4, 6 & 7 will be found in EX4.BAS. Several details apply to the reliable use of mode 4. On completion of an interrupt operation, the selected level of the 8259 interrupt mask register is disabled and the tri-state interrupt drivers of the DAS-4 are placed in the high impedance state, also the previous interrupt vectors used by this level are restored. This allows more than one hardware device, or multiple DAS-4's to use the same interrupt level as long as they do so sequentially. Since the segment registers are not incremented by the handler, the maximum data area available is 64K (a page) for 65,536 conversions. Be sure that your data area is not in use by your program or altered by subsequent operations. Data may be retrieved using mode 6 during or after the operation of mode 6 and this will not alter the memory. It is possible to re-write the interrupt handler to use several segments of main or extended memory for data storage.

3.10 MODE 5 - ANALOG TRIGGER FUNCTION

Mode 5 provides an analog trigger function similar to an oscilloscope trigger. It is sometimes useful to wait for a voltage to reach a certain level before starting to gather data and mode 5 provides this capability. Any of the analog input channels may be designated as a trigger channel, and you may set the level and slope for triggering.

The main use for mode 5 is in front of any of the other data acquisition modes as a gating or wait loop until the specified analog trigger conditions are met. Since it is possible to get stuck in the wait loop indefinitely if the trigger conditions are not fulfilled, you can also exit mode 5 by hitting any key which will return you to the calling program.

Parameters D%(0) thru D%(2) control the triggering and select the trigger channel number, the trigger level and the trigger direction (slope). D%(0) specifies the trigger channel number. It may be one of the scanned channels i.e. within the scan limits and carrying one of the measured signals, or a separate channel outside the scanned channels used only for triggering. The voltage level at which triggering occurs is set by D%(1) in bits, (valid range of -128 to +127 bits). The direction of triggering or slope is controlled by D%(2), for instance if D%(1) = 51 the trigger level will be +2v and if D%(2) = 0 (positive slope) triggering will take place when the signal exceeds +2v, alternatively if D%(2) = 1 (negative slope) triggering would take place when the trigger signal becomes less than +2V.

On entry the following variables should be initialized:-

MD% = 5

D%(0) = Channel number (0 - 7)

D%(1) = Trigger level (-128 to +127 bits)

D%(2) = Slope (0 = positive, 1 = negative)

D%(3) - (value does not matter)

FLAG% - (value does not matter)

then:-

CALL DAS4 (MD%, D%(0), FLAG%)

On return the variables contain data as follows:-

MD% = 5

D%(0 thru 3) - unchanged

The following error codes apply to mode 5:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)
	=	5	(if exit aborted by keyboard)

=	10	(trigger channel data D%(0) <0 or >7)
=	11	(trigger channel level D%(1) <-128 or >127)
=	12	(trigger slope data D%(2) neither 0 or 1)

For an example of the use of mode 5, see EX5.BAS.

3.11 MODE 6 - TRANSFER DATA FROM MEMORY TO ARRAY

Mode 6 transfers data from any segment of memory to integer array variables in BASIC workspace. It is used following a mode 4 operation. Data in memory derived from mode 4 is in the form of single bytes of A/D data corresponding to each conversion.

Mode 6 functions as follows. D%(0) provides the location of the starting element of the integer array that you wish to transfer to e.g. D%(0) = VARPTR(X%(0)) would start moving data to the beginning of the array , X%(0), whereas D%(0) = VARPTR(X%(N)) would start at the N'th element. D%(1) provides the number of conversions or data bytes to transfer, and D%(2) sets the number of the conversion (segment offset) to start transferring from. The segment area of the buffer area is assumed to be the same as that specified in mode 4. By way of an example, let's move 1000 conversions, starting at the 100'th element in the area and the 20,000'th conversion in the buffer:-

D%(0) = VARPTR(X%(100))

D%(1) = 1000

D%(2) = 20000

In this way we can move data piece by piece and analyze it. Note that we could not handle 64Kbytes of data directly in an integer array, as integers take 2 bytes of storage (one word) and BASIC could not provide enough storage area (try DIM X%(65535)). We could accomplish the same effect as mode 6 using DEF SEG's and PEEK's in a loop but it would be a lot slower.

Note that you must be careful about the transfer parameters. In particular:-

1: Do not transfer more words than an array will hold or overrun the end of the array. No checking is performed to detect this condition which will corrupt BASIC workspace and cause strange effects.

2: There is nothing to prevent you transferring garbage from a source segment that does not contain A/D data or from overrunning the end of A/D data. No checking is performed to detect this condition.

On entry the following parameters should be assigned:-

MD% = 6 (mode number)

D%(0) = VARPTR(X%(N)) - (array locator)

D%(1) = Number of transfers (cannot exceed 32,767)

D%(2) = Starting conversion number

D%(3) - (value does not matter)

FLAG% - (value does not matter)

then:-

CALL DAS4 (MD%, D%(0), FLAG%)

On return the variables contain data as follows:-

MD% = 6

D%(0 thru 3) - unchanged

The following error codes apply to mode 6:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)
	=	11	(number of transfers, D%(1)=0 or >32767)

For an example of the use of mode 6, see EX4.BAS.

Once data acquisition has been set up as a constant background operation using the recycle option of mode 4, a foreground program can be processing the data as it is acquired using mode 6 to retrieve the data. This is excellent for graphics, "digital oscilloscope" and continuous write to disk applications.

Note that it is advisable to make the assignment:-

D%(0) = VARPTR(X%(N))

immediately before the CALL statement, as declaring a new simple variable after making this assignment will dynamically relocate the arrays and upset operation of this mode. If the variable I had not been declared prior to line xxx10, then the following sequence would cause a problem (see mode 3 for a fuller description of this problem):-

```
xxx10 D%(0) = VARPTR(X%(0))
xxx20 I = 3
xxx30 CALL DAS4 (MD%, D%(0), FLAG%)
```

3.12 MODE 7 - READ STATUS

Mode 7 returns status data which is especially useful when performing background data acquisition using mode 4. It can inform a foreground operation about the progress of a background operation.

On entry the following parameters should be initialized:-

MD% = 7 (mode number)

D%(0 thru 3) - (value does not matter)

FLAG% - (value does not matter)

then:-

```
CALL DAS4 (MD%, D%(0), FLAG%)
```

On return the variables contain data as follows:-

MD% = 7 (unchanged)

D%(0) = Channel number of next channel to be converted

D%(1) = Status of interrupt operation:-

0 - Done (finished)
1 - Active (in progress)

D%(2) = Selected interrupt level
(corresponds to setting of mode 0)

D%(3) = Remaining number of conversions to be done

The following error codes apply to mode 7:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)

For an example of the use of mode 7, see EX4.BAS.

