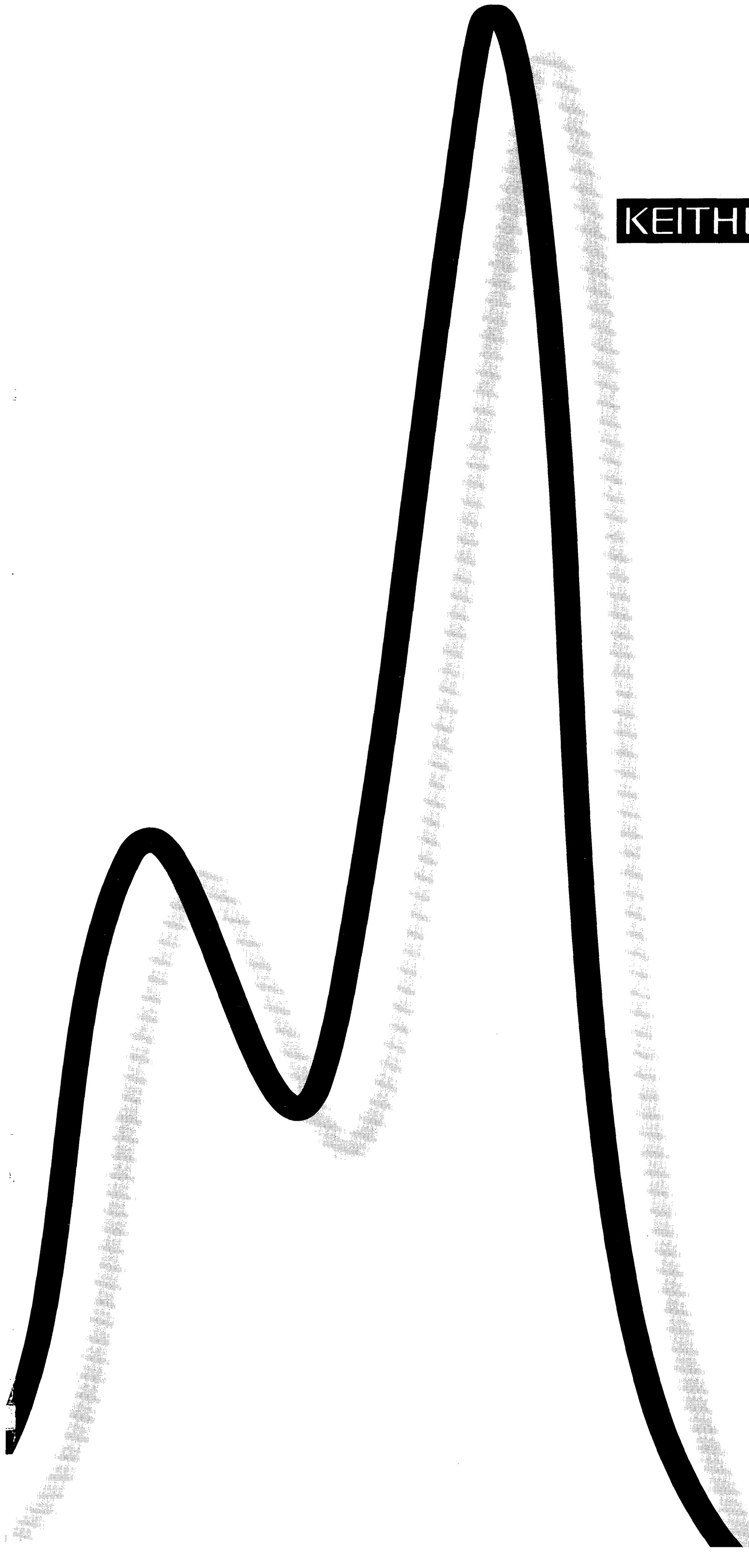


KEITHLEY METRABYTE

DAS-20

USER'S GUIDE

DAS-20

Keithley Instruments, Inc. Data Acquisition Division

❑

❑

❑

User Guide

for the

DAS-20

A/D & D/A Data Acquisition Boards

Revision D - April 1993

Part Number: 24855

Keithley Data Acquisition

440 MYLES STANDISH BLVD., Taunton, MA 02780

TEL. 508/880-3000, FAX 508/880-0179

Warranty Information

All products manufactured by Keithley Instruments, Inc. Data Acquisition Division 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 the manufacturer, be repaired or replaced. This warranty does not apply to products damaged by improper use.

Warning

Keithley Instruments, Inc. Data Acquisition Division assumes no liability for damages consequent to the use of this product. This product is not designed with components of a level of reliability suitable for use in life support or critical applications.

Disclaimer

Information furnished by Keithley Instruments, Inc. Data Acquisition Division is believed to be accurate and reliable. However, the Keithley Instruments, Inc. Data Acquisition Division assumes no responsibility for the use of such information nor for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent rights of the Keithley Instruments, Inc. Data Acquisition Division.

Copyright

Note:

Contents

Contents

■ ■ ■

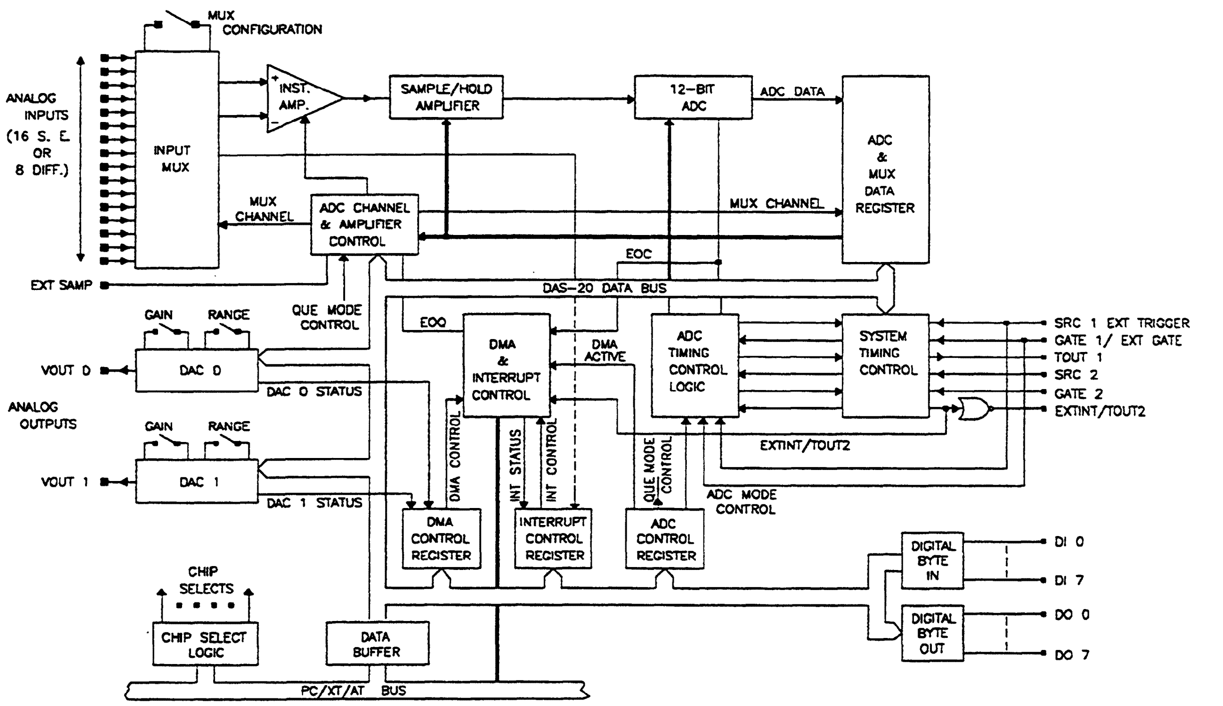
CHAPTER 1

INTRODUCTION

1.1 GENERAL

MetraByte's DAS-20 (see the following block diagram) is a multifunction, high-speed, A/D (Analog/Digital), I/O expansion board that turns a host computer into a precision data-acquisition and signal-analysis instrument. The board plugs directly into any expansion slot of an IBM PC/XT/AT, or PS/2 Models 25 and 30, or compatibles.

Using state-of-the-art data-conversion and digital-interface components, DAS-20 design offers high data throughput, high accuracy, and low noise. Multilayer board construction with integral ground planes, careful attention to board layout, and an on-board DC/DC power supply ensure low-noise analog measurements and analog outputs even at high data-transfer rates. As with all MetraByte products, each DAS-20 goes through rigorous inspection, test, and burn-in.



DAS-20 Block Diagram

DAS-20 features include the following:

- Analog Inputs.
- Analog Outputs.
- Counter Timers.
- Parallel Digital I/O Bits.
- Comprehensive Distribution Software.

The listed features are described in the following subsections.

Analog Inputs

Channel capacity of the DAS-20 is switch selectable for

- 16-channel, single-ended input
- 8-channel, differential input

Seven software-selectable input ranges include Unipolar and Bipolar configurations. Selectable gains range from X0.5 to X100 with input resolutions between 4.88mV and 24.4µV.

ADC (A/D Converter) functions use an AD774 successive-approximation converter with an 8.5µs conversion rate. The 8.5µs rate, combined with a 1400ns sample/hold time, allows the DAS-20 to perform just over 100,000 conversions per second. In the DMA data-transfer mode on a standard PC or XT compatible, data can be written to memory at the full 100,000 sample-per-second rate. On the PC/AT and some other AT-compatible machines, data transfer may be slightly slower (worst case: down to 65,000 samples per second), depending on the system clock speed and the number of wait states used by the computer. However, most AT-compatible machines are capable of more than 100,000 samples-per-second.

ADC triggering uses any one of three drives, as follows: (1) a software command, (2) the on-board counter/pacer clock, or (3) an external pulse or clock. Data transfer also uses any one of three drives: (1) a software/program control, (2) a PC interrupt (and a corresponding interrupt service routine), or (3) Direct Memory Access (DMA). The following table details possible data throughput rates of the various transfer types.

OPERATING MODE	THROUGHPUT (CONVERSIONS/S)
Data transfer to simple variable	200
Data transfer to Array variable	4000
Interrupt drive Transfer	4000
DMA transfer	100,000

The DAS-20 board has on-board queuing RAM that allows the creation of a Channel/Gain Sequence List. The board will automatically follow the Sequence. A typical Sequence List stores the following information:

SAMPLE #	CHANNEL #	GAIN
0	0	X 1
1	5	X 100
2	1	X 10
3	5	X 100
4	2	X 1
5	5	X 100
6	10	X 10
7	4	X 1
.	.	.
.	.	.
.	.	.
N	Return to SAMPLE # 0	

Where N is from 1 to 2047.

A Sequence List allows different channels to be sampled at different rates, under full hardware control. Hardware control is important for two reasons: first, it puts system timing under the DAS-20 Pacing Clock; second, it allows the computer to run software-controlled tasks in the foreground while the DAS-20 is acquiring data in the background. When system timing runs under the DAS-20 Pacing Clock rather than software control, it is not subject to interruption by other system functions. In addition, control by the Pacing Clock allows data-point sampling over uniform, jitter-free intervals (this is crucial if any type of frequency analysis is to be performed).

Analog Outputs

The DAS-20 provides two channels of analog output. D/A conversion in each channel has a 12-bit (one part in 4096) resolution and is switch-selectable for 0-10V, ±5V, or ±10V output ranges. The DACs (D/A Converters) are also double-buffered, assuring that no intermediate transients occur during output updates. The DACs can operate under software control to accommodate applications with a low rate of output updating, or they can operate in a DMA mode when a high-frequency output is required. The maximum update rate is 130,000 samples per second for one or both DACs.

Counter Timers

The DAS-20 includes an on-board, 5-channel counter-timer (AMD-9513A). Three of the counter-timers are driven by a 5MHz crystal-controlled oscillator to control A/D-converter sample timing. The remaining two counter-timers perform more complex A/D sampling schemes to pace the D/A converters, or they connect to external signals for use as frequency or pulse generators or for measuring frequency, pulse widths, or count events.

Parallel Digital I/O Bits

Digital I/O on the DAS-20 includes 16 TTL-compatible lines divided into one 8-bit output port and one 8-bit input port. These lines are useful for a variety of external triggering, switch sensing, and instrument control applications. The digital outputs are particularly important since they are used as controls by the EXP-20, and MB-02 accessory boards.

1.2 SPECIFICATIONS

Analog Input:

Channels	8 differential or 16 single-ended, switch selectable with software readable status.
Resolution	12 bits.
Accuracy	0.01% of reading ±1 LSB.
Input Ranges (Software Selectable)	0 to +10V ±10V ±5V 0 to +1V ±0.5V 0 to 100mV ±50mV
Coding	Left justified two's complement (bipolar). Left justified true binary (unipolar).
Overvoltage	Continuous single channel to ±35V.
Input current	1 nA max at 25 °C.
Input Impedance	Greater than 100 MΩ.

Instrumentation amplifier:

INPUT RANGE	SETTLING TIME
0-10V	5μS
±10V	5μS
±5V	5μS
0 to +1V	7.5μS
±0.5V	7.5μS
0-100mV	20μS
±50mV	20μS

A/D Converter:

Type	Successive Approximation.
Resolution	12 bits.
Conversion Time	8.5µS (typ); 9.0µS (max).
Monotonicity	Guaranteed over temperature range.
Linearity	±1 LSB.
Zero Drift	5 PPM/°C.
Gain Drift	45 PPM/°C.
Trigger Sources	Software command, timer generated, or external with programmable edge.
Gate Sources	Internal or external with programmable level.

Sample & Hold Amplifier:

Acquisition Time	1.0 to 1.5 µS to 0.01% accuracy.
Aperture Uncertainty	0.3 nS (typ).

Analog Outputs:

Channels	2 independent.
Type	12-bit, non-multipying, double-buffered.
Linearity	±1/4 LSB (typ), ±1/2 (max).
Monotonicity	Guaranteed Over Temperature.
Output Ranges	0 to +10V ±5V ±10V
Coding	Right justified true binary (unipolar); Right justified two's compliment (bipolar).
Output Drive	±5mA.
Output Resistance	0.2Ω.
Reference	Built in 6.3V, 10 PPM/°C.
Range Select	Switch-selectable/software-readable.
Settling Time	10V range: 3µS (typ), 4µS (max); 20V range 2µS (typ), 3µS (max).

Data transfer	Single Write or DMA to either or both modes.

Interrupt Capabilities:

Levels	IRQ2 thru IRQ7 software programmable.
Enable	Software programmable.
Sources	ADC end of conversion, End of ADC Que, Timer 2 terminal count, or DMA terminal count.

DMA Capabilities:

Levels	1 or 3 software programmable.
Enable	Software programmable.
Termination	By interrupt on terminal count or autoinitialize.
Transfer Modes	ADC data to host, from host to either or both DACs paced by Timer 2.
Transfer Rate	100 Kwords/Sec (ADC) (XT machine). 130 Kwords/Sec (DACs).

Digital I/O:

Output Port	8-bit latched with readback Low = 0.5V max, Isink = 8.0mA, High = 2.4V min, Isource = -0.4mA.
Input Port	8-bit transparent latch;Low = 0.8V max, -0.4mA max; High = 2.0V min, 20µA max.

Counter Timer:

Type Counters	AMD 9513A System Timing Controller; (5) 16 bit multimode programmable counter; Counters 3, 4, and 5 used to delay ADC start of conversion. Counter 2 used to pace DACs during DMA or Real time clock interrupt. Counters 1 & 2 are also usable externally.
Internal Time	5MHz, ±30 PPM/°C Timebase.
Output Drive	IOL = 3.2mA, IOH = -200µA.
Inputs, Gate, & Source	CMOS, TTL, CMOS-compatible.
Max Input Freq. Active Gate	5MHz user-definable software, level, or edge.
Minimum Ext Clock Pulse Width	70 nS.
Timer Range	5MHz to Real Time (24Hr).

Power Supply Available At Connector:

PC Bus	+5V.
Tolerance	±5%
Loading	1 Amp or less is recommended with 1.5 Amp as an absolute Max.

Power Consumption:

+5 Volt	1.6 amp (typ), 1.8 Amp (max).
+12 Volt	Not used.
-12 Volt	Not used.

General Environmental:

Operating Temp	0 to 50 °C.
Storage Temp	-20 to +70 °C.
Humidity	0 to 90% non-condensing.
Weight	14 Ounces (.396 kg).

1.3 DISTRIBUTION SOFTWARE

DAS-20 comes complete with a Distribution Software package that includes the following:

1. A Machine Language Driver that greatly simplifies DAS-20 programming. All DAS-20 functions are accessible by simple *Call* functions. The Driver supports interpreted and compiled BASIC programs. Other drivers for C, FORTRAN, and PASCAL are available as options From MetraByte.
2. Example and demonstration programs.
3. Initial installation and setup routines.
4. Calibration and test routines.
5. Thermocouple and other sensor linearization subroutines.

Distribution Software files are as follows:

Programming Examples for the Driver Modes

Programming examples on the floppy disk accompanying this manual are as follows:

EX0.BAS	This example illustrates two methods for loading DAS20.bin into the Basica environment. In addition it illustrates the initialization mode (MODE 0)
EX3.BAS	This program illustrates single A/D conversions using Mode 3.
EX4.BAS	Mode 4 is used in this program to demonstrate the loading of a Basica array with data from the ADC under program control. Mode 24 (Set/Start ADC PACER) is used to set conversion rate of ADC. If the ADC Pacer was set then Mode 26 is used to shut down the ADC PACER.
EX5.BAS	Mode 5 is used again to fill an absolute memory array with ADC data under interrupt control. Once this has been done Mode 13 is used to retrieve the ADC Data into one Basica Array and the channel data in the second Basic Array. If continuous mode was selected then Interrupt operation is shut down using Mode 11.
EX6.BAS	Like EX3.BAS except data is DMA'd directly into memory using Mode 6. If continuous mode was selected then DMA operation is shut down using Mode 11.
EX7.BAS	This program demonstrates single DAC outputs (Mode 7).
EX8.BAS	In this sample program data is moved to an absolute segment (Mode 8) and then output to the DAC under Interrupt control (Mode 9). Mode 25 is used to set up DAC pacer time base.

EX10.BAS	Like EX7.BAS except DAC IO is performed under DMA control Mode(10).
EX14.BAS	This program demonstrates the digital output and input Modes 14 and 15.
EX16.BAS	Analog Trig Mode(16).
EXP20.BAS	Demonstrates the use of the EXP20 with the DAS-20 (Mode 28).
EX23.BAS	Frequency Measure Mode(23). Program to demonstrate frequency measurement.
EX27.BAS	Block Scan Mode(27). This program illustrates Block Scan Mode.
SSH4.BAS	Illustrates the use of the SSH4 with the DAS20 (Mode 29).