3.13 MODE 8 - READ DIGITAL INPUTS IP1-3

Mode 8 allows you to read the state of digital inputs IP1-3. Data returned can range between 0 and 7 corresponding to all combinations of the 3 input bits, IP1-3.

On entry the following parameters should be initialized:-

MD% = 8

D%(0 thru 3) - value does not matter

FLAG% = X - value does not matter

then:-

CALL DAS4 (MD%, D%(0), FLAG%)

On return the variables contain data as follows:-

MD% = 8

D%(0) contains input data (range 0 - 7)

D%(1 thru 3) - unchanged

The following error codes apply to mode 8:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)

For an example of digital input using mode 8, see EX8.BAS.

3.14 MODE 9 - WRITE DIGITAL OUTPUT OP1-4

Mode 9 is used to write digital data to the 4 bit output port, OP1-4. Output data is checked to be in the range 0 - 15 and if not an error exit (error # 10) occurs.

On entry the following parameters should be assigned:-

MD% = 9

D%(0) = output data (range 0-15)

D%(1 thru 3) - value does not matter

FLAG% = X - value does not matter

then:-

CALL DAS4 (MD%, D%(0), FLAG%)

On return the variables contain data as follows:-

MD% = 9 - unchanged

D%(0 thru 3) - unchanged

The following error codes apply to mode 9:-

FLAG%	=	0	(no error, o.k.)
	=	1	(driver not initialized)
	=	2	(mode number out of range, <0 or >9)
	=	10	(D%(0), output data <0 or >15)

For an example of digital output using mode 9, see EX9.BAS.

3.15 SUMMARY OF ERROR CODES

If for any reason the FLAG% variable is returned non-zero, then an error has occurred in the input of data to the CALL routine. Checking of data occurs first in the routine and no action will be taken if an error condition exists. An immediate return will take place with the error specified in the FLAG% variable. The only exception to this rule is error #3 (hardware failure or installation error) , where an attempt will be made to initialize the hardware even if there appears to be a problem so that other modes may possibly be run to diagnose the problem.

Following is a list of error codes:-

ERROR	FAULT
0	No error, operation successful.
1	Driver not initialized. Using another mode before initializing with mode 0.
2	Mode number out of range. Specifying MD% less than 0 or greater than 9.
3	Hardware error. Board not at specified I/O address, not in computer or A/D faulty.
4	Scan limits in wrong order (mode 1) scan low limit > scan high limit
5	Mode aborted by keyboard (modes 3, 4 & 5)
10 + N	Error in range of D%(N) Note:- Errors 10, 11, 12, 13 are caused by data range errors in D%(0), D%(1), D%(2), D%(3). For a description of the specific error, look up under mode number.

Error detection after the CALL routine is easily implemented:-

```
        xxx10 CALL DAS4 (MD%, D%(0), FLAG%)
        xxx20 IF FLAG% <> 0 THEN GOSUB yyyyy
or:-   (xxx20 IF FLAG% <> 0 THEN PRINT "Error ";FLAG% : STOP)
        . . .
        . . .
        yyyyy REM: Error handling subroutine
        . . .
        . . .
        zzzzz RETURN
```

This is useful while debugging a new program, or with a suitable error handling subroutine, can be left permanently in the program. See EX0.BAS for an example of an error capturing routine.

3.16 PROGRAMMING EXAMPLES FOR THE DRIVER MODES

A number of program examples are included on the floppy disk accompanying this manual. They include:-

1: EX0.BAS - Example of loading the DAS4.BIN driver at an absolute memory location and initializing with mode 0.

2: EX00.BAS - Example of loading the DAS4.BIN driver by contracting BASIC's workspace and initializing with mode 0.

3: EX2.BAS - Using mode 2 (single conversion) for A/D, also shows use of mode 1.

4: EX3.BAS - Example of doing A/D conversions direct to an integer array (modes 1 & 3).

5: EX4.BAS - Example of doing A/D conversions using interrupts (modes 1, 4, 6 & 7).

6: EX5.BAS - Example of using A/D trigger mode 5.

7: EX8.BAS - Example of digital input mode 8.

8: EX9.BAS - Example of digital output mode 9.

Since all of these programs are in BASIC and are heavily commented, they are readily listed to provide a better understanding of how to program DAS-4. They can also be edited and adapted to provide the basis for your own programs.

3.17 OTHER PROGRAMS AND UTILITIES

In addition to the examples above, the DAS-4 distribution disks includes:-

1: DAS4.BIN - The DAS-4 driver routine

2: DAS4.OBJ - The object file of DAS-4 for compilers

3: DAS4.ASM - The DAS4.BIN assembly source listing. Use TYPE DAS4.ASM to display this or load into your favorite word processor.

4: HOWTO.BIN - Enter TYPE HOWTO.BIN to find out how to modify and re-assemble DAS4.BIN from DAS4.ASM (mainly of interest to assembly language programmers)

5: INSTALL.EXE - Base address switch setting aid.

6:	STRIP.EXE STRIP.BAS	-	Example of a "real time" strip chart program compiled from BASIC source STRIP.BAS.
7:	CAL.BAS CAL.EXE	-	Calibration and test program (compiled version)
8:	LOG.BAS	-	A simple low speed data logging program that writes an ASCII disk file (suitable for import to Lotus 1-2-3 etc.)
9:	README.DOC	-	ASCII text file providing information on any additional undocumented programs.

From time to time, MetraByte may add other utilities and examples to the disk that are not documented in this manual. For explanations enter TYPE README.DOC after the DOS prompt for the latest information.

3.18 ASSEMBLY LANGUAGE PROGRAMS AND CALLS IN OTHER LANGUAGES

To facilitate the use of the I/O driver CALL routines in other languages e.g. C etc. the assembly object code file DAS4.OBJ is provided. This was assembled using the IBM or Microsoft Macro Assembler and may be linked to other object modules from compilers etc. When using the linker, the routine's public name is DAS4 (see next section concerning use of BASIC COMPILER). The fully commented source code and listings of the CALL routines will be found on the utility disk in file DAS4.ASM. Simply run this through the Macro Assembler (MASM DAS4) to generate a full listing (DAS4.LST). This material is mainly of interest to assembly language programmers who will find the source listing an excellent starting point for adapting or customizing routines to special requirements. The process for generating a BLOADable DAS4.BIN file is described in HOWTO.BIN, simply enter TYPE HOWTO.BIN from the DOS prompt (pressing Ctrl-PrtScreen will give you a hard copy too). If your programming is entirely in BASIC or BASIC COMPILER, you most probably will never need to refer to the source listing.

3.19 A NOTE ON EXECUTION TIMES - COMPILED BASIC

The throughput of the DAS-4 A/D converter is a function of many interactions but the fundamental limitation is the speed of the A/D which has a conversion time of 20 microseconds and the sample/hold which takes 10 microseconds or less to settle. This would indicate that you could perform conversions at 33KHz and this can in fact be accomplished with a tight assembly language loop. The various A/D conversion modes of the driver involve a lot of "housekeeping" and are much slower, about 3KHz is the maximum speed. Once inside the CALL, all the functions run at assembly language rates and are not affected by the BASIC interpreter, but other operations in your program processing the data between CALLs may produce bottlenecks and reduce your overall throughput.

One quick fix to improve the speed of an interpreted BASIC program is to compile it using the IBM Basic Compiler, Microsoft Basic Compiler or Microsoft Quick Basic[2]. When you compile a BASIC program the significance of the DAS4 in the CALL statement is no longer the same:-

```
xxx10  CALL DAS4 (MD%, D%(0), FLAG%)
```

DAS4 is not interpreted by the compiler as a variable. It becomes the public name of the subroutine that you wish to call. Before compiling your program, remove lines that BLOAD the DAS4.BIN routine and all DEF SEG statements that control the location of the routine. These are not required as the linker will locate the DAS4 routine in memory automatically. After compiling your program, run the linking session as follows:-

```
A> LINK yourprog.obj + das4.obj
```

DAS4.OBJ is on your distribution disk for this purpose. An example of the effect on performance of compiling a program is supplied in the programs STRIP.BAS and STRIP.EXE. This is a real time "strip chart" emulating program. The BASIC interpreter form, STRIP.BAS, is extremely slow, but when compiled (STRIP.EXE) using the Microsoft Quick Basic compiler speeds up to a point of being able to display about 30 points/second and give an effective real time display.

3.20 MULTIPLE DAS-4's IN ONE SYSTEM

What if you wish to operate more than one DAS-4 in a computer? To avoid conflicts, each DAS-4 must have a different base address and if interrupts are used, be connected to a different interrupt level, or if on a common level, each board's interrupt can only be enabled in turn, one at a time. Each board must also be assigned its own CALL routine. To do this start by loading the DAS4.BIN routine at different locations in memory:-

```
xxx10 DEF SEG = SG1
xxx20 BLOAD "DAS4.BIN",0
xxx30 SG2 = SG1 + 2048/16        'allow 2K for each routine
xxx40 DEF SEG = SG2
xxx50 BLOAD "DAS4.BIN",0
xxx60 SG3 = SG2 + 2048/16        'etc. for other boards
```

Now the CALL appropriate to each board can be entered as required. Note that each CALL is preceded by a DEF SEG appropriate to that board:-

```
yyy10 DEF SEG = SG1
yyy20 CALL DAS4 (MD%, X%, FLAG%)
yyy30 DEF SEG = SG2
yyy40 CALL DAS4 (MD%, X%, FLAG%)
                                          'etc.
```

---

2. Microsoft and Quick Basic are registered trademarks of the Microsoft Corporation, 10700 Northup Way, Box 97200, Bellevue, WA. 98009

Section 4

APPLICATIONS

4.1 CHANNEL INPUTS

There are 8 analog input channels on DAS-4. Each has an input range of -5.000v to +4.961v and are single ended i.e. they share a common low level ground. Input voltages should be applied between the channel Hi and any L.L. Gnd. Do not return inputs to the digital common (DIG.COM.) as this is intended as a heavy current return for power supplies and digital logic signals and may differ from the low level ground by many millivolts. Correct use of the grounds is very important to obtain consistent noise free measurements as it is easy to introduce inadvertent ground loops when using single ended connections. The low level grounds are used for all analog signal returns and when used correctly should only carry signal currents less than a few milliamps. The seven identical low level ground inputs have been positioned in the connector so that they lie between the analog channel inputs in the flat connecting cable, this helps to prevent cross talk. The input current of each channel is about 100 nanoamps at 25 deg. C. thus presenting a high input impedance to the signal. Also the 508A solid state channel multiplexer used on the DAS-4 is designed to withstand continuous overloads of +/- 32v on each channel and transient overloads of several hundred volts. This multiplexer has two other desirable characteristics, a "break before make" action to prevent shorts between channels while switching, and all channel switches turn off when the power is off thus preventing signal to signal shorts when your computer is off.

Channels are numbered 0 - 7 and the required channel is selected by the three lower bits of the control register byte. The current channel that the multiplexer is set to can be determined by reading the three lower bits of the status register. The A/D convert routines of modes 2, 3 and 4 of the CALL will automatically increment the multiplexer within the limits set by mode 1. If mode 1 is not specified scanning will default to the full range 0 - 7. A typical set up after loading and initializing (mode 0) DAS4.BIN would be:-

```
xxx10 MD% = 1 : D%(0) = 2 : D%(1) = 5
xxx20 CALL DAS4 (MD%, D%(0), FLAG%)
```

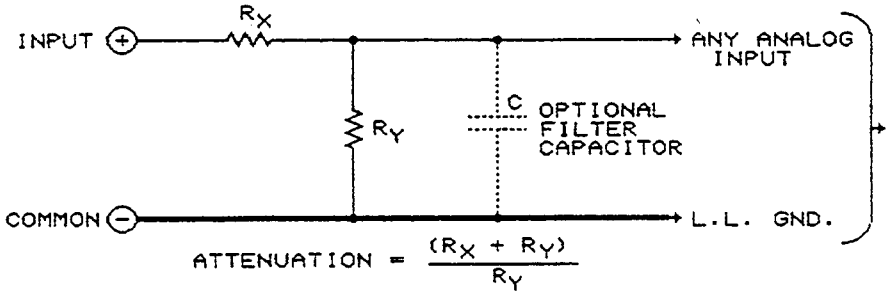
Line 20 enables scanning between channels 2 and 5, so the sequence would be 2, 3, 4, 5 - 2, 3, 4, 5 - 2, 3, 4, 5 - 2 etc.

```
xxx50 MD% = 2
xxx60 FOR I = 1 TO 100
xxx70 CALL (MD%, D%(0), FLAG%)
xxx80 PRINT USING "Channel # = ####";D%(1);D%(0)
xxx90 NEXT I
```

4.2 MEASURING VOLTAGE

Voltages in the range +/-5v may be directly applied to the analog inputs. Higher voltages should be attenuated, a simple resistive divider should be adequate as shown in Fig. 4.1.