These programs are also furnished in Quick BASIC; file names are prefixed with QB (for example, the Quick BASIC version of EX8.BAS is QBEX8.BAS). Since these programs are in BASIC and heavily commented, they are readily listed to provide a better understanding of how to program DAS-20. They can also be edited and adapted to provide the basis for your own programs.

Other Programs and Utilities

In addition to the examples in Section 4.4.1, the DAS-20 distribution disks include:

1. DAS20.BIN - The DAS-20 driver routine readable with GW BASIC.
2. DAS20.OBJ - The object file of DAS-20 for compilers.
3. DAS20.ADR - An ASCII file containing the DAS-20 base I/O address (default setting &H300 or 768).
4. DAS20.QLB - DAS-20 driver software compilede in a Quick BASIC library.
5. CAL20.EXE - Installation and Calibration program for DAS-20 (and SSH-4 and EXP-20). This program will also be very useful as a hardware debugging aid.
6. MAKEBIN.EXE - For making GW BASIC files.
7. HOWTOBIN.HLP - Help in making GW BASIC files.
7. INSTALL.EXE - DAS-20 installation program.
8. LEV3.EXE - Checks hard disk use of DMA level 3.
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. If these additions are not documented in this manual, enter TYPE README.DOC after the DOS prompt for the latest information.

* * * * *

CHAPTER 2

INSTALLATION & SETUP

2.1 GENERAL

This chapter provides instructions for the installing the DAS-20 in an IBM PC/XT/AT or compatible model. The chapter begins with procedures for unpacking, handling, and inspection; it follows with instructions for making working copies of your DAS-20 Distribution Software. Next are descriptions of the options and methods for setting all configurable parameters. Finally, there are instructions for installing the board.

2.2 UNPACKING & INSPECTING

After you remove the wrapped board from its outer shipping carton, proceed as follows:

1. Place one hand firmly on a metal portion of the computer chassis (the computer must be turned Off and grounded). You place your hand on the chassis to drain off static electricity from the package and your body, thereby preventing damage to board components.
2. Allow a moment for static electricity discharge; carefully unwrap the board from its anti-static wrapping material.
3. Inspect the board for signs of damage. If any damage is apparent, return the board to the factory.
4. Check the contents of your package against its packing list to be sure the order is complete. Report any missing items to MetraByte immediately.

You may find it advisable to retain the packing material in case the board must be returned to the factory.

2.3 BACKING UP THE DISTRIBUTION SOFTWARE

Distribution software is furnished on 5.25", 360K floppy diskettes and is not copy-protected. To accommodate users with 3.5" floppy drives, the Software is also available on a 720K diskette.

As soon as possible, make a working copy of your Distribution Software using the procedures that follow. Store your original software copy in a safe place as a backup.

The following back-up procedures cover the more common computer configurations: a single floppy-disk drive (with hard disk), dual-floppy disk drives, and a hard disk.

Single Floppy-Disk Machines

To copy to another diskette in a single floppy-disk machine (with hard disk), proceed as follows:

1. Turn on power to your computer and display.
2. After system boot-up, the DOS prompt should be C >
3. Be sure the DOS file *DISKCOPY.EXE* is in the root (C:\) directory. Then, type **DISKCOPY A: A:**
4. Insert the *source* diskette (your Distribution Software diskette) into Drive A. The system will prompt you through the disk copying process. When the source diskette has been copied into memory, the System will ask you to insert the *target* diskette into Drive A. The *target* diskette is a blank disk that is to be your back-up disk.
5. When completed, the computer will ask COPY ANOTHER (Y/N)?. Respond by typing **N** .
6. Put the original diskette in a safe place. Label the back-up disk *Working Disk* . Use this disk to run your programs.

Dual Floppy-Disk Machines

If your dual floppy-drive PC has a hard disk, follow the procedure in the preceding subsection, substituting the command **DISKCOPY A: B:** from Step 3. Then insert the Distribution Software diskette in Drive A and the Target diskette in Drive B.

If your machine has no hard disk, use the following procedure.

1. Turn on power to your computer and display, and place your DOS diskette in Drive A.
2. The DOS prompt should be A > If not, type **A:** followed by < **Enter** > . Be sure the diskette in Drive A contains the DISKCOPY.EXE file.
3. Then, type **DISKCOPY A: B:**
4. Insert the *source* diskette (your Distribution Software diskette) into Drive A. The system will prompt you through the disk copying process. It will ask you to insert the *target* diskette into Drive B. The *target* diskette is a blank disk that is to be your back-up disk.
5. When completed, the computer will ask COPY ANOTHER (Y/N)?. Respond by typing **N** .
6. When copying is complete, put the original Distribution Software diskette in a safe place. Label the back-up disk *Working Disk* . Use this disk to run the software.

Hard Disk Machines

1. Start your computer. You should see a prompt, which indicates you are at the DOS level (for example, if your hard drive is designated as C , you should see the prompt C >).
2. The following instructions create a special directory for the Distribution Software files. At the DOS prompt, type: **mkdir DAS20** followed by < **Enter** > . Change to the DAS20 directory by typing: **CD \ DAS20** followed by < **Enter** > .
3. Place the Distribution Software diskette into Floppy Drive A and type **A:** . When the prompt changes from C > to A > , type **copy *.* C:** followed by < **Enter** > .

4. You have now copied the contents of the Distribution Software diskette to your hard disk. Store the Distribution Software diskette in a safe place.

2.4 CONFIGURATION OPTIONS

The DAS-20 contains four switch-selectable configuration options, as follows:

- Base Address.
- MUX Configuration.
- Analog Output Gain and Range For DAC 0.
- Analog Output Gain and Range For DAC 1.

Switch locations for these options are shown in Figure 2-1.

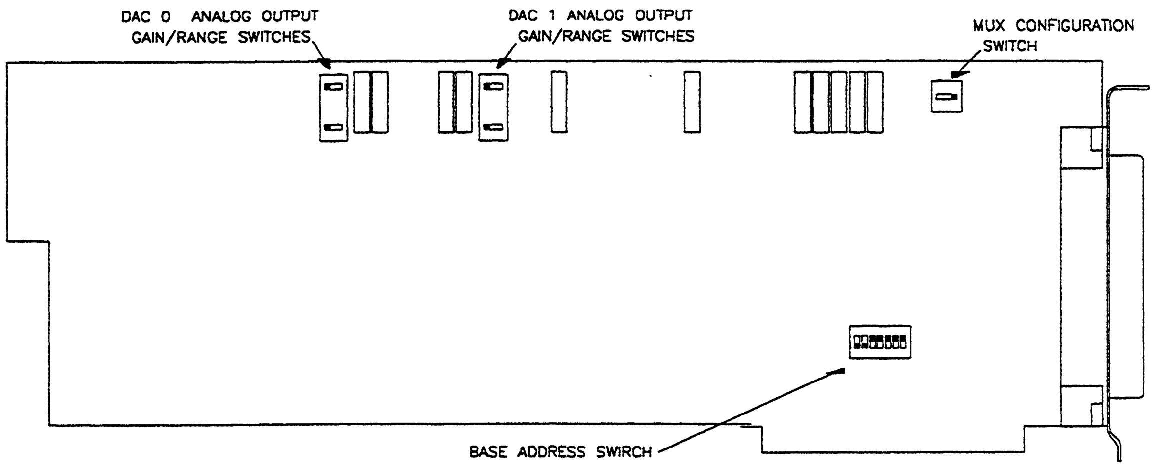

DAC 0 ANALOG OUTPUT GAIN/RANGE SWITCHES
DAC 1 ANALOG OUTPUT GAIN/RANGE SWITCHES
MUX CONFIGURATION SWITCH
BASE ADDRESS SWIRCH

Figure 2-1. Switch locations for configuration options.

Setting The Base Address

Before installing the board, check its Base Address setting on the Base Address Switch (see Figure 2-1). This switch is preset at the factory for an address of 300 Hex, as shown in Figure 2-2.

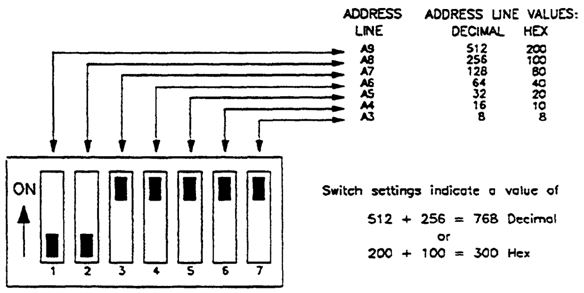


Figure 2-2. Diagram of the Base Address switch.

To communicate with a specific device such as a disk drive, a monitor, another DAS-20 board, etc., the computer must refer its communication to the device's Base Address. A Base Address is typically expressed as a 3-digit Hex number.

A DAS-20 board is preset for a Base Address of 300 Hex, which is within the address range used for a Prototype Card (300 to 31F, as shown in the table below). This default value will function in most computers without conflict, thereby eliminating any need for address selection and configuration. However, if you have a need to change the Base Address from its preset value, you must select an address within a range of 000 to 3FF (0 to 1023 Decimal). In addition, the address must be on an 8-byte boundary, and it must not conflict with addresses the computer is already using for other devices. As an aid to selecting a usable 3-digit Hex number, the following table is an industry-standard I/O address map for the full 000 to 3FF range.

HEX RANGE	USAGE	HEX RANGE	USAGE
000 to 1FF	Internal System	387 to 37F	LPT1:
200 to 20F	Game	380 to 38C	SDLC comm.
210 to 217	Expansion unit	380 to 389	Binary comm. 2
222220 to 24F	Reserved	3A0 to 3A9	Binary comm. 1
278 to 27F	Reserved	3B0 to 3BF	Mono dsp/LPT1:
2F0 to 2F7	LPT2:	3C0 to 3CF	Reserved
2F8 to 2FF	COM2:	3D0 to 3DF	Color graphics
300 to 31F	Prototype card*	3E0 to 3E7	Reserved
320 to 32F	Hard disk	3F0 to 3F7	Floppy disk
		3F8 to 3FF	COM1:

* The default Base Address of 300 Hex.

After you select a Base Address, run the DIPSW.EXE program (contained in your DAS-20 software) to determine the proper setting for the driver board's Base Address switch. To run this program, log to the directory that contains DIPSW.EXE and type DIPSW <Enter> The program will respond with the question DESIRED BASE ADDRESS ---> ?

Your response should be to type a decimal number (or a Hex number preceded by &H, such as &H300) representing the selected address. The computer will display the corresponding Base Address switch settings in diagram form, as shown in the Base Address switch diagram. If the entry is unacceptable, the computer will display an explanatory statement and a request for another entry. As the Base Address switch diagram indicates, the Base Address is set in binary code; also, the switches have value only in the OFF position.

Selecting The MUX Configuration

Decide whether you want eight differential or sixteen single-ended analog input channels and set the MUX Configuration Switch (see Figure 2-1) accordingly. To select the 8-channel position, push the switch left (<--); to select the 16-channel position, push the switch right (-->). Refer to Section 7.1 for details on differential vs. single-ended operation.

Selecting The Analog Output Range For DAC 0 & DAC 1

A dual switch on each of the DAC channels (DAC 0 and DAC 1) allows you to select the output range. These switches are located as shown in Figure 2-1 and set as shown in the following table.

RANGE	DAC 0	DAC 1
±10V	<--	<--
	<--	<--
±5V	<--	<--
	-->	-->
0-10V	-->	-->
	-->	-->
Illegal	-->	-->
	<--	<--

2.5 HARDWARE INSTALLATION

To install the Board in a PC, proceed as follows.

WARNING: ANY ATTEMPT TO INSERT OR REMOVE ANY ADAPTER BOARD WITH THE COMPUTER POWER ON COULD DAMAGE YOUR COMPUTER!

1. Turn Off power to the PC and all attached equipment.
2. Remove the cover of the PC as follows: First remove the cover-mounting screws from the rear panel of the computer. Then, slide the cover of the computer about 3/4 of the way forward. Finally, tilt the cover upwards and remove.
3. Choose an available option slot. Loosen and remove the screw at the top of the blank adapter plate. Then slide the plate up and out to remove.
4. Hold the DAS-20 board in one hand placing your other hand on any metallic part of the PC/AT chassis (but not on any components). This will safely discharge any static electricity from your body.
5. Make sure the board switches have been properly set (refer to the preceding section).
6. Align the board connector with the desired accessory slot and with the corresponding rear-panel slot. Gently press the board downward into the socket. Secure the board in place by inserting the rear-panel adapter-plate screw.
7. Replace the computer's cover. Tilt the cover up and slide it onto the system's base, making sure the front of the cover is under the rail along the front of the frame. Replace the mounting screws.
8. Plug in all cords and cables. Turn the power to the computer back on.

You are now ready to make any necessary system connections, install the DAS-20 software, and perform calibration and perform checks on calibration and adjustment, as described in Section 8.3 of Chapter 8.

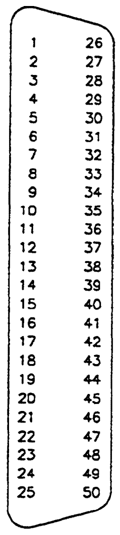
MetraByte recommends that you retain the static-shield packaging for possible future removal and handling of the DAS-20 board.

NOTE: If you intend to use the DMA capabilities of the DAS-20, it is important to know what other devices in your computer use DMA. This information will allow potentially troublesome board conflicts to be avoided. In particular, most Hard-Disk drives use DMA Level 3. The LEV3.EXE program on the DAS-20 disk will determine if your computer's hard disk is using DMA Level 3.

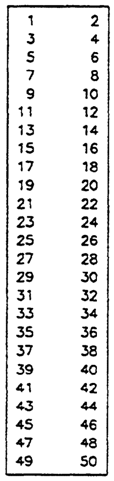
2.6 DAS-20/STA-20 I/O CONNECTIONS

SIGNAL NAME	DAS~~40~~20 PIN NO.	STA-~~40~~20 PIN NO.
CH 0, HI	1	1
CH 1, HI	2	3
CH 2, HI	3	5
CH 3, HI	4	7
CH 4, HI	5	9
CH 5, HI	6	11
CH 6, HI	7	13
CH 7, HI	8	15
Analog GND	9	17
DAC 0 OUT	10	19
Analog GND	11	21
Digital GND	12	23
Sample/Hold	13	25
CTR 1 Gate	14	27
CTR 2 SRC	15	29
CTR 2 OUT	16	31
Digital GND	17	33
DOUT Bit 0	18	35
DOUT Bit 2	19	37
DOUT Bit 4	20	39
DOUT Bit 6	21	41
DIN Bit 0	22	43
DIN Bit 2	23	45
DIN Bit 4	24	47
DIN Bit 6	25	49
CH 8 - HI/CH 0 - LO	26	2
CH 9 - HI/CH 1 - LO	27	4
CH 10 - HI/CH 2 - LO	28	6
CH 11 - HI/CH 3 - LO	29	8
CH 12 - HI/CH 4 - LO	30	10
CH 13 - HI/CH 5 - LO	31	12
CH 14 - HI/CH 6 - LO	32	14
CH 15 - HI/CH 7 - LO	33	16
Analog GND	34	18
DAC 1 OUT	35	20
Analog GND	36	22
Digital GND	37	24
CTR 1 SRC	38	26
CTR 1 OUT	39	28
CTR 2 Gate	40	30
5V Output	41	32
Digital GND	42	34
DOUT Bit 1	43	36
DOUT Bit 3	44	38
DOUT Bit 5	45	40
DOUT Bit 7	46	42
DIN Bit 1	47	44
DIN Bit 3	48	46
DIN Bit 5	49	48
DIN Bit 7	50	50

DAS-20 Board Connector



STA-20 Board Connector



CHAPTER 3

REGISTER MAPS & STRUCTURE

3.1 I/O REGISTER MAPPING

DAS-20 requires 16 consecutive addresses in the computer's I/O address space. The first of these is the *Base +0 Address* and is established for the board by the board's Base Address Switch (refer to the discussion of *Base Address* in Chapter 2). The I/O map for the 16 addresses is as follows:

LOCATION	FUNCTION	TYPE
Base + 0	ADC Low byte	R
	ADC Manual Start	W
Base + 1	ADC High Byte	R
Base + 2	Scanner Queue View	R
	Scanner Queue Load	W
Base + 3	Control Register	R
	Control Register	W
Base + 4	Interrupt Status Register	R
	Interrupt Control Register	W
Base + 5	DMA Control Register	R
	DMA control Register	W
Base + 6	Counter/Timer Data	R
	Counter/Timer Data	W
Base + 7	Counter/Timer Control	R
	Counter/Timer Control	W
Base + 8	DAC #0 Low Byte	W
Base + 9	DAC #0 High Byte	W
Base + 10	DAC #1 Low Byte	W
Base + 11	Digital Byte Input	R
	DAC #1 High Byte	W
Base + 12	Digital Byte Output	R
	Digital Byte Output	W
Base + 13	Not Used	
Base + 14	Not Used	
Base + 15	Not Used	

Where:	R signifies a Read operation.
	W signifies a Write operation.
	ADC signifies A/D Converter.
	DAC signifies D/A Converter.
	DMA signifies Direct Memory Access.

3.2 ADC REGISTERS (BASE ADDRESS +0 AND +1)

These registers are read in the standard Intel Lo/Hi Byte sequence, in which the acquired ADC (Analog/Digital Converter) data, along with an input channel tag, is carried in two 8-bit bytes. Where the DAS-20 contains a 12-bit converter, converter data uses 12 bits of the Lo/Hi Byte sequence while the channel tagging uses the four remaining bits (to show which channel the data comes from, as shown in the register representations, below).

Converter data is left-justified with the channel-tagging data in the least significant nybble (four bits). Writing anything to the Lo Byte (Base Address +0) while the DAS-20 is set to programmed I/O mode starts a conversion. If the Data is acquired with the Instrument Amplifier set to a unipolar range, output data will be in standard Binary format. If the Amplifier is set to a bipolar Range, output data will be in the Two's Complement notation of Binary format.

ADC Register Lo Byte Structure: (Read Base +0)

D7	D6	D5	D4	D3	D2	D1	D0
Bit3	Bit2	Bit1	Bit0	Mux3	Mux2	Mux1	Mux0

ADC Register Hi Byte Structure: (Read Base +1)

D7	D6	D5	D4	D3	D2	D1	D0
Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4

3.3 ADC INSTRUCTION QUEUE (BASE ADDRESS +2)

This port accesses the ADC Instruction Queue RAM. The DAS-20's Instruction Queue is a 2048 by 8 bit memory that sequentially sets the Input Multiplexer to the desired channel and sets the Instrumentation Amplifier to the desired Range & Gain. The last item in the Queue is marked by setting the LSB of that Instruction word high.

To load or read the Queue, Bit D7 in the ADC Control Register (Base Address + 3) must be set low while D6 must be set high. The Queue pointer (which determines which of the 2048 available Channel/Gain locations is to be read/written) is reset to zero each time a READ or WRITE is performed at Base Address + 3. Then (assuming the Queue is enabled), each time a Read is performed on the Queue Read/Write address, the pointer is incremented by one. A more detailed description of loading and reading the Queue is included in the discussion of the ADC Control Register (Section 3.1.3).

Scanner Queue Register Structure: (Read or Write Base +2)

D7	D6	D5	D4	D3	D2	D1	D0
Mux3	Mux2	Mux1	Mux0	Gain1	Gain0	Range	EndofQ

Multiplexer Mapping (Bits 4, 5, 6, and 7 of Queue Register)

SINGLE-ENDED	DIFFERENTIAL	D7	D6	D5	D4
0	0	0	0	0	0
1	1	0	0	0	1
2	2	0	0	1	0
3	3	0	0	1	1
4	4	0	1	0	0
5	5	0	1	0	1
6	6	0	1	1	0
7	7	0	1	1	1
8	-	1	0	0	0
9	-	1	0	0	1
10	-	1	0	1	0
11	-	1	0	1	1
12	-	1	1	0	0
13	-	1	1	0	1
14	-	1	1	1	0
15	-	1	1	1	1

Instrument Amplifier Gains and Ranges

INPUT RANGE	GAIN	UNI/BIPOLAR	D3	D2	D1
0 to +10V	X1	Unipolar	0	0	0
±10V	X.5	Bipolar	0	0	1
0 to +10V	X1	Unipolar	0	1	0
±5V	X1	Bipolar	0	1	1
0 to +1V	X10	Unipolar	1	0	0
±0.5V	X10	Bipolar	1	0	1
0 to 100mV	X100	Unipolar	1	1	0
±50mV	X100	Bipolar	1	1	1

3.4 ADC CONTROL REGISTER (BASE ADDRESS +3)

The ADC Control Register controls the Instruction Queue operation and access, the ADC triggering and gating, the DMA active/inactive, and the Timer's need to insert a delay. The delay allows the Instrumentation Amplifier time to settle in Block Scan modes. Reading the ADC Control Register resets the Instruction Queue the beginning location.

ADC Control Register Structure: (Read or Write Base +3)

D7	D6	D5	D4	D3	D2	D1	D0
ADC Qenab	ADC Wrap	DMA Active	Timer Delay	Trigger Select	Trigger Source	Gate Src	Gate Pol

ADC Control Register Bit Descriptions

D0 **Gate Polarity:** if the Gate Source has been selected as external (D1 = 1), then D0 selects the active level of the Gate. The Gate will be active low if D0 = 0 or active high if D0 = 1.

D1 **Gate Source:** controls whether the ADC is internally gated by ADC End of Conversion (EOC) status (D1 = 0), or Externally controlled D1 = 1.

D2 **Trigger Source:** if the Trigger Select has been set as hardware (D3 = 1), then D2 selects the source of the trigger. The hardware trigger will be Counter 5's output if D2 = 0 or the EXT TRIG pin if D2 = 1.

D3 **Trigger Select:** sets the ADC Trigger mode to Software (writing to Base + 0) when D3 = 0 or Hardware when D3 = 1.

D4 **Timer Delay:** sets the System Timing Controller to Delay Start of Conversion when D4 = 1 or Begin the Conversion immediately when D4 = 0. When using the A/D converter in a "burst" mode, this delay assures that the instrumentation amplifier has had time to settle before the A/D conversion is initiated.

D5 **DMA Active:** controls the DMA Request Signal. The DMA Control register's DMA Enable connects the DAS-20's DMA circuit to the Computer's DMA channel. The DMA is activated when D5 = 1, and deactivated when D5 = 0.

D6 **Queue Control:** works with D7 to select the operation of the DAS-20's Channel/Gain Queuing RAM. The table below shows how these bits are decoded.

D7 **Queue Control:** works with D6 to select the operation of the DAS-20's Channel/Gain Queuing RAM. The table below shows how these bits are decoded.

INSTRUCTION QUEUE MODE	D7	D6
Queue Stationary	0	0
Queue Access to be Loaded or Examined	0	1
Queue Sequences and Stops at End of Queue Flag	1	0
Queue Sequences and Wraps around at End of Queue Flag	1	1

Note that each reading of the Control Register causes the Queue Location Pointer to reset. Each subsequent read or write to the Queue (Base Address +2) increments the Pointer by one. A short example of loading the Queue is shown below.

```
V% = INP(BASEADR + #)       'Clear the Queue pointer
OUT (BASEADR + 3),64        'Enable Queue access
OUT (BASEADR + 2),0         'set scan 1 to: gain of 1, unipolar, channel # 0
OUT (BASEADR + 2),30        'set scan 2 to:gainof 100 Bipolar, Channel # 1
OUT (BASEADR + 2),46        'set scan 3 to: gain of 100 Bipolar, channel # 2
OUT (BASEADR + 2),1         'Set scan 4 to: gain of 1, unipolar, channel #
                                  '0, and end scan routine
```

3.5 INTERRUPT MODE CONTROL (BASE ADDRESS +4)

The Interrupt Control Register enables and disables DAS-20 interrupts, and it determine the interrupt level to be used. Reading the Interrupt control register indicates the DAS-20 interrupt and input multiplexer statuses.

Interrupt Mode Control Register Structure: (Read/Write Base +4)

D7	D6	D5	D4	D3	D2	D1	D0
Int Enab	Int Level	Int Level	Int Level	Int Source	Int Source	Int Status	Mux Status

Interrupt Mode Control Register Bit Descriptions

D0 **Mux Status:** of the Interrupt Control Register indicates the configuration of the Input Multiplexer. When the input is high, configuration is set for 16 Single-ended inputs if D0 = 0 or 8 differential inputs if D0 = 1.

D1 **Interrupt Status:** of the Interrupt Control Register holds the latched interrupt status. It is 0 when an Interrupt is cleared and 1 when an Interrupt is pending. The latch clears by writing to the Interrupt Control Register.

D2 **Interrupt Source:** works with D3 to determine what event will be used to generate the interrupt, as shown in the table below:

D3 **Interrupt Source:** See D2.

INTERRUPT SOURCE	D3	D2
ADC End of Conversion	0	0
Timer#2 Output	0	1
End of Instruction Queue	1	0
DMA Terminal Count	1	1

D4 **Interrupt Level:** works with the D5 and D6 bits to determine which of the IBM interrupt levels is to be used. The table below details the settings.

D5 **Interrupt Level:** See D4.

D6 **Interrupt Level:** See D4.

INTERRUPT LEVEL	D6	D5	D4
N.A.	0	0	0
N.A.	0	0	1
IRQ2	0	1	0
IRQ3	0	1	1
IRQ4	1	0	0
IRQ5	1	0	1
IRQ6	1	1	0
IRQ7	1	1	1

D7 **Interrupt Enable:** enables interrupts from the DAS-20 when set to D7 = 1, and disables interrupts when D7 has been set to 0.

DAC0 High Byte Register Structure: (Write Base +9)

D7	D6	D5	D4	D3	D2	D1	D0
N/A	N/A	N/A	N/A	Bit11	Bit10	Bit 9	Bit 8

3.10 ANALOG OUTPUT #1 (BASE ADDRESS +10 AND +11)

Data transfered to DAC1 is right-justified. The DAC inputs are double buffered. First the LSB then the MSB are written. When the MSB is written, the both bytes of data are loaded into the D/A. If the DAC is configured for Unipolar output, the data must be in true binary format. If the DAC is configured for Bipolar output, the data must be in two's complement format.

OUTPUT CONFIGURATION	DATA RANGE
Unipolar	0 to 4095
Bipolar	-2048 to +2047

DAC1 Low Byte Register Structure: (Write Base +10)

D7	D6	D5	D4	D3	D2	D1	D0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

DAC1 High Byte Register Structure: (Write Base +11)

D7	D6	D5	D4	D3	D2	D1	D0
N/A	N/A	N/A	N/A	Bit11	Bit10	Bit 9	Bit 8

NOTE: Base Address +11 is a shared address that controls Analog Output #1, when written to; but it returns the current status of the digital input port, when read.

3.11 DIGITAL INPUT PORT (BASE ADDRESS +11)

The Digital input port consists of an 8-bit input register that latches the input data when the input port is read. Register structure is as follows:

D7	D6	D5	D4	D3	D2	D1	D0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

3.12 DIGITAL OUTPUT PORT (BASE ADDRESS +12)

The digital output port is latched with the data when Base Address +12 is written to. A read of this address will return the current state of the digital output port. Register structure is as follows:

D7	D6	D5	D4	D3	D2	D1	D0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

3.13 OTHER BASE ADDRESSES

Base Address +13, +14, and +15 are not used by the DAS-20.

* * * * *

CHAPTER 4

PROGRAMMING NOTES

4.1 PROGRAMMING METHODS

At its lowest level, DAS-20 programming uses I/O (input and output) instructions. With *BASIC,* these are the INP (X) and OUT X,Y functions. *Assembly* and most other high-level languages have equivalent instructions (IN AL,DX and OUT DX,AL). Using these functions usually involves formatting data and dealing with absolute I/O addresses. Although not demanding, DAS-20 programming can require many lines of code and requires an understanding of the devices, the data format, and the DAS-20 architecture. To simplify program generation, a special I/O driver routine (DAS20.BIN) is included in the DAS-20 Distribution Software; it covers the majority of common operating modes. You may access DAS20.BIN from BASIC by a single line CALL statement.

The various modes of the CALL routine select all the functions of the DAS-20 format and error-check data, and they perform frequently used sequences of instructions. An example is MODE 3, which performs a sequence of operations required to perform an A/D conversion, to check A/D status, to read data, and to increment the multiplexer.

Using the DAS20.BIN driver saves programming time and has other benefits as well. The driver also supports data collection on interrupt, or DMA from an external source, or the DAS-20 timer/counter. Note that BASIC has no interrupt or DMA processing functions, and so called "background" data collection using these methods is available only while using the CALL routine.

Both methods of programming, using INP and OUT functions and the CALL routines, achieve the same result; and they leave you free to choose either, although usually the BASIC programmer will find the CALL routine much simpler to implement. If you need to perform a special function not supported by the DAS-20 driver, you can modify the DAS20.BIN driver to your requirements. The fully commented assembly source is available from MetraByte on a floppy disk (PN SLD20) and is a good starting point for assembly language programmers who wish to modify the standard driver routines.

The DAS-20 driver software is also provided in a compiled Quick BASIC (4) callable library (DAS20B.QLB). Microsoft Pascal, Small, Medium and Compact "C", and FORTRAN linkable software libraries are available in the MetraByte's optional PCF-20 software package.

4.2 BASIC PROGRAMMING TIPS

BASIC commands that are useful with DAS-20 are as follows:

1. **WAIT** *port, n[,m]*

   Data read at the port is EXCLUSIVE-ORed (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.

2. **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 entering **TIMER ON**

   Trapping is disabled by entering **TIMER OFF**

   Note that trapping occurs only 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 an alternative to using the internal DAS-20/20F counter timer.

4.3 THE CALL ROUTINE

Loading The Machine Language CALL Routine DAS20.BIN

Direct I/O using BASIC INP and OUT is possible but can be tedious to implement (though a lot of the required programming could be handled in subroutines). Using the CALL routine (described herein) not only avoids these problems but circumvents the execution time delays of interpreted or compiled BASIC and also permits high-speed DMA (direct memory access - see Appendix B) and interrupt-driven operations that BASIC does not support.

To use CALL routine DAS20.BIN, you must first load it into memory. You must avoid loading it over any part of memory being used by another program such as BASIC (watch out for BASICA.EXE if you are using GW BASIC), print spoolers, or RAM-Disk. If you interfere with another program's use of memory, the CALL routine will not work and your PC is likely to do strange things for a second or two and then hang up (no damage will result, but you will have to turn off power for a few seconds to return to normal). A word of advice: when you write programs with CALLs, **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; this information supplements the limited information in Appendix C of the IBM BASIC manual.

Microsoft BASIC limits the memory segment for BASIC to 64KB. If BASIC is using all of its 64KB, you will see a message with the following type of statement after you power up or after you load BASIC(A): **61807 Bytes free** . The number of **Bytes free** varies with the versions of BASIC(A) and DOS but is usually greater than 60,000 bytes if memory beyond what BASIC can use is available. But when the number for **Bytes free** is less than 60,000, your PC's memory is already fully utilized and BASIC is adjusting by using less than its 64K requirement. If your PC has little available memory to spare, follow the suggestions for Option 1, below; otherwise follow the suggestions for Option 2.

Option 1: Dealing With Inadequate Memory

If the number for **Bytes free** is less than 60,000, you will have to load the CALL routine by forced contraction of the BASIC workspace and by loading the routine at the end of the newly defined workspace. DAS20.BIN will occupy about 4KB, but you are urged to go for extra margin by clearing an 8KB space.

Step 1 is to work out how much memory BASIC is able to use. First, load BASIC(A) from DOS by entering **BASIC(A)**. Note the number of **Bytes free** in BASIC's opening message, then enter **SYSTEM** to return to DOS.

Reload BASIC(A) with the optional /M parameter by entering **BASIC(A) /M:WS** where WS (the workspace parameter) is set for 30000 or 40000. Then note the number of bytes free in the opening message. The objective is to determine the workspace that must be specified to reduce the **Bytes free** number by at least 8192.

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. xxxxx **CLEAR, WS** where xxxxx is the statement number and WS is the workspace parameter determined above.

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

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

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

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

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

The routine can now be loaded as follows:

```
xxxxx DEF SEG = SG
xxxxx BLOAD "DAS20.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%, DIO%, FLAG% prior to the CALL sequence, as follows:

```
xxxxx DEF SEG = SG
xxxxx DAS20 = 0
xxxxx CALL DAS20 (MD%, DIO%(0), FLAG%)
```

Note that in interpreted Microsoft BASIC, DAS20 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 DAS20 as a name, as it makes CALL DAS20 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 or Quick Basic, the significance of DAS20 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. DAS20.OBJ is supplied on the software disk for this purpose (see Sections 4.4.2 & 4.4.3).

Returning to interpreted BASIC, DAS20 is the offset (actually zero) from the current segment as defined by the last DEF SEG statement. Be careful that you do not inadvertently redefine the current segment somewhere in a program before executing 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 memory's BASIC working area whereas we must set aside space for our subroutine from the top. 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, the data space, and the stack--hanging up the computer. Be careful this does not happen inadvertently if you are memory-limited and later load BASIC with DEBUG or some other coresident 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 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 BASIC load into memory from the bottom up; BASICA.EXE loads from top 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.

If these considerations sound complex, look for assistance by running and listing the file EX0.BAS. This file gives examples of loading and using the CALL routine, and it provides ready-made loading and initializing code that you can merge into your own programs.

Option 2: Dealing With Adequate Memory

The second option 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 8K bytes clear at its beginning. For example you might choose &H2800, which is at 160K, then proceed as follows:

```
xxxxx DEF SEG = &H2800      'Sets up load segment
xxxxx BLOAD "DAS20.BIN",0   'Loads at 2800:0000
xxxxx DAS20 = 0
xxxxx DIM DIO%(4)
""
""
""
xxxxx DEF SEG = &H2800
xxxxx CALL DAS20 (MD%, DIO%(0), FLAG%)
xxxxx etc.
```

An example of this approach is also contained in file EX0.BAS. Before you try loading outside the workspace, be sure you really do have an unused 4K of memory at 160K. 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 DAS20.BIN much 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.

Formatting the CALL Statement

If you are new at 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 DAS20.BIN driver must be of the form:

```
xxxxx  CALL DAS20 (MD%, DIO%(0), FLAG%)
```

As explained in the previous section, DAS20 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 zero and DAS20 = 0.

The three variables within parens 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, as follows:

1. The CALL parameters are positional. 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 DAS20 (DIO%(0), MD%, FLAG%)
   ```

   you will mix up the CALL routine, since it will interpret DIO%(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 DAS20 (MD% + 2, DIO%(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 DAS20 (7, 2, FLAG%)
   ```

   This must be programmed as:

   ```
   xxxxx  MD% = 7
   xxxxx  DIO%(0) = 2
   xxxxx  CALL DAS20 (MD%, DIO%(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. Most modes of the DAS20.BIN CALL routine require multiple items of data to be passed in an array. For this reason, DIO%(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 DIO% array as a two or more dimension array, for example:

```
xxx00 DIM DIO%(4,100)
xxx10 DEF SEG = SG
xxx20 FOR I = 0 to 100
xxx30 CALL DAS20 (MD%, DIO%(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.

Examples of Using the CALL Routine

The DAS-20 driver is a BASICA loadable ".BIN" format driver. The Basic interfacet to the driver functions is in the standard MetraByte three argument format:

```
xx100 CALL DAS20 (MD%, DIO%(0), FLAG%)
```

where

MD% is the number of the function to perform.

DIO%() is an array used for passing additional parameters (if needed) and for returning data (if needed).

FLAG% is an error return value.

The following DAS-20 functions are supported by the DAS20.BIN driver:

MODE	DESCRIPTION OF FUNCTION
0	Initialize the DAS-20, (Base Address, Interrupt level, and DMA level).
1	Load the Channel/Gain queuing RAM.
2	View the current Channel/Gain Queuing RAM.
3	Perform a single A/D conversion and load the data into a BASICA variable.
4	Perform an "N" conversion scan and store the data in a BASICA array. Sample rate is set by pacer clock or external trigger and maximum.
5	Perform an "N" conversion scan, and store the data in memory under interrupt control. Maximum conversion rate = 4000 samples/sec.
6	Perform an "N" conversion scan under DMA control. Conversion rate is set by on board pacer clock or by external trigger. Maximum conversion rate = 100,000 KHz.
7	Command a single D/A conversion.
8	Load memory with D/A conversion data for Modes 9 and 10.
9	Perform "N" D/A conversions under interrupt control.
10	Perform "N" D/A conversions, DMA data transfers. Conversion timing from pacer clock or external trigger. Up to 260,000 Conversions per second.
11	Cancel DMA or interrupt driven operations. Return control completely to program software.
12	Return current status of DMA or Interrupt driven data transfers.
13	Transfer data from memory into BASICA arrays. This mode is necessary since interrupt and DMA driven conversions write and read directly from memory locations without regard to BASICA variables.
14	Read the 8 digital input Bits.
15	Write to the 8 digital output bits.
16	Set Analog trigger mode. This mode can cause any other mode to wait until a certain specified input condition is met before proceeding.
17	Initialize the Counter/Timer chip.
18	Set the 9513 counter's master mode register.
19	Set counter "N" mode register.
20	Set Multiple counter control register.
21	Set Counter "N" load register.
22	Read counter "N" hold register.
23	Measure Frequency with counter timer.
24	Set A/D pacer clock.
25	Set D/A or Block Scan pacer clock.
26	Stop A/D & D/A pacer clocks.
27	Perform "N" scans of a block of analog input channels.
28	Sample Data from EXP-20 board.
29	Set Flag for using SSH-4 accessory board.

4.4 USING DAS20.OBJ FOR CALLS IN COMPILED BASIC

The object code file DAS20.OBJ is a product of an IBM Macro Assembler and may be linked to other object modules from compilers etc. When using the linker, the routine's public name is DAS20. If your programming is entirely in BASIC or BASIC COMPILER, you will not need the source listing.

One quick way 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 (Microsoft and Quick Basic are registered trademarks of the Microsoft Corporation, 10700 Northup Way, Box 97200, Bellevue, WA. 98009). When you compile a BASIC program, the significance of the DAS20 in the CALL statement is no longer the same:

```
xxx10  CALL DAS20 (MD%, DIO%(0), FLAG%)
```

DAS20 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 DAS20.BIN routine and all DEF SEG statements that control the location of the routine. These are not required as the linker will locate the DAS20 routine in memory automatically. After compiling your program, run the linking session as follows:

```
A>  LINK yourprog.obj + DAS20.obj
```

DAS20.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, it speeds up to a point of being able to display about 1 point/s and give an effective real-time display.

4.5 PROGRAMMING IN OTHER LANGUAGES

For users not wishing to program with MetraByte's optional dirver software for "C", Pascal, Fortran, and BASIC (available in the PCF-20 package – see MetraByte catalog), the fully commented source code and listings of the CALL routines are a MetraByte option available on floppy disk in IBM PC-DOS 2.10 format as Part No. SLD-20. This material will interest assembly language programmers, who will find the source listing an excellent starting point for adapting or customizing routines to special requirements.

4.6 MULTIPLE DAS-20S IN ONE SYSTEM

What if you wish to operate more than one DAS-20 in a system? To avoid conflicts, each DAS-20 must have a different Base Address. If interrupts are used, the interrupt must be connected to a different interrupt/DMA level, or if on a common level, each board's interrupt/DMA can be enabled only one at a time. Each board must also be assigned its own CALL routine. To do this, start by loading the DAS20.BIN routine at different locations in memory:

```
xxxxx DEF SEG = SG1
xxxxx BLOAD "DAS20.BIN",0
xxxxx SG2 = SG1 + 8192/16          'allow 8K for each routine
xxxxx DEF SEG = SG2
xxxxx BLOAD "DAS20.BIN",0
xxxxx SG3 = SG2 + 8192/16            '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 DAS201 (MD%, X%, FLAG%)
yyy30 DEF SEG = SG2
yyy40 CALL DAS202 (MD%, X%, FLAG%)    'etc.]
```

Note that this method is for interpreted BASIC only. For compiled BASIC, DAS201 and DAS202 will have to be modified in Assembly language with the SLD-20 source disk. Contact MetraByte for assistance in this endeavor.

4.7 QUICKBASIC APPLICATION NOTE

Procedures

This section deals with converting programs from BASICA to QuickBasic 4.5. The programs of interest are those using MetraByte drivers. As in converting any interpreted BASIC program to a compiler, this section asks you to delete such statements as DEF SEG, CLEAR, BLOAD, DRIVER = 0. Since the program is going to want to make calls to an external driver, that driver must be prepared for use with QB. This is done by using the DRIVER.OBJ of the driver to create a Quick Library as follows:

1. From DOS type the command line

   LINK /Q DRIVER,,,<path>BQLB40.LIB;

   to create DRIVER.QLB, which will be loaded into the QB environment at the time QB is invoked.

2. Invoke QB from DOS by typing the following command line:

   QB /L DRIVER

   Optionally, an application program such as MYPROG.BAS could also be loaded with:

   QB /L DRIVER MYPROG

3. Next, the driver has to be declared to QB by the statement

   DECLARE DAS20 (MODE%,BYVAL DIO%,FLAG%)

   This is unlike the previous method used in BASICA where the driver was made into a special .BIN file and then loaded into BASICA with the BLOAD statement. This method is used since you cannot determine where to BLOAD the DRIVER.BIN as in BASICA.

Next, you must declare variable arrays. The first and third parameters are passed by reference, while the second parameter is passed by value. This is necessary since this parameter must represent the offset of the command integer array. To pass the actual offset, use the VARPTR function as follows:

```
   CALL DRIVER(MD%, VARPTR(DIO%(0)), FLAG%)
```

The VARPTR function has returned the address of DIO%(0), which you must pass as a value to the driver that will use it as a pointer to the first element of your command integer array DIO%. DIO% is declared as a $STATIC array since it must reside in DGROUP in order to be relative to QB data segment, as required by the driver. Since parameters are passed to and returned by the external driver, you must declare those parameters as inter-module global variables; for example:

```
   COMMON SHARED DIO%(), A%()
```

The COMMON makes these variables GLOBAL between modules, and the SHARED statement at the module level makes them known globally in this module. Now that these variables are named in a COMMON statement, the position of their dimension statement determines whether the array is $STATIC or $DYNAMIC. If the array is dimensioned BEFORE the COMMON statement, it is $STATIC. If dimensioned AFTER the COMMON statement, it is $DYNAMIC. If the array is NOT GIVEN in the COMMON statement, it is $DYNAMIC but not inter-module global.

```
DIM DIO%(4)                      $STATIC
DIM A%(5000)                     $STATIC
COMMON SHARED DIO%(), A%()       '$DYNAMIC
dat%(30000)                      '$DYNAMIC, but not inter-module global
                                        '$STATIC
```

The $ Metacommands are necessary for proper compilation using BC in DOS. For more details of the factors determining array types, see Section 2.5 ($STATIC and $DYNAMIC Arrays) and (COMMON statement) in the QuickBasic Language Reference Manual.

The dat% array is used to pass the segment to MODES that collect data and store directly to an intermediate memory buffer as

```
DIO%(1) = VARSEG(dat%(0))
CALL DAS20(MD%, VARPTR(DIO%(0)), FLAG%)
```

Here, QB is managing the location of the buffer rather than the more hazardous practice of guessing where to put the data as

```
DIO%(1) = 5000H
```

At this point the declaration header should be as follows:

```
DIM DIO%(4)
DIM A%(5000)
COMMON SHARED DIO%(), A%()
DECLARE SUB DRIVER(MODE %, BYVAL dummy%, FLAG%)
'$DYNAMIC
dat%(30000)                      '$STATIC
```

A final note on arrays: if there is a need to erase and redimension an array during program execution, it should be declared as $DYNAMIC. The REDIM statement can now change the size, but not the dimensional structure. The ERRASE statement is not necessary. In general, the differences between BASICA and QB should be studied in Appendix A of the Learning and Using QB manual, but a few specific problems that can be encountered are

1. Watch out for reserved words (Appendix B, Language Reference). Some of our example programs use an array sub%, which will cause an error. A good substitute is mux%.
2. Rounding Rules: remember that a compiler will round integers differently than an interpreter. An 0.5 will be rounded to 1 in BASICA, but to 0 in QB.
3. OVERFLOW errors: QB will use default or declared variable types for intermediate calculations where BASICA will make adjustments.

   X%=A%(I) * 1000 / 10000 may cause an overflow since A%(I) could = 20000, then 20000 * 1000 = 20000000 , which is too large to fit an integer. The problem is not the assignment, but the evaluation of the multiplication. Using 1000! solves this problem.

Making Executable Programs

To compile QB programs from the environment when the DRIVER.QLB has been loaded, it is necessary to have the DRIVER.LIB version of the DRIVER.OBJ. This can be done while in DOS as follows:

```
LIB DRIVER.LIB+DRIVER.OBJ
```

It is a good idea to make this library at the same time that the Quick Library is made. To make an executable from the environment, use the following sequence:

1. First, invoke the environment with

   QB /L DRIVER MYPROG

2. Then, to produce a stand alone DRIVER.EXE that does not require Run Time support:
   Select the RUN option.
   Select the Make Exe option.
   Select the Standalone .EXE option.
   Select the Make and Exit option.

   Or, to produce a DRIVER.EXE that requires the Run Time support program BRUN40.EXE to execute either program type MYPROG from DOS:
   Select the RUN option.
   Select the Make Exe option.
   Select the Make .exe requiring BRUN40.EXE option.
   Select the Make and Exit option.

To compile QB programs outside the environment, use the following command sequence from DOS:

```
BC /o MYPROG.BAS;
LINK MYPROG.OBJ+DRIVER.OBJ;
```

The /o option, in the compiler line, causes references to the BCOM40.LIB library to be placed in the object module, so the library response need not be given in the LINK line. This sequence will produce a standalone executable.

As an alternative:

```
BC  MYPROG.BAS;
LINK MYPROG.OBJ+DRIVER.OBJ;
```

The absence of the /o option in the compiler line, causes references to the BRUN40.LIB library to be placed in the object module, so the library response need not be given in the LINK line. This sequence will produce an executable the requires the presence of BRUN40.EXE in the directory named \BIN or in the current directory at the time the program is executed. Refer to the MicroSoft QuickBasic Manual for a discussion of the advantages of both types of executable programs.

Note that the Compiler and Linker expect to find executables in the \BIN directory and in Libraries of the directory named by the environ ment variable LIB. To run either type of executable program, type MYPROG.

Conversion Example

DAS-20 Distribution Software contains several examples of BASICA programs and their QuickBasic equivalents. BASICA examples are recognized by their EX prefixes: EX20.BAS. QuickBasic examples are prefixed with QB: QBEX20.BAS.

4.8 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 Type -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.

All positive error FLAG%s are returned with the mode number trailed by an error code. For example, a FLAG% returned of 61, would signify a Type "1" error, found while calling a Mode 6. Similarly a 143 error would signify a Type 3 error in a Mode 14 call.

The following table lists the standard error types. In addition to these errors, some modes have other applicable error flags. For special error codes, please refer to the specific mode description.

ERROR TYPE	FAULT
0	No error.
-3	Hardware Error.
-2	Driver Not Initialized before non-Mode 0 call.
-1	Mode number out of range <0 or >29.
1	DIO%(0) out of range.
2	DIO%(1) out of range.
3	DIO%(2) out of range.
4	DIO%(3) out of range.
5	DIO%(4) out of range.
6	DIO%(5) out of range.
7	DIO%(6) out of range.
8	DIO%(7) out of range.
9	DIO%(8) out of range.
0	DIO%(9) out of range or Special Mode dependent error.

Error detection after the CALL routine is easily implemented:

```
xxx10 CALL DASH20 (MD%, DIO%(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 it can be left permanently in the program.

* * * * *

CHAPTER **5**

CALL MODES

GENERAL

The following section provides a detailed description of the functions and uses of CALL MODEs 0 - 29. Arguments marked with -> are values passed to the DAS20.BIN Driver. Those marked with <- are return values. Any arguments that are not specified or are marked with the value X are *Don't Care* arguments.

MODE 0: *Initialize the DAS-20*

MODE 0 sets the DAS-20's Base Address, DMA channel, and Interrupt Level. MODE 0 also resets the ADC and Sample Control Queue, sets the input gain to 1X/Bipolar, selects Input Channel 0, and resets the Timer (see Initialize Timer Function). A MODE 0 initialization call must be performed before any other CALLs are made to the DAS-20 driver. Trying to execute any other call before executing a MODE 0 call will generate a FLAG% =1 (Driver Not Initialized Error). An example of using MODE 0 is shown in example program EX0.BAS which has been included in the DAS-20 Distribution Software.

On entry the following parameters should be assigned:

MD% -> 0	'MODE 0
DIO%(0) -> BASE ADDRESS	'usually &H300
DIO%(1) -> Interrupt Level	'2 through 7
DIO%(2) -> DMA channel	'1 or 3
FLAG% <-	'Error checking flag, the value before the call 'does not matter.

Then execute the Call

```
CALL DAS20 (MD%, DIO%(0), FLAG%)
```

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

On return from the call the data will be as follows:

MD% = 0	(unchanged)
DIO%(0) Through DIO%(2)	(unchanged)

The following error codes apply to MODE 0:

FLAG%	= 0: No Error, OK.
	= 1: Base Address out of range <512 or > 1008.
	= 2: Interrupt Level < 2 or > 7.
	= -2: MODE number not equal to zero.
	= 3: DMA level not 1 or 3.
	= -3: Board not present, I/O address wrong.

Error #1 occurs if you specified an I/O address that is less than 512 (Hex 200) or greater than 1008 (Hex 3F0). I/O addresses below Hex 200 are used internally by devices on the IBM PC system board and would always cause an address conflict and I/O addresses above Hex 3FF are not decoded on the IBM PC.

Error #2 occurs if you specified a non-valid Interrupt Level. The available levels on the PC expansion bus correspond to 2 thru 7. Some 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 as follows:

Level 2: Reserved (but not used) for Color Graphics Adapter in an IBM PC-XT.*
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.

* Level 2 in an IBM PC-AT is used to cascade to a second interrupt controller. In this case, Level 5 is the better choice.

If you do not have a particular device installed, it is safe to assume that that level is available for use by DAS-20. The lower the level number, the higher the interrupt priority. Note that the interrupt will not be enabled unless you enter a MODE which requires interrupts for operation. If you are not going to make use of interrupts any level can be chosen e.g. DIO%(1) = 2.

Error #3 occurs if you specify a DMA level other than 1 or 3. There are 4 DMA levels available on the IBM PC (see Appendix B), the highest priority is internally used for dynamic memory refresh and is not accessible to the user. The other levels (1 thru 3) are available on the expansion bus, but Level 2 is always used by the floppy disk drive(s) and cannot be shared. In hard disk (XT) computers, Level 3 may be used by the hard disk, but depending on the design of the disk controller hardware and fixed disk BIOS, may in some cases be available. To determine whether your computer's hard disk uses Level 3, run LEV3.EXE from DOS. In any case Level 1 is usually available so if Level 3 is used by the hard disk set DIO%(2) = 1. In floppy-disk-only machines DIO%(2) can be 1 or 3. Also, note that DMA requests and the 8237 controller are not enabled until you enter MODEs 6 or 10. If you are not using MODEs 6 or 10, it is irrelevant whether you set DIO%(2) = 1 or 3.

MODE 0 performs several other initializing functions. The Queuing RAM sequencer is cleared. The DAS-20 control and timer counter enable registers are cleared, disabling all interrupt, DMA and external trigger functions. MODE 0 also performs a simple read/write test as a check on the function and presence of the DAS-20 hardware. If you obtain Error -3, it indicates either a hardware fault in the DAS-20 or more commonly a discrepancy between the Base Address specified in DIO%(0) and the actual switch setting on the board.

MODE 1: *Load the ADC Control Queue*

MODE 1 allows the channel/gain queue to be loaded one step at a time. All Analog input MODEs except for MODE 3 get the channel-number and input-range information from the ADC queue. The Queue is a 2048 Byte RAM that is used to control the Analog Input Multiplexor (which selects the input channel sampled) and the input instrumentation amplifier (which selects the input range).

When making a CALL that specifies that more than one ADC conversion (MODEs 4, 5, and 6) the Sampling Queue is used to select the desired input conditions. The first sample taken is always from the first (Zeroeth) location in the Queue. At the end of the ADC conversion, the Queue Pointer is incremented to point at the second Queue location. Each time another sample is taken, the queue Pointer is incremented, and selects the next input Channel/Range in the Queue. The following list is an example of what a typical Queue might look like.

SAMPLE #	CHANNEL #	INPUT RANGE
*0	0	±10V
1	5	±10
2	1	0-1V
3	5	±10
4	2	0-1V
5	5	±10
6	10	0-1V
7	5	±5
8	0	±10
9	5	±5
10	1	0-1V
11	5	±5
12	2	0-1V
13	5	±0.5
14	10	0-1V
15	5	±0.5

* When used with SSH-4, Sample #0 does Sample & Hold. Channel # and Input Range are not used and can be any arbitrary pair. Sample #0 is first in Queue.

Note that this type of system allows extremely complex scans of multiple channels. Also, since the entire list is loaded into the RAM before the actual Data acquisition run, the complexity of the scan does not affect overall sample rate. A few interesting things this feature allows are depicted in the above list.

First Channel 5 is being scanned at 4 times the sample rate as the other monitored channels. Secondly, as the run proceeds, the input gain on channel 5 is increased (as might be desired when studying the decay of a step function in a linear system.

The Queue is loaded using the standard DAS-20 Call in the following format:

```
xx100 CALL DAS20 (MD%, DIO%(0), FLAG%)
```

where:

MD% -> 1

DIO%(0) -> Channel number (0 to 7 differential or 0 to 15 single ended.

DIO%(1) -> Gain/input range

DIO%(1) Instrument Amplifier Gains and Ranges

INPUT RANGE	GAIN	UNI/BIPOLAR	DIO%(1)
0 to +10V	X1	Unipolar	0
±10V	X.5	Bipolar	1
0 to +10V	X1	Unipolar	2
±5V	X1	Bipolar	3
0 to +1V	X10	Unipolar	4
±0.5V	X10	Bipolar	5
0 to 100mV	X100	Unipolar	6
±50mV	X100	Bipolar	7

DIO%(2) -> Command #

Where Command # is 0, 1 or 2. The available commands are described below:

0 = normal queue entry.

1 = last entry, add EOQ flag.

2 = first entry, initialize the counters.

FLAG% <- Error codes

Where:

FLAG%	= 0: No Error, OK.
	= -1: MODE number out of range, <0 or >29.
	= -2: Driver not installed.
	= 11: Illegal channel.
	= 12: Illegal gain.
	= 13: Illegal Queue command.

The first entry (when DIO%(2) is 2) automatically re-initializes the DAS-20 counters and sets the Sample/Hold signal to a low state. Subsequent (DIO%(2) = 0) entries simply load the queuing RAM; the final entry (DIO%(2) = 1) automatically inserts an EOQ (End Of Queue) bit. An example of loading and reading the sampling Queue is on the DAS-20 software disk. Please refer to EX4.BAS for the example.

The Scan list above would be set with the following calls:

```
MD% = 1                              'select MODE 1
DIO%(0) = 0                          'select Ch # 0
DIO%(1) = 2                          '+ 10 V input
DIO%(2) = 2                          'Set first entry in queue
CALL DAS20 (MD%, DIO%(0), FLAG%)     'execute call

DIO%(0) = 5                          'select Ch # 5
DIO%(1) = 2                          '+ 10 V input
DIO%(2) = 0                          'normal Queue entry
CALL DAS20 (MD%, DIO%(0), FLAG%)     'execute call
```

```
DIO%(0) = 1                     'select Ch # 1
DIO%(1) = 4                     '0-1 V input
DIO%(2) = 0                     'normal Queue entry
CALL DAS20 (MD%, DIO%(0), FLAG%) 'execute call

DIO%(0) = 0                     'select Ch # 5
DIO%(1) = 2                     '+ 10 V input
DIO%(2) = 0                     'normal Queue entry

CALL DAS20 (MD%, DIO%(0), FLAG%) 'execute call
.
.
.
DIO%(0) = 0                     'select Ch # 5
DIO%(1) = 5                     'Å0.5 V input
DIO%(2) = 1                     'Signify last entry in Queue
.
.
.
CALL DAS20 (MD%, DIO%(0), FLAG%) 'execute call
```

MODE 2: *View the Current Queue*

MODE 2 allows the current RAM Queue to be read. When DIO%(2) is 0, the Queue pointer is automatically incremented after the read so that the next read will be of the next Queue location. To reset the Queue pointer to 0, issue the same call with DIO%(2) = 2.