Single ended inputs have a common ground return which is connected to the ground (case) of the computer. If you are measuring a signal which is floating i.e. has no connection to ground, there will be no problem but if your signal source is also connected to ground, then there is the potential for a ground loop which may cause an error or noise in your readings. There are several ways to avoid this complication, some of the solutions are shown in Figs. 4.2, 4.3 & 4.4. All of these methods provide you with a differential input which allows you to reject any small differences in ground potential between your computer and signal source.



TYPICAL VALUES

KEEP R_X LOW AROUND 1 - 10K

ATTENUATION	R_Y	R_X
X 2	10K	10K
X 10	90K	10K
X 100	99K	1K
X 1000	999K	1K

FIG. 4.1 SIMPLE INPUT ATTENUATOR

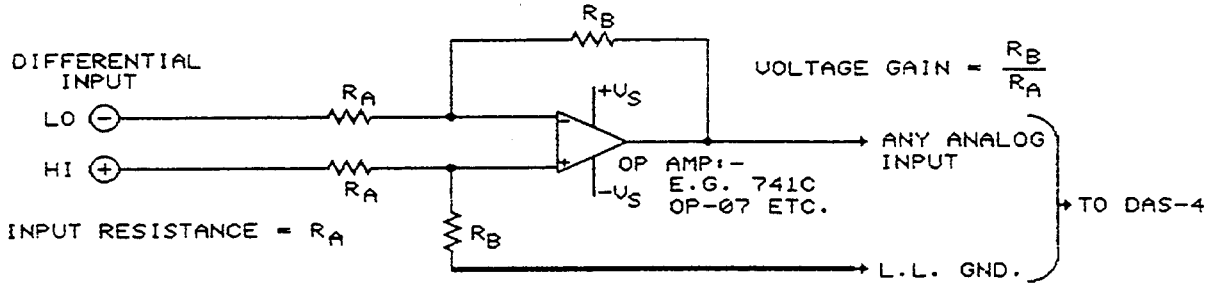


FIG. 4.2 SIMPLE DIFFERENTIAL AMPLIFIER

The circuit of Fig. 4.2 is the least expensive, but has the drawback of having an input resistance set by the input resistors. This may be quite large, in the 10Kohm to 100Kohm region, but may be too low for some applications. As an added benefit, the resistors may be chosen to provide gain or attenuation. This circuit is the classic differential connection for an operational amplifier and a full description can be found in any book on Operational Amplifiers[3].

---

3. See for instance "Operational Amplifiers - Design and Applications" by Tobey, Graeme & Huelsman. McGraw-Hill 1971.

Fig 4.3 is a variation on the circuit of Fig 4.2 and adds two voltage followers to this circuit to provide a very high input impedance for sensitive signals.

Finally if you want to buy a ready made differential amplifier, this part is available from integrated circuit manufacturers as a single component. In this form it is called an instrumentation amplifier, some types include gain setting resistors and others require external resistors. Instrumentation amplifiers are usually optimized for operation at high gains with small signals and usually have zero drifts of less than a few millionths (microvolts) per degree C.. Although more costly than simple operational amplifiers, operation under high gain conditions usually demands the extra stability and common mode rejection that instrumentation amplifiers provide.

If you are using the STA-08 screw connector accessory, there is a small bread board area with +/-12v & +5v power that can be used to accommodate attenuators, filters or amplifiers.

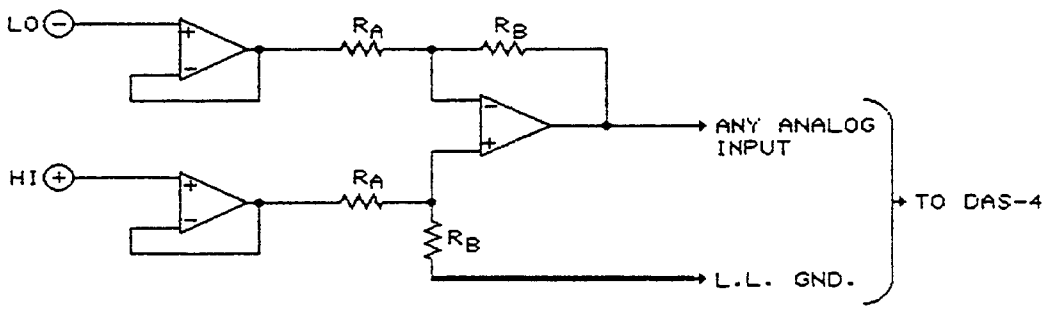
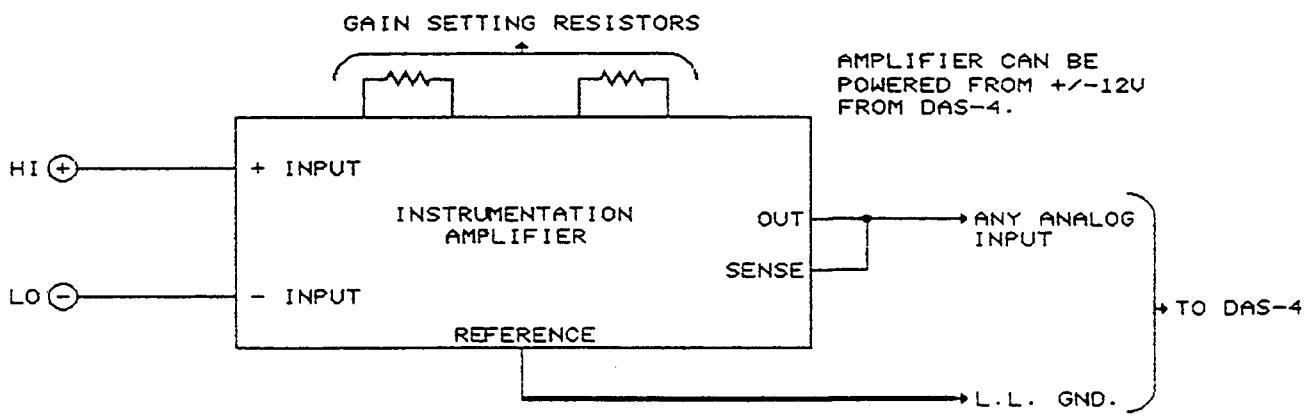


FIG. 4.3 HIGH INPUT IMPEDANCE DIFFERENTIAL AMPLIFIER



TYPICAL INSTRUMENTATION AMPLIFIERS:-

LM363	NATIONAL SEMICONDUCTOR
AMP-01	PRECISION MONOLITHICS
AD-524	ANALOG DEVICES
INA-102	BURR BROWN

FIG. 4.4 CORRECT CONNECTIONS FOR AN INSTRUMENTATION AMPLIFIER

For a ready made differential input with instrumentation amplifier, you can add the EXP-16 expansion multiplexer from MetraByte. This is designed to connect directly through flat cable to the DAS-4 and each EXP-16 provides 16 differential inputs and a common instrumentation amplifier connected to one of DAS-4's input channels. Up to 8 EXP-16's can be connected to one DAS-4 providing a total of 128 channels of differential analog input, each group of 16 channels being individually gain selectable. The EXP-16 is powered by the computer and the sub-multiplex channel is selected by the DAS-4 digital outputs (OP1-4).

The ISO-4 is a 4 channel isolation amplifier that is plug compatible with DAS-4. Up to 32 ISO-4's can be connected to one DAS-4. Each channel provides analog input isolation of up to 500 volts and gains ranging from 1 to 1000. See our catalog for detail specifications on the EXP-16 and ISO-4.

These various methods provide a variety of different interfacing solutions of different costs and complexities. Almost certainly, one of these will be appropriate for your requirements.

4.3 4-20mA CURRENT LOOPS

Process control current loop transducers are easily interfaced to DAS-4 by adding a suitable shunt resistor across the input. Since the maximum current will be 20mA and the maximum input range is +5v, a 250 ohm precision shunt resistor will be required. This should be of low temperature coefficient metal film or wire wound construction for stability with time and temperature. Using this interface, the 4-20mA working range of the current loop corresponds to 102 bits of input, a resolution of about 1%.

4.4 THE REFERENCE

The 5v A/D voltage reference (Vref) is brought out for users. It may be used for offsetting signals etc. but should not be heavily loaded. The maximum available output current is 5mA.

4.5 USING DIGITAL INPUT/OUTPUT

DAS-4 provides 4 TTL/DTL compatible digital outputs (OP1-4) and 3 TTL/DTL compatible digital inputs.

The digital outputs correspond to bits 4 - 7 of the control register and are accessed by writing to the control register (see section 3.1.5). When you write to the control register you will usually need to maintain the state of bits 0 - 3 that control the multiplexer address and interrupt enable. The DAS4.BIN routine does this for you by holding an "image" of the control register in temporary storage. If you are not using this routine, it may be advisable to set up a similar capability in your own software. The multiplexer address bits 0 - 2 can always be determined by reading the status register, so the only state that has to be stored is whether or not the interrupt is enabled.

Digital outputs can sink 8mA (5 standard TTL loads or 20 LSTTL loads). If you wish to interface to CMOS, 1Kohm pull-up resistors connected to +5v should be attached to the outputs.

This will raise the logic high output level from its minimum TTL level of 2.4v to +5v suitable for CMOS interface.

If EXP-16 or ISO-4 expansion interface(s) are used, then the digital outputs are usually fully committed to providing the sub-multiplex address.

Digital inputs are available through bits 4 - 6 of the status register (see section 3.1.4). The digital data is easily obtained by masking out these bits using a logical AND operation. The inputs present a -0.4mA loading corresponding to 1 LSTTL load.

4.6 ADDING MORE ANALOG INPUTS

You may add sub-multiplexers to any or all of the 8 analog inputs. MetraByte's EXP-16 provides 16 channels per input and is directly plug compatible with DAS-4. Up to 8 EXP 16's can be added to one DAS-4 providing a total of 128 channels. The sub-multiplexer address is set by digital outputs OP1-4 so that a typical scan would be to use Mode 2 and then increment the sub-multiplexer address with Mode 9 and repeat the scan of Mode 2 on the next set of sub-multiplex channels etc.

The EXP-16 cards are designed to cascade with flat cable and connectors similar to that used to connect to DAS-4. One cable should be provided for each EXP-16. All analog channel connections are made by screw connectors, and each EXP-16 (group of 16 channels) can be operated at a different gain. In this way a system can be configured with a variety of different channel functions and gains, single ended and differential.

4.7 INTERFACE TO TRANSDUCERS, THERMOCOUPLES ETC.

Low level transducers such as thermocouples and strain gage bridges (load cells, pressure & force transducers) require amplification before applying to the high level DAS-4 inputs. The EXP-16 expansion multiplexer incorporates an instrumentation amplifier that can provide stable amplification and also includes circuitry that allows cold junction compensation of thermocouples. EXP-16 will handle most interfacing requirements to D.C. output transducers and also includes spaces for filters, shunts and attenuators. In general, the limited 8 bit resolution of DAS-4 will not give very precise results when used with thermocouples, MetraByte's 12 bit DAS-8 will give better performance. Most thermocouples (J,K,T & E characteristics) produce about 40 microvolts/deg C.. Amplified by a gain of 1000, this corresponds to 40 millivolts/deg C. and since the resolution of the DAS-4 is 39 millivolts/bit, each bit will correspond to 1 deg. C. which for many purposes is a coarse resolution. This is why a 12 bit converter with 16 times better resolution can will give better results. However, for alarm or threshold sensing, the DAS-4 may be adequate when used with a thermocouple, or if the accuracy is not very demanding.

For inexpensive temperature measurement in the -50 to +125 deg. C. temperature range, semiconductor temperature transducers are a good choice. The most popular types are the AD590 (Analog Devices) which behaves like a constant current source with an output of 273uA at 0 deg.C. and a scaling of 1uA/deg.C. and the LM335 (National Semiconductor) that has an output of 2.73 volts at 0 deg.C. and a temperature coefficient of 10mV/deg.C.. Both of these devices can be powered from the +12v available from the computer and interfaced to DAS-4

either directly or through a simple operational amplifier. These can give much better resolution than a thermocouple for ambient temperature measurements, and are easier to interface.