Arguments:

MD% -> 2

DIO%(0) <- channel number

DIO%(1) <- Gain

DIO%(2) -> Command #

Where "Command #" operates as follows:

0 = get next queue command

2 =reset and return to first command in the queue

DIO%(2) <- EOQ condition

On Return, DIO%(2) will be:

0 – End of Queue bit not set.

1 – End of Queue bit set.

FLAG% <- Errors

0 = : No Error.
-1 = : MODE # out of Range.
20 = : Command # out of range.

MODE 3: *Perform a Single ADC Conversion Without Using Queuing RAM*

Initialize the ADC converter, set the channel and input range desired, wait for completion, and return data. Note that this is the only MODE that does not perform conversions based on the Queuing RAM. An example of using MODE 3 is provided in the EX3.BAS program included on the DAS-20 disk.

Arguments:

MD% -> 3

DIO%(0): = Gain Code upon entry to call; = Data upon exit from call.

DIO%(0) Instrument Amplifier Gains and Ranges

INPUT RANGE	GAIN	UNI/BIPOLAR	DIO%(0)
0 to +10V	X1	Unipolar	0
±10V	X.5	Bipolar	1
0 to +10V	X1	Unipolar	2
±5V	X1	Bipolar	3
0 to +1V	X10	Unipolar	4
±0.5V	X10	Bipolar	5
0 to +100mV	X100	Unipolar	6
±50mV	X100	Bipolar	7

DIO%(1): = Channel # upon entry to call; = Channel # upon exit from call.

DIO%(1) Channel select data byte

CHANNEL NUMBER		
SINGLE ENDED	DIFFERENTIAL	DIO%(1)
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	-	8
9	-	9
10	-	10
11	-	11
12	-	12
13	-	13
14	-	14
15	-	15

FLAG% <- Errors

Where

Flag%	= 0: No Errors.
	= -3: Hardware Error.
	= -1: MODE # out of range.
	= 32: Gain/input range out of range.
	= 31: Channel # out of range (note that if a channel # between 8 and 15 is entered, and the board is set to 8 channel Diff operation this error will result).

An example selecting Channel 2, at an input range of ±5 Volts is shown below:

```
MD% = 3                            'Set MODE 3
DIO%(0) = 3                        '±5 Volt input
DIO%(1) = 2                        'CH # 2
CALL DAS20 (MD%, DIO%(0), FLAG%)   'Execute call
VOLTS = DIO%(0) * 10/4096          'Convert the input to volts
PRINT VOLTS
```

Note that to convert the DIO%(0) into volts, we need only multiply DIO%(0) by the full scale input range divided by 4096. For example:

±10V is 20V full-scale. Multiply DIO%(0) by 20/4096 or 10/2048 to convert bits to volts.

0-1V is 1V full-scale. Divide DIO%(0) by 4096 to convert the bits into volts.

MODE 4: *Perform N Conversions (Program Control)*

MODE 4 performs N ADC conversions and transfers data directly into a BASIC integer array. Since the CPU is performing the ADC polling and data transfers as a "foreground" operation, exit from the CALL will not occur until all conversions have been completed. If you do not want to wait for data to be collected, both MODEs 5 or 6 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 ADC will perform conversions on channels in accordance with the scan Queue conditions set in MODE 1. When the number of conversions "N" is larger than the number of items held in the Scan Queue, the Queue resets after sampling it's final entry and begins sampling from Queue address 0 again. If MODE 1 has not been entered prior to MODE 4, a Flag% error may be returned; more likely, however, a false set of samples from arbitrary channels will be returned.

The ADC may be triggered from 2 sources, the programmable interval timer or an external trigger pulse according to DIO%(2). If the programmable interval timer is used, then EXT TRIG acts as a start gate to the operation. If an external trigger is used, trigger pulses are applied to EXT TRIG and positive edges start conversions.

On entry the following parameters should be initialized:-

MD% -> 4 (MODE number)

DIO%(0) -> Number of conversions required (Word count). Range 1 to N where N-1 <= array dimension.

DIO%(1) -> VARPTR(ARRAY%(M)) - array pointer. Conversions may be loaded starting at the Mth position in an array or at the start if M = 0.

NOTE: If the DIO% array and the DataArray (ARRAY%(M)) reside in different data GROUPS, specify the DIO% parameters as follows:

DIO%(1) = -1 ;

DIO%(4) = segadr(DataArray) ; (* Segment Of The Data Array *)

DIO%(5) = offadr(DataArray) ; (* Offset Of The Data Array *)

DIO%(2) - Trigger source. There are 3 possible, as follows:

DIO%(2) = 0 : External clock. Pin 26 (COUNTER 1 SOURCE) on the 3M connector is the source for the external clock. One ADC conversion occurs immediately after each rising edge of the signal applied to this pin (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(2) = 1 : Internal Clock/External Gate. Pin 27 (COUNTER 1 GATE) on the 3M connector is the source for an external gating signal; the conversion clock is derived from the on-board timers. An ADC conversion occurs whenever the timers issue a rising edge and Pin 27 is at a logical high (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(2) = 2 : Internal Clock/No Gate. An ADC conversion occurs whenever the timers issue a rising edge (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(3) : Data from UNIpolar or BIpolar inputs.

DIO%(3) = 0 : Unipolar data

DIO%(3) = 1 : Bipolar data

Since Unipolar (0-4095) and Bipolar (-2048 to +2047) data use different formats, it is necessary to tell the transfer routine which type is being used. In channel scans that include both Unipolar and Bipolar, either format can be selected then scaled to the correct form. Scaling conventions are as follows:

Subtract 4096 from Bipolar data transferred in Unipolar MODE when Value > 2047.

Add 4096 to Unipolar data transferred in Bipolar MODE when value < 0.

FLAG% = X (value does not matter)

Then:

```
CALL DAS20 (MD%, DIO%(0), FLAG%)
```

On return the variables contain data as follows:

MD% = 4 (unchanged)

DIO%(0 thru 4) (unchanged)

ARRAY%(M) = first data word

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

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

Etc.

The following error codes apply to MODE 4:

FLAG%	= 0: No Error, OK.
	= -2: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.
	= 41: Number of conversions 0 or negative.
	= 43: Ttrigger MODE not 0, 1, or 2.

An example of the use of MODE 4 may be found in EX4.BAS.

Certain precautions are prerequisite to the reliable use of MODE 4. First, you must dimension a receiving array that has at least as many elements as the number of conversions specified in DIO%(0). No checks are made by the CALL routine on whether you are performing more conversions than the array will hold. If you are, some of the data area of BASIC will be overwritten and may destroy descriptors and other variable data and cause strange effects. **Do not overun 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 OK:

```
xxx10 DIM DIO%(4)                    'declare DIO%(*)
xxx20 MD% = 4                        'MODE #
xxx30 DIO%(0) = 1000                 'number of conversions
xxx40 DIO%(1) = VARPTR(X%(0))        'pointer to array
xxx50 DIO%(2) = 1                    'trigger MODE
xxx60 CALL DAS20 (MD%, DIO%(0), FLAG%)
```

But this will cause a problem:

```
xxx10 DIM DIO%(4)                    'declare DIO%(*)
xxx20 MD% = 4                        'MODE #
xxx30 DIO%(0) = 1000                 'number of conversions
xxx40 DIO%(1) = VARPTR(X%(0))        'pointer to array
xxx50 DIO%(2) = 1                    'trigger MODE
xxx60 NEW_VARIABLE = X               'new variable
xxx60 CALL DAS20 (MD%, DIO%(0), FLAG%)
```

The problem arises because BASIC stores array variables in memory above the data area (located above the program storage area) for simple (non-array) variables. If you introduce a simple variable that has not been used before, BASIC makes room for this variable by relocating all the array variables upwards in memory. If we assign the pointer (using VARPTR) to the receiving array before a new variable is introduced and we then enter the CALL, 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 4 **cannot** occur until EXT TRIG has been taken high if using the programmable timer and a sufficient number of trigger pulses to perform the desired number of conversions has been supplied. If these conditions are not met, your computer may give the erroneous appearance of being hung up. To abandon further conversions, hit Ctrl + Alt + Del and reboot the computer.

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 ADC for several hundred microseconds, data may be lost above 3000 samples/sec.

MODE 5: *Multiple A/D Conversions -- Interrupt Driven*

MODE 5 performs N A/D conversions triggered either externally or by the programmable timer using MODE 24. At the end of each conversion, an interrupt is generated that invokes an interrupt handler routine installed by this mode. This routine transfers the data from each conversion to a specified segment of memory and keeps track of the total number of conversions performed. When the number reaches N, as specified by DIO%(0), interrupts are disabled if in the nonrecycle MODE (DIO%(3)=1), or the process is repeated continuously to the same segment of memory if DIO%(3)=0.

Note that once MODE 5 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 DAS20.BIN driver or the A/D data area). For this reason it is termed a background operation and in most respects is functionally similar to MODE 6 (DMA) although much slower. About 3000 samples/sec (machine dependent) are possible in MODE 5.

To return data to a BASIC array when MODE 5 is operating or has finished operation, use MODE 13. To assess the progress of an operation initiated by MODE 5, use MODE 12. To abort an interrupt operation initiated by MODE 5, use MODE 11. For detailed 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 Queue that must be set by MODE 1. When the number of conversions "N" is larger than the number of items held in the Scan Queue, the Queue resets after sampling it's final entry and begins sampling from Queue address 0 again.

The A/D may be triggered from 2 sources, the programmable interval timer or an external trigger pulse according to DIO%(2). If the programmable interval timer is used, then EXT TRIG acts as a start gate to the operation. If an external trigger is used, trigger pulses are applied to EXT TRIG, positive edges start conversions.

On entry the following parameters should be initialized:

MD% -> 5 (MODE number).

DIO%(0) = Number of conversions required (Word count). Range 1 to N, where N can range from 1 to 32766. When N is larger than the number of items in the sample Queue, the program automatically resets the Queue pointer, and continues sampling.

DIO%(1) = Segment of memory to receive data. Segments are on paragraph (16 bit) boundaries e.g. DIO%(1) = &H2000 would start loading A/D data at 128K. Be sure to choose an empty area of memory for a data buffer (always outside BASIC workspace).

DIO%(2) - Trigger source. There are 3 possible:

DIO%(2) = 0 : External clock. Pin 26 (COUNTER 1 SOURCE) on the 3M connector is the source for the external clock. One A/D conversion occurs immediately after each rising edge of the signal applied to this pin (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(2) = 1 : Internal Clock/External Gate. Pin 27 (COUNTER 1 GATE) on the 3M connector is the source for an external gating signal; the conversion clock is derived from the on-board timers. An A/D conversion occurs whenever the timers issue a rising edge and Pin 27 is at a logical high (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(2) = 2 : Internal Clock/No Gate. An A/D conversion occurs whenever the timers issue a rising edge (so long as the number of conversions has not reached the number specified in DIO%(0)).

Since MODE 5 is a background task, the foreground program may stop the A/D collection at any time by executing a MODE 11 call. If the MODE 5 operation is being internally paced, MODE 26 (stop A/D clock) should also be called.

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

DIO%(3) = 0 : Re-cycle. In this case data is continuously written to the same memory. DIO%(0) corresponds to the memory "buffer" length. The status of the operation is 1 = active until stopped by MODE 11.

DIO%(3) = 1 : One cycle. After completion of the number of conversions specified, interrupts are disabled, setting the operation status to zero.

DIO%(4) = X	(value does not matter)
FLAG% = X	(value does not matter)

An example of executing MODE 5 would be:

```
CALL DAS20 (MD%, DIO%(0), FLAG%)
```

On return the variables contain data as follows:

MD% = 5 (unchanged)

DIO%(0 thru 4) (unchanged)

The following Error Codes apply to MODE 5:

FLAG%	= 0: No error, OK.
	= -2: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.
	= 50: Interrupt or DMA already active.
	= 51: Number of conversions 0 or negative.
	= 53: Trigger input out of range): 0, 1, or 2.
	= 54: Recycle flag not 0 or 1.

EX5.BAS contains usage examples MODEs 0, 1, 5, 11, 12, 13, and 24.

Several details are pre-requisite to the reliable use of MODE 5. First, MODE 5 will not allow reinstallation of the interrupt handler if the interrupt is still active; Error #50 will result. If you request the recycle MODE (which generates continuous interrupts), you must then run MODE 11 (which disables interrupts) before you can succesfully run MODE 5 again. If you are in nonrecycle MODE, then it must have reached the word count (which automatically disables interrupts) before MODE 5 can be run again. On completion of an interrupt operation, the selected level of the 8259 interrupt mask register is disabled and the tristate interrupt driver of the DAS-20 is placed in the high impedance state. This allows more than one hardware device, or multiple DAS-20's to use the same interrupt level.

Second, since the segment registers are not incremented by the handler, the maximum data area available is 64K (a page) for 32,766 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 13 during or after the operation of MODE 5 and MODE 13 will not alter the memory. It is possible to rewrite the interrupt handler to use several segments of memory for data storage.

Third, conversion speeds up to 3000 samples/sec can be reliably achieved with MODE 5. For higher speeds use MODE 6. MODE 5 is subject to disruptions from higher priority interrupts, notably the system timer, and this is the main limitation on throughput. In general, MODE 6 is a better choice than MODE 5 at any speed, since it uses much less processing time; however, MODE 5 is included as the hardware can support this type of operation.

MODE 6: *Multiple A/D Samples with DMA Data Transfer*

MODE 6 performs *N* A/D conversions triggered either externally or by the programmable timer using MODE 24; and at the end of each conversion, a direct memory transfer from the A/D to memory is performed under the control of the IBM PC 8237 DMA controller. This device transfers the data from each conversion to a specified segment of memory and keeps track of the total number of conversions performed without involving the CPU. When the number reaches N, as specified by DIO%(0), DMA transfers cease and a terminal interrupt is generated if in the non-recycle MODE (DIO%(3)=1), or DMA transfers are repeated continuously to the same segment of memory if DIO%(3) specifies the re-cycle (autoinitialize) MODE. Note that once MODE 6 has enabled DMA, conversions continue regardless of what other programs the user may be running. In fact, the DAS20.BIN driver may be erased from memory. DMA is performed purely by the system and DAS-20 hardware, is a background operation and is extremely fast. Throughput is limited by the speed of the A/D converter and settling of the sample hold. Using the Harris HI-774 A/D converter, the DAS-20 can sustain a throughput slightly in excess of 100,000 samples/sec. A technical description of the IBM P.C. DMA arrangements is contained in Appendix E.

To return data to a BASIC array when MODE 6 is operating or has finished operation, use MODE 13. To assess the progress of an operation initiated by MODE 6, use MODE 12. To abort a DMA operation initiated by MODE 6, use MODE 11. 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 Queue set in MODE 1. When the number of conversions "N" is larger than the number of items held in the Scan Queue, the Queue resets after sampling it's final entry and begins sampling from Queue address 0 again. If MODE 1 has not been entered prior to MODE 6, a FLAG% error will be generated and no conversions will be performed.

NOTE: The DAS-20's input instrumentation amplifier has been designed for low noise, high gain, and high speed performance. However, in order to minimize noise at high gains, the input bandwidth has been reduced. To assure accurate readings the following maximum sample rates at higher gains should be observed.

INPUT RANGE RECOMMENDED	MAXIMUM SAMPLE RATE
0 to 10V	100,000 samples/sec
±10 V	100,000 samples/sec
±5 V	100,000 samples/sec
0-1 V	80,000 samples/sec
±0.5 V	80,000 samples/sec
0 to 100mV	40,000 samples/sec
±50 mV	40,000 samples/sec

On entry the following parameters should be initialized:

MD% -> 6 (MODE number).

DIO%(0) -> Number of conversions required (Word count). Range 1 to N, where N can range from 1 to 32766. When N is larger than the number of items in the sample Queue, the program automatically resets the Queue pointer, and continues sampling.

DIO%(1) -> Segment of memory to receive data. Segments are on paragraph (16 bit) boundaries e.g. DIO%(1) = &H5000 would start loading A/D data at 320K. Be sure to choose an empty area of memory for a data buffer (always outside BASIC workspace).

DIO%(2) -> Trigger source. There are 3 possible:

DIO%(2) = 0 : External clock. Pin 26 (COUNTER 1 SOURCE) on the 3M connector is the source for the external clock. One A/D conversion occurs immediately after each rising edge of the signal applied to this pin (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(2) = 1 : Internal Clock/External Gate. Pin 27 (COUNTER 1 GATE) on the 3M connector is the source for an external gating signal; the conversion clock is derived from the on-board timers. An A/D conversion occurs whenever the timers issue a rising edge and Pin 27 is at a logical high (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(2) = 2 : Internal Clock/No Gate. An A/D conversion occurs whenever the timers issue a rising edge (so long as the number of conversions has not reached the number specified in DIO%(0)).

Since MODE 6 is a background task, the foreground program may stop the A/D collection at any time by executing a MODE 11 call. If the MODE 6 operation is being internally paced, MODE 26 (stop A/D clock) should also be called.

DIO%(3) -> Single-cycle/Re-cycle operation:

DIO%(3) = 1 : One cycle. After completion of the number of conversions specified, DMA ceases, an interrupt is generated and DMA is disabled, setting the DMA status to zero.

DIO%(3) = 0 : Re-cycle. In this case data is continuously written to the same memory. DIO%(0) corresponds to the memory "buffer" length. This corresponds to a DMA auto-initialize operation. To end the acquisition, MODE 11 must be run.

DIO%(4) = X (value does not matter)

FLAG% = X (value does not matter)

then

```
        CALL DAS20 (MD%, DIO%(0), FLAG%)
```

On return the variables contain data as follows:

MD% = 6 (unchanged)

DIO%(0 thru 4) (unchanged)

The following Error Codes apply to MODE 6:

FLAG%	= 0: No error, OK.
	= 1: Driver not initialized.
	=-1: MODE number out of range, <0 or >29.
	= 60: Interrupt or DMA already active.
	= 61: Number of conversions 0 or negative.
	= 63: Trigger out of range)(Not 0, 1, or 2.
	= 64: DIO%(3) one-shot or recycle, not 0 or 1.
	= 67: Wrap Around) on DMA page.

Usage examples for MODEs 1, 6, 11, 12, 13, and 24 may be found in EX6.BAS.