4.8 POWER OUTPUT FROM THE DAS-4 CONNECTOR

The +5v and +/-12v I.B.M. P.C. bus supplies are available on the DAS-4 rear connector. These are provided as a convenience to users who wish to add external signal conditioning and logic circuits but they should be used with caution and an awareness that these same supplies are connected to the internal circuitry of your PC. The +/-12v can be used for analog circuits, operational amplifiers, comparators, indicators relays etc. and the +5v will power logic circuits, TTL, CMOS etc. Careful use of these supplies can often avoid the expense and bulk of external supplies to power your signal sources and the nuisance of multiple power switches.

The available power will depend on the type of computer and what other peripheral cards are installed. Consult your users technical reference manual for details on the power supply specification. Due to cabling and connector limitations, it is recommended that you limit current draw to less than 1 amp from any supply (the -12v supply is usually limited to a few 100 milliamps). Most analog circuits will consume a few tens of milliamps which is negligible.

If the power outputs are subjected to an over current (overload) or over voltage condition, most power supplies are designed to shut down and the computer will have to be turned off and turned on again to restore operation after removing the fault. Although these protective devices are built into your computer supply, do not depend on them, use your computer power with care and consideration. ***If there is any possibility of frequent short circuits or shorts to high voltages and signal sources which could damage your whole computer, then it is strongly advisable to provide an external (and more easily repaired) power supply for your user circuits.***

4.9 PRECAUTIONS IN USE - NOISE, GROUNDLOOPS AND OVERLOADS

Unavoidably, data acquisition systems give users access to inputs to the computer. Do NOT, whatever else you do, get these inputs mixed up with the A.C. line. An inadvertent short can in an instant can cause extensive and costly damage to DAS-4 as well as your computer. MetraByte can accept no liability for this type of accident. As an aid to avoiding this problem:-

1. - Avoid direct connections to the A.C. line.

2. - Make sure that all connections are tight and sound so that signal wires are not likely to come loose and short to high voltages.

3. - Use isolation amplifiers and transformers where necessary.

There are two ground connections on the rear connector called DIG. COM. and L.L. GND. Digital common is the noisy or "dirty" ground that is meant to carry all digital signal and heavy current (power supply) currents. Low level ground is the signal ground for all analog input functions. It is only meant to carry signal currents (less than a few mA) and is the ground

reference for the A/D channels. Due to connector contact resistance and cable resistance there may be many millivolts difference between the two grounds although they are connected to each other and the computer and power line grounds on the DAS-4 board.

Section 5

CALIBRATION AND TEST

5.1 CALIBRATION AND TEST

The 8 bit resolution and 39 millivolt bit size of the DAS-4 is so coarse that once correctly set, units will most likely never require further calibration. If you wish to check the calibration, run CAL.EXE from DOS, or CAL.BAS from BASIC(A). You should connect a voltage calibrator or multimeter with a variable D.C. source attached connected to one or more of the inputs, the CAL program will guide you through the necessary adjustments. Operation of the digital outputs and inputs can also be checked from the calibration program. The DAS4 calibration potentiometer locations are shown in Fig. 5.1.

5.2 SERVICE AND REPAIR

MetraByte products are warranted for 1 year against defects in workmanship and material. Most data acquisition products are susceptible to damage from external over voltages such as connecting inputs or grounds to line voltages. We hope you will take precautions to avoid this sort of calamity but if you do, there are a couple of remedies.

The integrated circuits that provide the digital input and output and the analog multiplexer are installed in sockets. In the event that you apply an over voltage and succeed in "zapping" an input or output, you can replace these inexpensive parts yourself if you wish.

Alternatively, MetraByte can service your DAS-4, simply call and ask for customer service. We will issue you with a Return Material Authorization (R.M.A.) number which should be attached to any packages returned to MetraByte. Please do not return material without an R.M.A. number as it greatly complicates the handling of the material and paperwork. If the fault is not obvious, it will help our repair technicians if you include a brief note describing the problem and under what conditions it occurs.

5.3 TECHNICAL ASSISTANCE

MetraByte has a staff of technical assistance engineers. Should you have a problem or require help or advice on using any MetraByte product, call Technical assistance is free of charge and is available on Mondays thru Fridays from 9:00 a.m. to 5.00 p.m. E.S.T.