Certain details are pre-requisite to the reliable use of MODE 6. First, you cannot rerun MODE 6 if a MODE 5 interrupt or a previous MODE 6 DMA operation is still active. Error #60 will result. If you request the recycle MODE which generates continuous DMA transfers, then MODE 11 (which disables DMA and interrupts) must be run before you can succesfully run MODE 6 again. If you are in nonrecycle MODE, then it must have reached the word count which automatically disables DMA before MODE 6 can be run again. On completion of a DMA operation, the tristate DMA request (DRQ) drivers of the DAS-20 are placed in the high impedance state. This allows more than one hardware device, or multiple DAS-20's to use the same DMA level as long as they do so sequentially.

Second, since the DMA page registers cannot be incremented by the controller, the maximum data area available is 64K (a page) for 32,768 conversions. Be sure that your data area is not in use by your program or altered by subsequent operations. Also be careful that your word count added to your segment will not cross a page boundary. This condition which would otherwise cause a wrap around to the beginning of the page is detected by the driver and produces error #67. To avoid this, a conservative rule is to use a segment that is also a page boundary e.g. &H1000, &H2000, etc. Data may be retrieved using MODE 13 during or after the operation of MODE 6, and MODE 13 will not alter the memory.

Note that once DMA operation is initiated, it may well continue during or following execution of the program. This is a "background" operation. Strange effects can occur if you inadvertently transfer data into your program area. Be sure to use a free area of memory for your DMA buffer. DEBUG can help you locate free areas; they are usually loaded with zeroes on boot-up.

If you must transfer more data than one 64K data segment can hold, MetraByte offers optional data software (STREAMER) to perform continuous transfer through a DMA buffer to hard disk. STREAMER allows you to continuosly stream "gapless" data to your hard disk at more than 50KHz on a PC/AT (somewhat slower on an XT). A standard 20MB hard disk will hold about 3 minutes of data at 50KHz. The maximum speeds attainable depend on your hard-disk controller type and manufacturer; contact MetraByte for further information. Call also for information on STREAMER.

MODE 7: *Command a Single D/A Conversion*

MODE 7 performs a write to one of the D/A converters.

MD% -> 7

DIO%(0) -> D/A channel (0 or 1)

DIO%(1) -> D/A data (0 - 4095 for unipolar; -2048 to 2047 for Bipolar)

FLAG%	= 0: No error, OK.
	= 1: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.
	= -2: Driver not initialized.
	= 70: D/A channel # not 0 or 1.

Note: The user is responsible for ensuring that data is within the valid range for DAC operation; this is not covered by error checking.

For an example of D/A output using MODE 7, see EX7.BAS.

MODE 8: *Load Memory Segment for D/A Conversion*

MODE 8 has been developed to allow the user to load a segment of memory in preparation for writing the data out to the DAS-20 D/A converters. Each conversion requires two bytes of memory to be stored. This allows up to 32,768 conversions (single D/A), or 16,384 conversions (both D/A's) to be loaded into a 64 KB page of memory.

Arguments:

MD% -> 8

DIO%(0) -> Number of Data words (conversions) to transfer.

DIO%(1) -> Memory Segment Address for raw data (the data that will be used by the D/A converters). Note that it is recommended that the buffer start on an even page since the Data buffer must not cross a page boundary. (e.g. &h2000, or &h3000.)

DIO%(2) -> Starting offset for entering data into the raw data buffer. DIO%(2) will usually be 0, and the first data word will be written into the first buffer location. However, in some instances, when data is being taken from more than one array, it is necessary to load one array into memory, then load another array immediately following the first.

DIO%(3) -> Pointer to BASIC array containing the data to write to the D/As. The pointer may be found by executing the following instructions:

Data is assumed to be in the array DACDATA%(1000)

```
xxx50 DIM DACDATA%(1000)
***60   . . .
xxx70   . . .  Load the
xxx80   . . .  data into the
```

```
xxx90   . . .  Array
xx100   . . .
xx110 DIO%(3) = VARPTR (DACDATA%(0))
```

Note that it is advisable to make the assignments:

```
DIO%(3) = VARPTR(DACDATA%(0))
```

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:

```
xxx10  DIO%(3) = VARPTR(DT%(0))
xxx20  I = 3
xxx30  CALL DAS20 (MD%, DIO%(0), FLAG%)
```

NOTE: If the DIO% array and the DataArray (DT%(0)) reside in different data GROUPS, specify the DIO% parameters as follows:

DIO%(3) = -1 ;

DIO%(4) = segadr(DataArray) ; (* Segment of the DataArray *)

DIO%(5) = offadr(DataArray) ; (* Offset Of The DataArray *)

As an alternative to MODE 8, you may be tempted to write data using BASIC's PEEK function.

FLAG% <- Errors

Where

FLAG%	= 0: No error, OK.
	= -1: MODE number out of range,<0or >29.
	= 2: MODE 0 not initialized.
	= 80: Word count zero or >32767.
	= 81: Start offset out of range.

For examples of using MODEs 8, 9 and 10 and 25, refer to EX8.BAS.

MODE 9: *Multiple Interrupt D/A Conversions*

MODE 9 performs "N" D/A conversions based on interrupt control. Data is read directly from memory (loaded by MODE 8) and written to the D/A converters. If MODE 8 was set to load data for both D/As, MODE 9 must also select 2-channel operation. However, if MODE 8 loaded data for only one D/A channel, MODE 9 can select either D/A channel. The update rate for D/A conversions can be set by MODE 25 or an external signal. The following variables are defined by MODE 9:

MD% -> 9

DIO%(0) -> Number of conversions

DIO%(1) -> Segment address of buffer

DIO%(2) -> Trigger source; there are three possible, as follows:

DIO%(2) = 0. Maximum speed, external gate. Conversions occur at the rate of 52 kHz (19 µs per conversion), and Pin 30 (COUNTER 2 GATE) on the 3M connector acts as a gate signal.

DIO%(2) = 1. Internal Clock/External Gate. Pin 27 (COUNTER 1 GATE) of the 3M connector is the source for an external gating signal; the conversion clock is derived from the on-board timers. Upon positive-level transition, DAC(s) outputs will be active. If both channels are selected, the number of conversions is twice the number desired for each channel. For example, selecting 20 conversions per cycle and two channels gives 10 conversions per cycle per channel. Also note that the update rate is doubled; for example, selecting a 100Hz update rate and two channels gives a rate of 200Hz.

DIO%(2) = 2. Internal clock, no gate. An A/D conversion occurs with each rising edge from the timers (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(3) -> Recycle flag.

0 = restart on Nth conversion.

X = terminate on Xth cycle of N conversions.

DIO%(4) -> Channel select.

0 = Channel 0.

1 = Channel 1.

2 = Both channels.

FLAG% <- Error Codes

Where

FLAG%	= 0: No error, OK.
	= 1: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.
	= 90: Interrupt already active.
	= 91: Word count zero or >32767.
	= 92: Trigger source not 0, 1, or 2.
	= 93: Recycle flag out of range.
	= 94: D/A channel not 0, 1 or 2)

MODE 10: *DMA Driven D/A Conversions*

MODE 10 performs *N* D/A conversions based on DMA data transfers. Data is read directly from memory (loaded by MODE 8) and written to the D/A converters. If MODE 8 was set to load data for both D/A's than MODE 10 must also select two channel operation. However, if MODE 8 only loaded data for one D/A channel, MODE 10's channel select controlcan select either channel 0 or 1.

The update rate of the D/A conversions can be set by the internal pacer clock (and MODE 25) or can be synchronized to an external trigger.

EX10.BAS provides an example of the use of MODEs 8, 10, and 25.

The following variables are defined by MODE 10:

MD% -> 10.

DIO%(0) -> Number of conversions.

DIO%(1) -> Segment address of buffer.

DIO%(2) -> Trigger source; three possible, as follows:

DIO%(2) = 0. Maximum speed, external gate. Conversions occur at the rate of 52 kHz (19 µs per conversion), and Pin 30 (COUNTER 2 GATE) on the 3M connector acts as a gate signal.

DIO%(2) = 1. Internal Clock/External Gate. Pin 27 (COUNTER 1 GATE) of the 3M connector is the source for an external gating signal; the conversion clock is derived from the on-board timers. An A/D conversion occurs with each rising edge from the timers and Pin 26 is at a logical high (so long as the number of conversions has not reached the number specified in DIO%(0).

DIO%(2) = 2. Internal clock, no gate. An A/D conversion occurs with each rising edge from the timers and Pin 26 is at a logical high (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(3) -> Recycle flag.

0 = restart on Nth conversion.

X = terminate on Xth cycle of N conversions.

DIO%(4) -> Channel select.

0 = Channel 0.

1 = Channel 1.

2 = Both channels.

FLAG% <- Error Codes.

FLAG%	= 0: No error, OK.
	= -1: MODE out of range.
	= -2: Driver not initialized.
	= 100: Interrupt or DMA already active.
	= 101: Word count zero or >32767)
	= 102: Buffer address out of range)
	= 103: Trigger source not 0, 1, or 2)
	= 104: D/A channel not 0, 1 or 2)

MODE 11: *Cancel DMA or Interrupt*

MODE 11 causes an immediate disable of any running interrupt or DMA operation initiated by MODEs 5,6,9 or 10. The operation will be abandoned at the time MODE 11 executes. MODE 11 also resets the DAS-20 S/H output to a high (sample) state if a DMA MODE (6 or 27) is used to collect samples.

On entry the following parameters should be initialized:

MD% -> 11.

DIO%(0) -> X (where X = any value).

FLAG% <- Errors (if any).

The following error codes apply to MODE 11:

FLAG%	= 0: No error, OK.
	= -1: MODE number out of range, <0 or >29.
	= -2: Driver not initialized.

For an example of the use of MODE 11, see EX5.BAS, EX6.BAS, EX9.BAS or EX10.BAS.

MODE 12: *Determine Status of Interrupt/DMA*

MODE 12 allows you to monitor the status of a background operation initiated by MODEs 5, 6, 10 or 27.

Arguments:

MD% -> 12.

After CALL, data returned is as follows:

DIO%(0) <- operation type in progress.

0 = None.

1 = DMA.

2 = Interrupt.

3 = Block Scan Mode if MODE 27 is in progress.

DIO%(1) <- status of operation.

0 = Done (finished).

1 = Active.

DIO%(2) <- Current word count (Number of conversions so far).

FLAG% <- Errors, if any.

Where

FLAG%	= 0: No error.
	= -2: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.

NOTE: MODE 12 is an optional operation. However, to ensure that the memory segment is completely filled with valid samples, it is wise to use this MODE after calling an interrupt or DMA MODE (MODE 5, 6, 9, 10, or 27) and before returning data from memory segment to the specified array (MODE 13). A simple statement such as the one that follows checks validity of the flag before the program continues.

```
XX10  MD% = 12
XX20  CALL DAS20 (MD%,DIO%(0),FLAG%)
XX30  IF DIO%(1)<>0 THEN GO TO XX10
```

MODE 13: *Transfer Data From Memory to Basic Array*

MODE 13 transfers data from any segment of memory to integer array variables in BASIC workspace. Data in memory derived from MODEs 5, 6, and 27 is in packed form consisting of a word (2 bytes) of A/D data + channel number.

Note how MODE 13 functions. The number of words (or conversions) that you wish to transfer to the data and channel arrays is set into DIO%(0). The segment of memory that you wish to transfer data from is set into DIO%(1). Data can be transferred from the beginning of this segment (DIO%(2) = 0) or any other point. A/D data is transferred to a dimensioned integer array. The transfer will start at the pointer set into DIO%(3). To start at the beginning of the array DT%(*), set DIO%(3) = VARPTR(DT%(0)). If we want to start at the Mth element, then DIO%(3) = VARPTR(DT%(M)). In this way you can move any number of words (conversions) from any source location to any destination location in an array. If you want to keep track of what channel the data came from, dimension an identical CH%(*) array and set DIO%(4) = VARPTR(CH%(0)) or VARPTR(CH%(M)) as required. In this way every element of the DT%(*) array will have a corresponding element of the CH%(*) array that provides the channel data. If the channel data is not required, setting DIO%(4) = 0 suppresses transfer of the channel data.

A/D data is shifted and the MSB complimented if the DAS-20 is operating in bipolar MODE. In unipolar MODE data ranges from 0 to 4095, in bipolar -2048 to +2047. The channel data is masked out and ranges from 0 - 15.

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

1. Do not transfer more words than an array will hold or overun 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.
3. Due to the reformatting of data that this MODE performs, it is not a general purpose block move utility.

On entry the following parameters should be assigned:

MD% = 13 (MODE number).

DIO%(0) -> Number of words to transfer (1 - 32767) (Number of Scans if DIO%(6) > 0).

DIO%(1) -> Buffer segment in memory (0 - 65536).

DIO%(2) -> Starting conversion number (0 - 32767) (Starting Scan # if DIO%(6) > 0).

DIO%(3) -> VARPTR (DataArray1) e.g. VARPTR(DT%(0)).

DIO%(4) -> VARPTR (DataArray2 or Channel array) e.g. VARPTR (CH%(0)) (set = 0 if no channel data required).

NOTE: For DIO%(3) and DIO%(4), if the DIO% array and Data array reside in different data GROUPS, specify the DIO% parameters as follows:

If DIO%(3) = -1 ;

If DIO%(4) = -1 ;

DIO%(7) = segadr(DataArray1) ; (* Segment Of Data Array 1 *)

DIO%(8) = offadr(DataArray1) ; (* Offset Of Data Array 1 *)

DIO%(9) = segadr(DataArray2) ; (* Segment Of Data Array 2 *)

DIO%(10) = offadr(DataArray2) ; (* Offset of DataArray 2 *)

DIO%(5) -> Unipolar/Bipolar Flag.

= 0 (transfer Unipolar data).

= 1 (transfer Bipolar data).

Since Unipolar (0-4095) and Bipolar (-2048 to +2047) data is in different formats it is necessary to tell the transfer routine which type of data is being used. For channel scans which include both unipolar and Bipolar either format can be selected then scaled to the correct form. The scaling conventions are listed below:

Subtract 4096 from Bipolar data transferred in Unipolar MODE for data > 2047.

Add 4096 to Unipolar data transferred in Bipolar MODE when value < 0.

DIO%(6) -> SSH-4 Flag. If the input data has been acquired using the SSH-4, set DIO%(6) to the number of entries in each scan. This removes the Dummy first conversion in each SSH-4 scan. Otherwise, DIO%(6) = 0. This will operate in the same fashion, even when MODE 29 (see MODE 29 for more information) and a DMA MODE (MODE 6 or 27) are used for data collection.

FLAG% = X (value does not matter).

then

```
CALL DAS20 (MD%, DIO%(0), FLAG%)
```

On return the variables contain data as follows:

MD% = 13.

DIO%(0 thru 4) - Unchanged.

The following error codes apply to MODE 13:

FLAG%	= 0: No error.
	= -2: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.
	= 131: Wword count, DIO%(0), zero or negative.
	= 132: Start conversion number, DIO%(2), negative.
	= 133: Unipolar/Bipolar Flag not 0 or 1.
	= 134: SSH-4 scan length out of range.

For an example of the use of MODE 13, see EX5.BAS, EX6.BAS. Also, for an example of the use of MODE 13 in conjunction with the SSH-4, see SSH-4.BAS.

Once data acquisition has been set up as a constant background operation using the recycle options of MODE 5 or 6, a foreground program can be processing the data as it is acquired using MODE 13 to retrieve the data. This is excellent for graphic and "digital oscilloscope" applications.

Note that it is advisable to make the assignments:

DIO%(3) = VARPTR(DT%(0)).

DIO%(4) = VARPTR(CH%(0))

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*, the following sequence would cause a problem.

```
xxx10  DIO%(3) = VARPTR(DT%))
xxx20  I = 3
xxx30  CALL DAS20 (MD%, DIO%(0), FLAG%)
```

As an alternative to MODE 13, you may be tempted to retrieve data using BASIC's *PEEK* command. If interrupt transfers are active, be warned that since PEEK retrieves data a byte at a time, it is possible for an interrupt to occur between reading the Low Byte and High Byte of a data word, thereby changing the halves of a word in mid-flight and causing your time-sequential PEEKs to return erroneous data. Using MODE 13 avoids this problem.

MODE 14: *Read the Digital Inputs*

MODE 14 allows you to read the state of digital inputs DIN0 through DIN7. Data returned can range between 0 and 255 corresponding to all combinations of the 8 input bits.

On entry the following parameters should be initialized:

MD% = 14.

DIO%(0 thru 4) - Value does not matter.

FLAG% = X - Value does not matter.

Then:

```
        CALL DAS20 (MD%, DIO%(0), FLAG%)
```

On return the variables contain data as follows:

MD% = 14.

DIO%(0) contains input data (range 0 - 255).

DIO%(1 thru 4) - Unchanged.

The following error codes apply to MODE 14:

FLAG%	= 0: No error, OK.
	= -2: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.

For an example of digital I/O using MODE 14, see EX14.BAS.

MODE 15: *Write To the Digital Outputs*

MODE 15 is used to write digital data to the 8-bit output port, DOUT0 through DOUT7. Output data is checked to be in the range 0 - 255 and if not an error exit (Error # 151) occurs.

On entry the following parameters should be assigned:

MD% = 15.

DIO%(0) - Output data (range 0-255).

DIO%(1 thru 4) - Value does not matter.

FLAG% = X - Value does not matter.

Then:

```
        CALL DAS20 (MD%, DIO%(0), FLAG%)
```

On return the variables contain data as follows:

MD% = 15 - Unchanged.

DIO%(0 thru 4) - Unchanged.

The following error codes apply to MODE 15:

FLAG%	= 0: No error, OK.
	= -2: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.
	= 151: Output data <0 or >255.

For an example of digital I/O using MODE 15, see EX14.BAS.

MODE 16: *Analog Trigger*

MODE 16 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 16 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 input range for the analog triggering MODE is always set at ±10 Volts, full scale.

The main use for MODE 16 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 16 by hitting any key which will return you to the calling program.

Parameters DIO%(0) thru DIO%(2) control the triggering and select the trigger channel number, the trigger level, and the trigger direction (slope). DIO%(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 DIO%(1) in bit range of -2048 to +2047 corresponding to bipolar input ranges of -10V to +10V. The direction of triggering or slope is controlled by DIO%(2). For instance, if DIO%(1) = 1024 on the ±10V range, the trigger level will be +5.00V; and if DIO%(2) = 0 (positive slope), triggering will take place when the signal exceeds +5.00V. Alternatively, if DIO%(2) = 1 (negative slope), triggering will take place when the trigger signal becomes less than +5.0 Volt.

On entry the following variables should be initialized:

MD% = 16.

DIO%(0) = Channel to be used for measurment (0- 15).

DIO%(1) = Trigger level (-2048 to +2047 bits for -10 to +10V). Triggering is edge sensitive: for example, a request for triggering at 1024 (+5V) with positive edge when the input is already above 5V will not be met until the input cycles below and then back to 5V.

DIO%(2) = Slope (0 = positive, 1 = negative).

DIO%(3) thru DIO%(4) - Value irrelevant.

FLAG% = X - Value does not matter.

Then:

```
CALL DAS20 (MD%, DIO%(0), FLAG%)
```

On return the variables contain data as follows:

MD% = 16 (unchanged).

DIO%(0 thru 4) - unchanged.

The following error codes apply to MODE 16:

FLAG%	= 0: No error, OK.
	= -2: Driver not initialized.
	= -1: MODE number out of range, <0 or >29.
	= 160: Trigger aborted by Keyboard.
	= 161: Trigger channel out of range.
	= 162: Trigger data out of range <-2048 or >2047.
	= 163: Slope data not 0 or 1.

NOTE: If you have selected a positive full-scale value of 2047, you can trigger only on the positive slope. Similarly, if you have selected a negative full-scale value of -2048, you can trigger only on the negative slope.

This MODE will work only with a channel that is a direct input to DAS-20; it does not support the SSH-4 or EXP-20 because it does not issue control of any hardware lines for mux addresses or sample/hold. For an example of the use of MODE 16, see EX16.BAS.

MODE 17: *Initialize Timer - Reset Timer*

Set counters 3, 4 & 5 to ADC time delays.

Arguments:

MD% -> 17.

DIO%(0) -> Don't Care.

FLAG% <- Errors (if any).

= 0: No errors.
= -1: MODE # out of range.

MODE 18: *Set Timer Master MODE Register*

MODE 18 sets the AMD9513 to a user specified configuration. Data bus width set to 8 bits, Data pointer auto increment disabled, Binary division. Executing MODE 18 resets the entire AMD9513 counter chip and enables Fout.

Arguments:

MD% -> 18.

DIO%(0) -> Fout divider ratio (1 - 16).

DIO%(1) -> Fout source.

0 = F1

1 = Source 1

2 = Source 2

3 = Source 3 ⎫
4 = Source 4 ⎬ No Connection.
5 = Source 5 ⎭

6 = Gate 1

7 = Gate 2

8 = Gate 3 ⎫
9 = Gate 4 ⎬ No Connection
10 = Gate 5 ⎭

11 = F1

12 = F2

13 = F3

14 = F4

15 = F5

DIO%(2) -> Compare 2 disable/enable (0/1).

DIO%(3) -> Compare 1 Disable/Enable (0/1).

DIO%(4) -> Time Of Day MODE control.

0 = TOD disabled

1 = TOD Enabled /5 input

2 = TOD Enabled /6 input

3 = TOD Enabled /10 input

FLAG% <- Errors

= 0: No error.
= -2: Driver not initialized.
= -1: MODE out or range <0 or >29.
= 181: DIO%(0) out of range.
= 182: DIO%(1) out of range.
= 183: DIO%(2) not 0 or 1.
= 184: DIO%(3) not 0 or 1.
= 185: DIO%(4) out of range <0 or >3.

MODE 19: *Set Counter 'N' MODE Register*

MODE 19 sets Counter N to to user specified value.

Arguments:

MD% -> 19.

DIO%(0) -> Counter Number (1 - 5). Counters 3, 4, and 5 are used for timing functions in A/D or D/A MODEs. Do not access counters 3, 4, and 5 during an acquisition cycle or unpredictable data/timing will occur.

DIO%(1) -> Gating Control (0 - 7).

0 = No gating.

1 = Active high level TCN-1.

2 = Active high level Gate N+1.

3 = Active high level Gate N-1.

4 = Active high level Gate N.

5 = Active low level Gate N.

6 = Active high edge Gate N.

7 = Active low edge Gate N.

DIO%(2) -> Count Edge positive/negative (0/1).

DIO%(3) -> Count Source Selection (0 - 15).

0 = TCN-1.

1 = Source 1.

2 = Source 2.

$\left.\begin{array}{l}\text{3 = Source 3}\\\text{4 = Source 4}\\\text{5 = Source 5}\end{array}\right\}$ No Connection

6 = Gate 1.

7 = Gate 2.

$\left.\begin{array}{l}\text{8 = Gate 3}\\\text{9 = Gate 4}\\\text{10 = Gate 5}\end{array}\right\}$ No Connection

11 = F1.

12 = F2.

13 = F3.

14 = F4.

15 = F5.

DIO%(4) -> Disable/Enable special gate (0/1).

DIO%(5) -> Reload from Load/ReLoad from Load or Hold (0/1).

DIO%(6) -> Count once/Count repetitively (0/1).

DIO%(7) -> Binary count/B.C.D. count (0/1).

DIO%(8) -> Count down/Count up (0/1).

DIO%(9) -> Output control (0 - 5, except 3).

0 = Inactive.

1 = Active high terminal count pulse.

2 = Terminal count toggled.

3 = *** Illegal ***

4 = Inactive, output high impedence.

5 = Active low terminal count pulse.

FLAG% <- Errors.

= 0: No error.
= -2: Driver not initialized.
= -1: MODE out or range <0 or >29.
= 190: DIO%(0) out of range <1 or >5.
= 191: DIO%(1) out of range <0 or >7.
= 192: DIO%(2) not 0 or 1.
= 193: DIO%(3) out of range <0 or >15.
= 194: DIO%(4) not 0 or 1.
= 195: DIO%(5) not 0 or 1.
= 196: DIO%(6) not 0 or 1.
= 197: DIO%(7) not 0 or 1.
= 198: DIO%(8) not 0 or 1.
= 199: DIO%(9) not 0, 1, 2, 4 or 5.

MODE 20: *Set Multiple Counter Control Registers - To User-Specified Value*

Arguments:

MD% -> 20.

DIO%(0) -> Command (1 - 6).

1 = Arm selected counter.

2 = Load source to counter.

3 = Load and arm counter.

4 = Disarm and save counter.

5 = Latch counter to hold register.

6 = Disarm counter.

DIO%(1) -> Select Counter 1 (0/1).

DIO%(2) -> Select Counter 2 (0/1).

DIO%(3) -> Select Counter 3 (0/1).

DIO%(4) -> Select Counter 4 (0/1).

DIO%(5) -> Select Counter 5 (0/1).

Where 0 = not selected; 1 = selected. (Do not select counters 3, 4, and 5 during paced A/D or D/A conversion MODEs; otherwise, unpredictable timing or results can occur.)

FLAG% <- Errors (if any).

= 0: No errors.
= -2: Driver not initialized.
= -1: MODE out of range <0 or >29.
= 200: Command # out of range.
= 201: Select Counter 1 not 0 or 1.
= 202: Select Counter 2 not 0 or 1.
= 203: Select Counter 3 not 0 or 1.
= 204: Select Counter 4 not 0 or 1.
= 205: Select Counter 5 not 0 or 1.

MODE 21: *Set Counter N Load Register - To User-Specified Value*

Arguments:

MD% -> 21.

DIO%(0) -> Counter # (1 - 5). Do not select Counters 3, 4, or 5 during paced A/D or D/A conversion MODEs or you will get unpredictable timeing or results.

DIO%(1) -> 16-bit value.

FLAG% <- errors (if any).

= 0: No errors.
= -2: Driver not initialized.
= -1: MODE out of range <0 or >29.
= 210: Counter out of range <1 or >5.

MODE 22: *Read Counter 'N' Hold Register*

Arguments:

MD% -> 22.

DIO%(0) -> Counter # (1 - 5).

DIO%(1) <- 16-bit value.

FLAG% <- Errors (if any).

= 0: No errors.
= -2: Driver not initialized.
= -1: MODE out of range <0 or >29.
= 220: Counter out of range <1 or >5.

MODE 23: *Measure Frequency*

MODE 23 returns frequency expressed as cycles per gating interval. See the file EX23.BAS for an example of MODE 23.

NOTE: The output of C/T # 1, must be physically jumpered to the Gate of C/T # 2 for proper operation.

Arguments:

MD% -> 23.

DIO%(0) -> Gating interval in ms (0 - 32767).

DIO%(1) -> Select source input signal (1 - 3).

1 = Source 1.

2 = Source 2.

3 = Gate 1.

DIO%(2) <- Count accumulated during gating interval.

FLAG% <- Errors (if any).

= 0: No errors.
= -2: Driver not initialized.
= -1: MODE out of range <0 or >29.
= 230: Gate interval out of range.
= 231: Invalid source # >3 or <1.

MODE 24: *Set A/D Pacer Clock*

The A/D timer allows the analog inputs to be updated based on the DAS-20's on-board 5 MHz clock. The A/D pacer clock uses the AMD-9513 counter chip to divide the 5 MHz clock. The board uses 2 of the AMD-9513's 16 bit counters allowing the 5 Mhz clock to be divided by any value from 50 to 4.29 E9 (100 kHz to .00116 Hz respectivley). The actual divisor number will be DIVISOR 1, (DIO%(0)) multiplyed by DIVISOR 2, (DIO%(1). To obtain an update rate of 100 KHz (5 MHz divided by fifty), enter a divisor of 10 in DIO%(0) and 5 in DIO%(1), or (5 in DIO%(0) and 10 in DIO%(1)). For 25 KHz, use DIO%(0) = 4, DIO%(1) = 50, etc. Note that DIO%(1) may be set to zero or one if only one divisor is required.

Arguments:

MD% -> 24.

DIO%(0) -> Divisor 1 (50 - 65536).*

DIO%(1) -> Divisor 2 (0 - 65536; typically set to 0 in sample programs).*

FLAG% <- Error Codes.

= 0: No errors.
= -2: Driver not initialized.
= -1MODE out of range <0 or >29.
= 241: Divisor 1 out of range.
= 242: Divisor 2 out or range.

NOTE: Basic integers greater than 32767 must be entered in 2s complement form. For example, if DIO%(0) is 50,000, form the 2s complement number by subtracting 50,000 from 32767; the result is -17233. Refer to the BASIC manual for further explanation of intetgers.

Examples of programs using MODE 24 are included in example programs EX4.BAS, EX5.BAS, and EX6.BAS.

MODE 25: *SET D/A or Block Scan Pacer Clock*

The D/A timer allows analog outputs to be updated based on the DAS-20's on-board 5 MHz clock. The D/A pacer clock uses the AMD-9513 counter chip to divide the 5 MHz clock by any value from 20 to 2^{32} (250 Khz to Once every 14 minutes). Note that MODE 25 is also used to set the scan rate for MODE 27. The actual divisor is transfered in DIO%(0) and DIO%(1). If the total divisor is less than 65536 then DIO%(1) can be set to 0. The AMD-9513's counter 2 is set to divide the 5 MHz by Divisor # 1. If Divisor # 2 is not zero, then AMD-9513 counter 1 is automatically connected to counter 2, and the total division ratio is the 5 MHz divided by Divisor 1 multiplied by Divisor 2. To obtain an update rate of 100 KHz, set Divisor # 1 equal to 50, and Divisor # 2 equal to 0. Alternatively you could set Divisor # 1 to 10, and Divisor # 2 to 5 and obtain the same result. Note that MODE 25 uses AMD-9513 channel 2 (and channel 1 if Divisor # 2 is not zero). Counter 2 (and 1 if non-zero Divisor # 2) cannot be used for other timing/counting applications while MODE 25 is being used to pace D/A conversions.

Arguments:

MD% -> 25.

DIO%(0) -> Divisor # 1 (1 to 65536).

DIO%(1) -> Divisor # 2 (0 - 65536; typically set to 0 in example programs).

FLAG% <- Errors (if any).

NOTE: Basic integers greater than 32767 must be entered in 2s complement form. For example, if DIO%(0) is 50,000, form the 2s complement number by subtracting 50,000 from 32767; the result is -17233. Refer to the BASIC manual for further explanation of intetgers.

The following error codes apply to MODE 25:

FLAG%	= 0: No error.
	= -2: Driver not initialized.
	= -1: MODE out or range <0 or >29.
	= 251: Divisor # 1 out of range.
	= 252: Divisor # 2 out of range.

MODE 26: *Stop A/D and D/A Pacer Clocks*

MODE 26 can be used to stop either the A/D or D/A pacer clocks. MODE 24 or 25 can then be executed to restart the timers.

MD% -> 26.

DIO%(0) -> A/D or D/A clock.

0 = Stop A/D timer.

1 = Stop D/A timer.

2 = Both.

FLAG% <- Errors.

= 0: No error.
= -2: Driver not initialized.
= -1: MODE out or range <0 or >29.
= 261: DIO%(0) not 0, 1, or 2.

MODE 27: *Perform N Scans of the A/D Control Queue*

MODE 27 performs a block scan of channels. On an input trigger (either from the Pacer Clock or an external trigger), the entire Sampling Queue is scanned at the DAS-20's maximum sample rate, and the input data is transfered to memory via DMA; the Pacer Clock for MODE 27 is set using MODE 25. The Board then waits for the next trigger and repeats the process. Sampled channels must have been set with a MODE 1 call or a Flag% error will result. This function is useful when all channels must be sampled at the same time but when the sample rate need not be extremely high. Note that the same memory limitations apply to MODE 27 as to MODE 6; memory used by MODE 27 is limited to one Page (64 KB). The scanning is controlled by a clean gating signal (active high) from the external connection. See the DIO%(2) below for details.

In the Block Scan Mode, Counters 3, 4, and 5 will control the conversion delay timings. Refer to Section 6.2 (Chapter 6) for details. The delays are needed for proper settling when the analog inputs are switching from channel to channel.

The start of Scan can be synchronized to the Counter #2 Pacer Clock (using MODE 25) or to an external trigger. Note that for proper operation the block scan time (# of conversions times the internal sample rate) must be less than the period of the input trigger (the block scan must be completed once before another scan can be performed again). Because MODE 27 uses interrupts to perform block operations, the maximum throughput for block conversions is machine-dependent. A rate of 3-5 KHz is typical for interrupt driver operations.

MODE 27 is ideally suited for use with the SSH-4 simultaneous sample & hold board. See Appendix D for details of the SSH-4.

MD% -> 27.

DIO%(0) -> Total # of Queue scans.

DIO%(1) -> Segment Address of Buffer.

DIO%(2) -> Trigger source; three possible, as follows:

DIO%(2) = 0. Maximum speed, external gate. Conversions occur at the rate of 52 kHz (19 µs per conversion), and Pin 30 (COUNTER 2 GATE) on the 3M connector acts as a gate signal.

DIO%(2) = 1. Internal Clock/External Gate. Pin 27 (COUNTER 1 GATE) of the 3M connector is the source for an external gating signal; the conversion clock is derived from the on-board timers. An A/D conversion occurs with each rising edge from the timers and Pin 27 is at a logical high (so long as the number of conversions has not reached the number specified in DIO%(0)). When Internal Gate is used and Internal Clock is set by MODE 25, you may not set the Divisor #2 in MODE 25 to '0.' A minimum of '2' must be used for DIO%(1) in MODE 25. This may compromise your selection of sampling rate, but it ensures application of the proper gating signal to Pin 27.

DIO%(2) = 2. Internal clock, no gate. An A/D conversion occurs with each rising edge from the timers and Pin 26 is at a logical high (so long as the number of conversions has not reached the number specified in DIO%(0)).

DIO%(3) -> Single Cycle/Recycle operation:

DIO%(3) = 1: Single Cycle. After completion of the number of scans specified, DMA ceases, an interrupt is generated and DMA is disabled, setting the DMA status to zero.

DIO%(3) = 0 : Recycle. In this case data is continuously written to the same memory. DIO%(0) corresponds to the memory "buffer" length. This corresponds to a D.M.A. auto-initialize operation. To end the acquisition, MODE 11 must be run.

DIO%(4) = X (value does not matter).

DIO%(5) = X (Value does not matter).

DIO%(6) <- Returns number of entries in block scan queue if SSH-4 flag is set; otherwise returns a 0.

FLAG% = X Errors, if any.

= 0: No error.
= -2: Driver not initialized.
= -1: MODE out or range <0 or >29.
= 271: # of scans out of range.
= 272: Buffer segment out of range.
= 273: Trigger source out of range.
= 274: Recycle flag not 0 or 1.

MODE 28: *Sample EXP-20 Inputs*

Use of the EXP-20 with the DAS-20 is greatly simplified by using the MODE 28 call. MODE 28 allows a complete or partial scan of an EXP-20 by combining A/D input control with Digital output control. MODE 28 allows the user to scan from 1 to 256 EXP-20 channels. When the number of channels to scan is larger than 16, MODE 28 assumes that there is more than one EXP-20 installed. As more EXP-20s are cascaded, it is important to note that the first EXP-20 must be connected to DAS-20 Channel #0, the second to DAS-20 Channel #1, and so on. **Note that for proper operation, ARRAY%(N) must be dimensioned for the full array with the statement DIM ARRAY%(N) before executing the MODE 28 call.** Also, no new simple variable should be defined between the VARPTR statement and the MODE 28 call. For best (safest) results, execute the VARPTR statement immediately before calling MODE 28.

The MODE 28 arguments are:

MD% -> 28.

DIO%(0) -> DAS-20 Gain/Range.

INPUT RANGE	GAIN	UNI/BIPOLAR	DIO%(1)
0 to +10V	X1	Unipolar	0
±10V	X.5	Bipolar	1
0 to +10V	X1	Unipolar	2
±5V	X1	Bipolar	3
0 to +1V	X10	Unipolar	4
±0.5V	X10	Bipolar	5
0 to 100mV	X100	Unipolar	6
±50mV	X100	Bipolar	7

DIO%(1) -> "N", the total number of channels.

DIO%(2) -> VARPTR(ARRAY%(0))- the array pointer to the array the data will be loaded into.

NOTE: If the DIO% array and the Data Array (ARRAY%(0)) reside in different data GROUPS, specify the DIO% parameters as follows:

DIO%(2) = -1

DIO%(3) = segadr(DataArray) ; (* Segment Of The DataArray *)

DIO%(4) = offadr(DataArray) ; (* Offset Of The Data Array *)

The data returned is of the form 0 to 4095 for unipolar inputs, and -2048 to +2047 for Bipolar inputs. The table below shows the scan sequence/assignments for MODE 28:

DATA VARIABLE	DAS-20 CHANNEL	EXP-20 CHANNEL	
ARRAY%(0)	0	A	
ARRAY%(1)	0	B	
ARRAY%(2)	0	C	
ARRAY%(3)	0	D	
ARRAY%(4)	0	E	EXP-20
ARRAY%(5)	0	F	
ARRAY%(6)	0	G	NUMBER
ARRAY%(7)	0	H	
ARRAY%(8)	0	I	ONE
ARRAY%(9)	0	J	
ARRAY%(10)	0	K	
ARRAY%(11)	0	L	
ARRAY%(12)	0	M	
ARRAY%(13)	0	N	
ARRAY%(14)	0	O	
ARRAY%(15)	0	P	
ARRAY%(16)	1	A	
ARRAY%(17)	1	B	
ARRAY%(18)	1	C	
ARRAY%(19)	1	D	
ARRAY%(20)	1	E	EXP-20
ARRAY%(21)	1	F	NUMBER
ARRAY%(22)	1	G	TWO
.	.	.	
.	.	.	
.	.	.	
.	.		
.			
ARRAY%(N-1)	N/16		
P*			

* The last EXP-20 channel scanned will be Channel P only if the total number of channels is evenly divisable by 16.