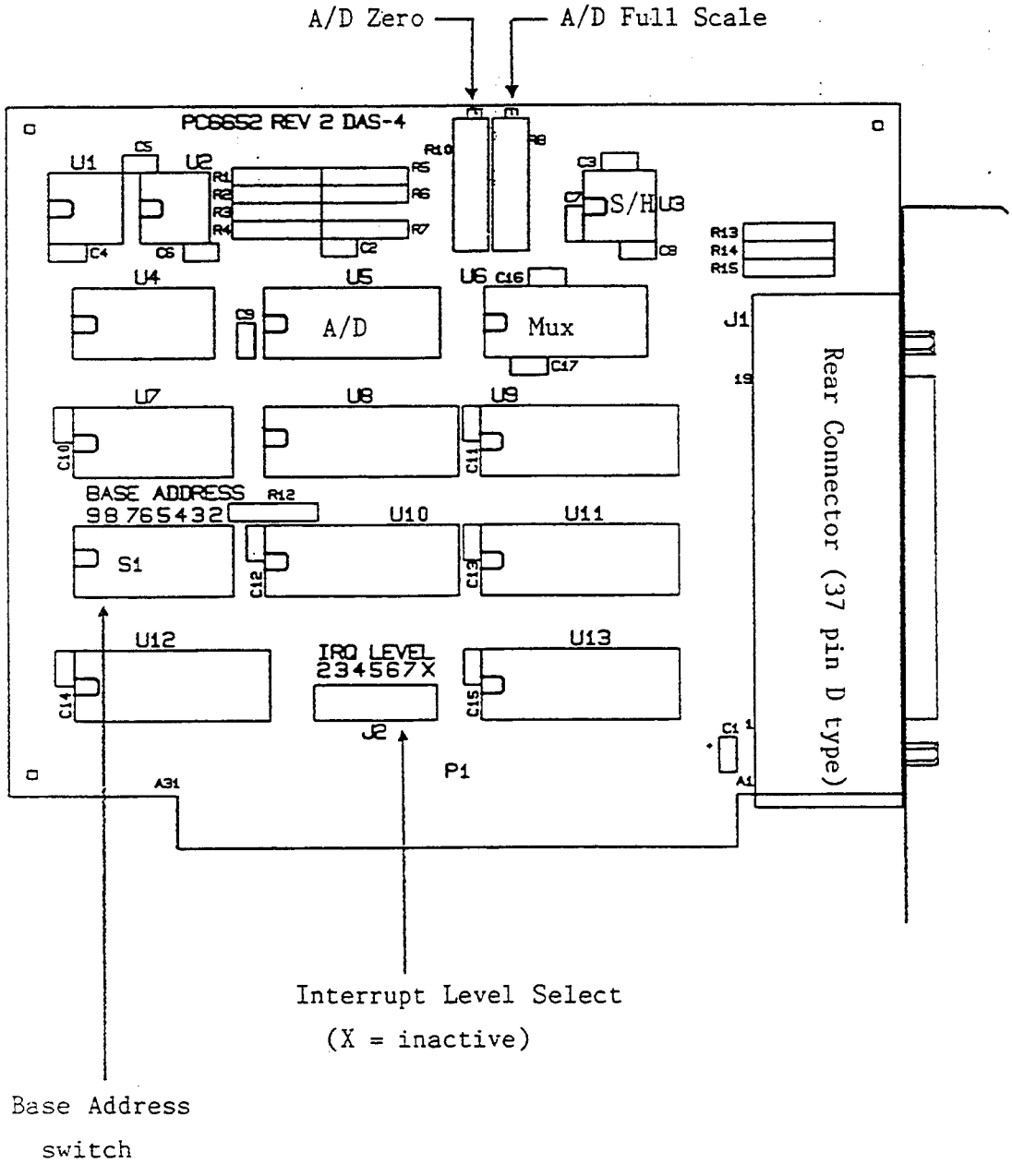


Fig. 5.1 DAS-4 ADJUSTMENTS, JUMPERS AND SWITCH LOCATIONS

Appendix A

CONNECTIONS

A.1 MAIN I/O CONNECTOR

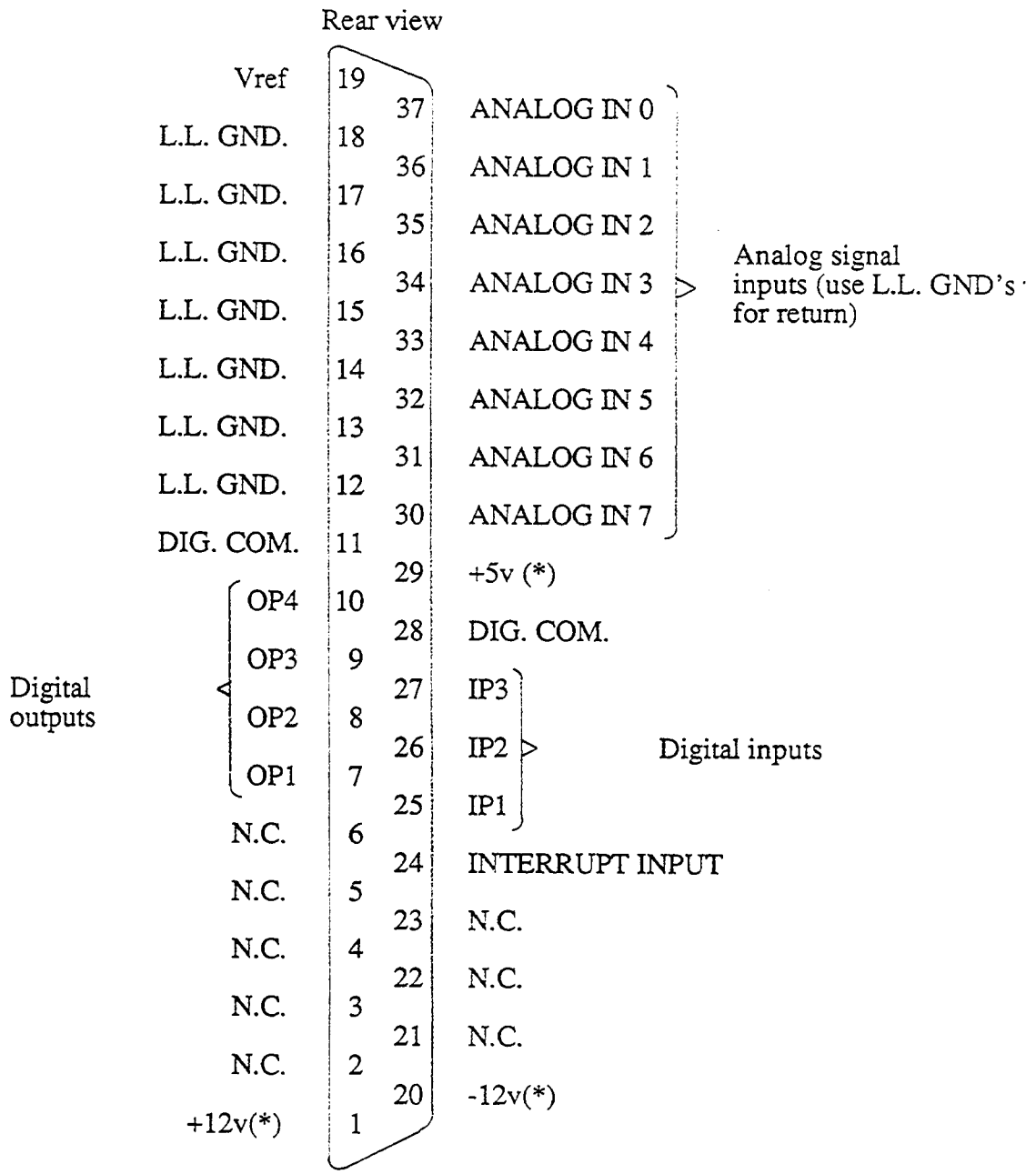
The main analog and digital I/O is via a 37 pin D type connector that projects through the computer case at the rear of the board. The pin functions are as follows (see Fig A.1 for locations):-

PIN	NAME	FUNCTION
1	+12v	+12v power supply from P.C. bus (observe loading limits)
2	-	No connection
3	-	No connection
4	-	No connection
5	-	No connection
6	-	No connection
7	OP1	Digital output #1
8	OP2	Digital output #2
9	OP3	Digital output #3
10	OP4	Digital output #4
11	DIG.COM.	Digital Common. Return for all logic signals and power supply currents. Connected to computer frame.
12	L.L.GND.	Low level grounds. These are common returns and shields for the analog input channels.
13	L.L.GND.	
14	L.L.GND.	
15	L.L.GND.	
16	L.L.GND.	
17	L.L.GND.	
18	L.L.GND.	

19	VREF	+5.00v(+/-0.2v) precision reference voltage from A/D.
20	-12v	-12v power from P.C. bus. (observe loading limits).
21	-	No connection
22	-	No connection
23	-	No connection
24	INT. IN.	Interrupt input. Positive edge triggered input.
25	IP1	Digital input #1
26	IP2	Digital input #2
27	IP3	Digital input #3
28	DIG. COM.	Digital common. (same as pin 11).
29	+5v	+5v power from P.C. bus (observe loading limits).
30	IN 7	Channel #7 analog input
31	IN 6	Channel #6 analog input
32	IN 5	Channel #5 analog input
33	IN 4	Channel #4 analog input
34	IN 3	Channel #3 analog input
35	IN 2	Channel #2 analog input
36	IN 1	Channel #1 analog input
37	IN 0	Channel #0 analog input

The mating connector for DAS-4 is a standard 37 pin D type female such as ITT/Cannon #DC-37S for soldered connections. Insulation displacement (flat cable) types are readily available e.g. Amp #745242-1.

A.2 REAR VIEW OF DAS-4 CONNECTOR



* - These are power outputs from the computer.

Appendix B

SPECIFICATIONS

B.1 POWER CONSUMPTION

+5v supply	-	170mA typ. / 240mA max.
+12v supply	-	7mA typ. / 11mA max.
-12v supply	-	7mA typ. / 11mA max.

B.2 ANALOG INPUT SPECIFICATIONS

8 analog input channels each with the following specification:-

Resolution	-	8 bits. (39mV/bit)
Accuracy	-	0.2% of reading +/-1 bit.
Full scale	-	+/-5 volts
Coding	-	Offset binary
Over voltage	-	Continuous single channel to +/-35v
Configuration	-	Single ended.
Input current	-	100nA max at 25 deg.C.
Temperature Coefficient	-	Gain or F.S., +/-50ppm/deg.C. max. Zero, +/-10 microvolt/deg.C. max.

B.3 A/D SPECIFICATION

Type	-	Successive approximation.
Resolution	-	8 bits
Conversion time.	-	20 microseconds typ. 30 microseconds max.

Monotonicity	-	Guaranteed over operating temperature range.
Linearity	-	+/-1 bit.
Zero drift	-	10ppm/deg. C. max.
Gain drift	-	50 ppm/deg. C max.

B.4 SAMPLE HOLD AMPLIFIER

Acquisition time	-	10 microseconds to 0.1% typ. for full scale step input
Dynamic sampling error	-	2mV @ 2000v/sec.

B.5 REFERENCE VOLTAGE OUTPUT

Reference voltage	-	+5.0v +/- 0.2v
Temperature coefficient	-	50 ppm/deg.C max.
Load current	-	+/-5mA max.

B.6 DIGITAL I/O

OP1-4 output low voltage	-	0.5v max at Isink = 8.0mA
OP1-4 output high voltage	-	2.7v min at Isource = -0.4mA
IP1-3 input low voltage	-	0.8v max
IP1-3 input low current	-	-0.4mA max
IP1-3 input high voltage	-	2.0v min
IP1-3 input current	-	20uA max. @ 2.7v

B.7 INTERRUPT INPUT

Type	-	Positive edge triggered
Level	-	2 - 7 jumper selectable
Enable	-	Via INTE of CONTROL register

Interrupts are latched in an internal flip-flop on the DAS-4 board. The state of this flip-flop corresponds to the INT bit in the STATUS register. The interrupt flip-flop is cleared by a write to the CONTROL register. Service routines should acknowledge and re-enable the interrupt flop.

B.8 POWER OUTPUTS

IBM P.C. bus supplies	-	+5v & +/-12v
Tolerance	-	+5v +/-5% +12v +/-5% -12v +/-10%
Loading	-	Dependent on other peripherals (see Section 4.8)

B.9 GENERAL ENVIRONMENTAL

Operating temperature range.	-	0 to 50 deg. C.
Storage temperature range	-	-20 to +70 deg.C.
Humidity	-	0 to 90% non-condensing.
Weight	-	4 oz. (120 gm.)

Appendix C

STORAGE OF INTEGER VARIABLES

Data is stored in integer variables (% type) in 2's complement form. Each integer variable uses 16 bits or 2 bytes of memory. 16 bits of data is equivalent to values from 0 to 65,535 decimal, but the 2's complement convention interprets the most significant bit as a sign bit so the actual range becomes -32,768 to +32,767 (a span of 65,535). Numbers are represented as follows:-

	High byte								Low byte							
	D7	.	.	.	.	.	.	D0	D7	.	.	.	.	.	.	D0
+32,767	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
+10,000	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0
+1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-10,000	1	1	0	1	1	0	0	0	1	1	1	1	0	0	0	0
-32,768	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

^
Sign bit (1 if negative, 0 if positive)

Integer variables are the most compact form of storage for the 12 bit data from the A/D converter and 16 bit data of the 8254 interval timer and so to conserve memory and disk space and optimize execution speed, all data exchange via the CALL is through integer type variables. This poses a programming problem when handling unsigned numbers in the range 32,768 to 65,535.

If you wish to input or output an unsigned integer greater than 32,767 then it is necessary to work out what its 2's compliment signed equivalent is. As an example, assume we want to load a 16 bit counter with 50,000 decimal. An easy way of turning this to binary is to enter BASIC and execute PRINT HEX$(50000). This returns C350 or binary 1100 0011 0101 0000. Since the most significant bit is 1 this would be stored as a negative integer and in fact the correct integer variable value would be 50,000 - 65,536 = -15,536. The programming steps for switching between integer and real variables for representation of unsigned numbers between 0 and 65,535 is therefore:-

From real variable N (0 <= N <= 65,535) to integer variable N%:-

```
xxx10 IF N<=32767 THEN N% = N ELSE N% = N - 65536
```

From integer variable N% to real variable N:-

```
xxx20 IF N% >= 0 THEN N=N% ELSE N = N% + 65536
```

Index

E

G

I

L

M

O

P

R

S

T

V

W