FLAG% <- Errors.

= 0: No error.
= -2: Driver not initialized.
= -1: MODE out or range <0 or >29.
= 281: DIO%(0) not 0 through 7.
= 282: Number of channels out of range.

MODE 29: Set SSH-4 Flag

MODE 29 (if equal to 1) sets the Block Scan flag in MODE 27 so that MODE 27 is able to return (after a CALL) the total number of elements in the Scan Queue. The value returned in the DIO%(6) (of MODE 27) variable is the total number of elements in queue, including the dummy first channel.

MODE 29 does not affect MODE 13 data retrieval. The user must set DIO%(6) of MODE 13 to the proper value to ensure that the dummy channel is dropped when transferring to an array. See

MODEs 27 and 13 for further info on blockscan and data retrieval.

Arguments:

MD% -> 29.

DIO%(0) -> SSH-4 Flag.

0 = Normal operation.

1 = Set SSH-4 Flag.

FLAG% <- Errors.

= 0: No error.
= -2: Driver not initialized.
= -1: MODE out or range <0 or >29.
= 291: DIO%(0) not 0 or 1.

* * * * *

CHAPTER 6

PROGRAMMING THE DAS-20 WITHOUT THE ASSEMBLY DRIVERS

6.1 GENERAL

Because the DAS-20 is a complex device, MetraByte recommends that programmers use the free assembly driver for BASIC, or the PCF-20 Fortran, "C," and Pascal packages wherever possible. The source code for the driver is provided in the SLD-20 package. For applications that require programming in languages that are not currently supported, the easier course may be to modify the source code of the driver rather than to write the routine from scratch.

However, for programming the board directly, this chapter provides expanded coverage of the register mapping in Chapter 3. Though most of the DAS-20 controls are straight-forward (digital inputs, digital outputs, software controlled A/D and D/A conversions), additional information on DMA and interrupt-driven operation and specifically on A/D and D/A triggering is required and is provided herein. For a block diagram of the DAS-20, refer to Figure 6-1.

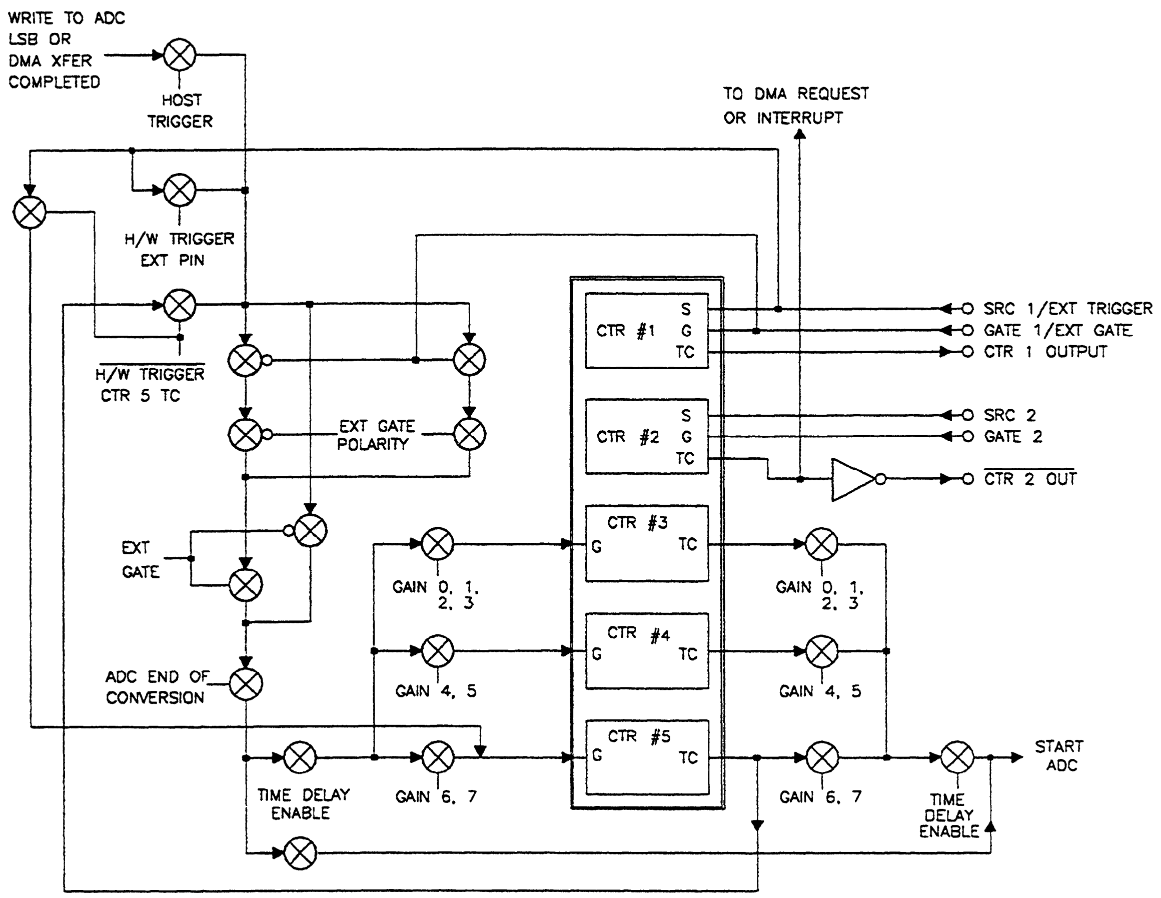


Figure 6-1. Block diagram of DAS-20 circuitry.

6.2 A/D CONVERSIONS

DAS-20 hardware configuration allows a high degree of flexibility in timing the Start-of-Conversion and method of data transfer. The following material describes a few of the possible combinations. Figure 7-1 shows the DAS-20's key timing components.

Setting the A/D Control Register.

The ADC Control Register (Base Address +3) programs the ADC's modes of operation. The two MSBs (most significant bits) control the operation of the ADC instruction queue, as noted in the following table.

BIT 7	BIT 6	MODE OF OPERATION
0	0	Queue is loaded with one instruction and is static.
0	1	Queue is written to or read from and auto-incremented after each read or write. This mode is used to load or view the instruction queue.
1	0	Queue is enabled and auto-increments after each ADC Start Of Conversion. Queue halts when the End Of Queue flag is encountered.
1	1	Queue is enabled and auto-increments after each ADC Start Of Conversion. Queue resets back to location zero when EOQ flag is encountered.

Bit 5 of the ADC Control Register sets the DMA Enable/Disable signal. Bit 4 enables Time-Delayed Start-Of-Conversion (if set to 1) or no Timed-Delayed conversion (if 0). Bits 3 - 0 control gating and triggering as noted below:

BIT 3	BIT 2	TRIGGERING
0	x	Host initiated.
1	0	Hardware initiated, Counter 5 terminal count.
1	1	Hardware initiated, direct external trigger input (positive pulse width must be 200ns to 5µs at rate of 100kHz or less.

BIT 1	BIT 0	GATING*
0	x	Internal.
1	0	External, low active gate.
1	1	External, high active gate.

* All conversion are additionally gated by ADC End-Of-Conversion.

A/D Triggering Options:

Host Initiated With/Without Gating

In this mode, the ADC receives the Start-Of-Conversion signal from programmed I/O or DMA Acknowledge, depending on the transfer mode used. The sequence of programming is as follows:

1. Write a 0 to Bit 3 of the ADC control register.
2. A Write to Location 0 or completion of DMA request starts a conversion.
3. If External gating is required program Bits 1 and 0 of the ADC control register to select Ext gate Enable and/or Gate level, respectively.

External Trigger (Using 9513) With/Without External Gating

In this mode, the ADC receives the Start-Of-Conversion signal from CTR 5 terminal count output of the STC, which is set to the edge-triggered one-shot mode. The sequence of programming is as follows:

1. Program CTR 5 of the STC to Gate off the Pos or Neg going edge of the Ext Trigger pin. This counter is programmed to Mode R (Hardware retriggerable One-Shot). The load register is programmed to 0002H with the Count source set to F1.
2. Write 1 and 0 to Bits 3 and 2 of the ADC Control Register, respectively. This triggers ADC from CTR 5 output.
3. If external gating is required, program Bits 1 and 0 of the ADC control register to select Ext Gate Enable and/or Gate level, respectively.

External Trigger (Ext Trig pin) With/Without Gating

In this mode, the ADC receives the Start-Of-Conversion signal from the External Trigger Pin. This is a level-sensitive trigger, requiring the trigger input to be a 1 - 5µs active high pulse with a a frequency 100 kHz or less.

1. Write 1 and 1 to bits 3 and 2 of the ADC Control Register, respectively. This triggers ADC directly from the External Trigger pin.
2. If External gating is required program Bits 1 and 0 of the ADC control register to select Ext Gate Enable and/or Gate level, respectively.

Internal Pacing (Using STC) With/Without External Gating

In this mode, the ADC receives the Start Of Conversion signal from the CTR 5 terminal count output of the STC, which is set to the gated or free-running rate generator. The sequence of programming:

1. Program CTR 5 Mode register to mode D. Load Register 5 is programmed with a 5 MHz/Sample rate (in Hz).
2. Write 1 and 0 to bits 3 and 2 of the ADC control register, respectively. This will trigger ADC from

CTR 5 output.

3. If External gating is required program Bits 1 and 0 of the ADC control register to select Ext Gate Enable and/or Gate Level, respectively.

ADC Instruction Queue Programming

Program the ADC Instruction Queue by first setting the ADC control register to the Queue Programming Mode. This requires setting Bits 7 and 6 to 0 and 1, respectively. Any time the ADC Control Register is read, the Queue is reset to zero. Each queue instruction word controls the channel to be measured by the ADC and the gain and range of the instrumentation amp for that measurement. The LS bit of the instruction queue word is always reserved for the End Of Queue flag. This bit (when set to a 1) indicates the end of the queue sequence and either halts the queue from moving or resets it back to zero, according to how bits 7 and 6 of the ADC Control Register are set. The tables below show the programming of the Instruction Queue words:

Single-Ended Input Mux Mode-

BIT 7	BIT 6	BIT 5	BIT 4	INPUT CHANNEL
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	11
1	0	1	1	12
1	1	0	0	13
1	1	0	1	14
1	1	1	0	15
1	1	1	1	16

Differential Input Mux Mode-

BIT 7	BIT 6	BIT 5	BIT 4	INPUT CHANNEL
x	0	0	0	1
x	0	0	1	2
x	0	1	0	3
x	0	1	1	4
x	1	0	0	5
x	1	0	1	6
x	1	1	0	7
x	1	1	1	8

Gains and Ranges-

BIT 3	BIT 2	BIT 1	UNI/BIP	GAIN	INPUT RANGE
0	0	0	U	x1	0 to +10V
0	0	1	B	x.5	±10V
0	1	0	U	x1	0 to +10V
0	1	1	B	x1	±5V
1	0	0	U	x10	0 to +1V
1	0	1	B	x10	±0.5V
1	1	0	U	x100	0 to 100mV
1	1	1	B	x100	±50mV

The gains and ranges stated are for the standard DAS-20 configuration and may be altered for custom configurations. Remember to set Bit0 high on the last value written to the queue!

Mux Mode Switch Setting Status is examined by reading the LS bit of the Interrupt Control Register. If this bit is zero, the input Mux is set to 16 single-ended inputs. If it is set to one, then the input Mux is set to 8-differential-input mode.

ADC Data input Format:

The ADC MS Byte contains the eight MS bits of the ADC measurement. The ADC LS Byte contains the 4 LS bits of the ADC measurement and the input channel data. The entire 2-Byte value is left-justified with the lowest 4 bits representing the channel data.

If the measurement was performed while the Instrumentation amp was set to unipolar range, the data output will be in 12-bit unsigned binary format. If the measurement was performed while the instrumentation amp was set to the bipolar range, the data output will be in 12-bit 2s complement format.

A/D Conversion Data Transfer Modes

The DAS-20 ADC can be programmed to operate in one of three basic data-transfer modes. These modes are described below.

Standard Read/Write Mode.

This mode of data transfer is purely software-initiated by writing to location 0 (Base Address + 0) to initiate the ADC conversion. The data may then be read at Location 0 and 1 (Base Address + 0 for low byte and B.A. + 1 for high byte). This word has the 12 bits of data left-justified and the associated channel number in the low order nibble.

Interrupt Driven Mode.

This mode of data transfer is driven by either a host-initiated or a hardware-initiated interrupt. This mode is usually used when the System Timing Controller (STC) is used to delay or pace the start of conversion, so that the program does not have to wait around for the End Of Conversion. The Interrupt Service Routine should do the following:

1. Read Bit 1 of the Interrupt Control Register (Base Address + 4) to determine if in fact the ADC was the interrupting device (if there are more than one possible interrupting devices).
2. Read the ADC data LSB and MSB
3. Write to clear the interrupt latch (bit 1 of ICR) being careful to write the proper data into the upper six bits of the ICR.

Interrupt Control Register.

The six MSBs of this register program the interrupt function of the DAS-20. Bit 7 enables the interrupt function. Bits 6, 5, and 4 set the interrupt level to IRQ2 - IRQ7. Illegally selected levels 0 and 1 do nothing. The remaining Bits 3 and 2 select the interrupt source as noted below:

BIT 3	BIT 2	INTERRUPT SOURCE
0	0	ADC end of conversion.
0	1	Timer #2 output.
1	0	End of Queue.
1	1	DMA terminal count.

DMA Driven Mode.

This mode of data transfer is initiated by the ADC End of Conversion, which causes a DACK (DMA request). This mode will be used when high data-transfer rates are desired. During use of this mode, the ICR can be set to generate an interrupt at the Terminal Count of the DMA sequence or the End Of Queue flag of the ADC instruction RAM.

DMA Control Register.

The four MSBs of this register (Base Address + 5) program the DMA function of the DAS-20. Bit 7 enables the DMA function. Bit 6 sets the DMA Channel to either 1 (if zero) or 3 (if one). The re maining bits, 5 and 4, set the transfer mode as noted in the following table.

BIT 5	BIT4	TRANSFER MODE
0	0	2-Byte ADC data transfer to main memory initiated by ADC end of conversion.
0	1	2-Byte DAC 0 transfer from main memory initiated by Timer #2 output.
1	0	2-Byte DAC 1 data transfer from main memory initiated by Timer #2 output.
1	1	4-Byte DAC data transfer from main memory initiated by Timer #2 output.

DMA Programming Sequence (for ADC)

1. Initially the DMA Active bit (Bit 5) of the ADC control register must be set to inactive (low), the DMA enable of the DMA control register (bit 7) is set disabled (low).
2. The Trigger mode, Gating mode, Queue mode, and STC must be programmed to desired settings.
3. The Interrupt control register must be programmed, if applicable.
4. The DMA control register must be programmed to the DMA Channel 1 or 3, DMA transfer mode set to transfer on ADC End of Conversion, and the DMA enable bit must be enabled (high).
5. Next the host DMA controller (8237) must be programmed to the required mode and setup.
6. Finally, the DMA active bit of the ADC control register must be activated (high) until the operation is complete.

Block Scan ADC Conversions.

The block scan mode allows the timing interval of the A/D conversions to be "tied" to the input gain. The settling time of the instrumentation amplifier increases as the gain increases. So for maximum speed sampling, the DAS-20 allows input sample rates to be set based on the input gain. This block scan mode assures maximum sample rate is maintained, and the input amplifier is allowed enough time to settle. In this mode Counters 3, 4, and 5 must be programmed to be hardware retriggerable One-Shots (Mode R) with active high terminal counts with active high-edge gating (GATE N, Active high edge).

When using the AMD 9513 (analogous to Mode 27 in the assembly driver) to delay the Start Of Conversion, three of the five available 16 bit counters come into play. Just which counters are involved is set by the hardware, 9513 configuration, and the gain setting at the time the trigger occurs. If the Time Delay is enabled the gains stated below use their respective sections to delay start of conversion to allow sufficient settling times as stated below.

GAIN	COUNTER	LOAD VALUE	SPEED
X.5	TD 3	0008(Dec)	100 kHz
X1	TD 3	0008(Dec)	100 kHz
X10	TD 4	0020(Dec)	80 kHz
X100	TD 5	0083(Dec)	40 kHz

Remember the time-delay sequence must be initiated by a trigger, either host or hardware generated. If Time Delay bit is enabled, Counter 5 cannot be used for pacing conversions.

The other two counters contained in the STC can be programmed to pace the overall ADC block scans at intervals or at pre-designated real times (like the 1:00AM,2:00AM,etc...).

6.3 D/A CONVERSIONS

Programming the DAS-20 DACs

Both of the DAS-20's DACs are switch selectable to three different gains and ranges. These are noted in the table below:

RANGE SWITCH	GAIN SWITCH	DAC OUTPUT	DATA RANGE
Right	Right	0 to +10V	0 to 4095
Left	Right	±5V	-2048 to 2047
Right	Left	Illegal setting	
Left	Left	±10v	-2048 to 2047

Notes:

* DAC Data output format is always in 2s compliment format and right justified. Conversion from 2s compliment to Bipolar Offset Binary is done automatically when the DAC range is set to Bipolar.
* DAC writing should always be performed by writing the LSB first then the MSB. When the MSB is written the LSB and MSB which are stored in a temporary register are are simultaneously converted into the DAC output voltages.
* The status of the DAC output Range and Gain switches examined by reading the four LSBs of the DMA control register. Their meanings are stated below:

DAC# 0

BIT 2	BIT 0	MEANING
0	0	0 to +10V Switch Setting
0	1	±5V Switch Setting
1	0	Illegal Switch Setting
1	1	±10V Switch Setting

DAC# 1

BIT3	BIT1	MEANING
0	0	0 to +10V Switch Setting
0	1	±5V Switch Setting
1	0	Illegal Switch Setting
1	1	±10V Switch Setting

DMA Programming Sequence (for D/As)

1. Initially, the DMA Active bit (bit 5) of the ADC control register must be set to inactive (low), the DMA enable of the DMA control register (bit 7) is set disabled (low).
2. Counter 2 of the STC must be programmed as a rate generator. The output of CTR 2 generates the DMA request that starts a DMA transfer sequence to the DAC(s).
3. The DMA control register must be programmed to the DMA Channel 1 or 3, DMA transfer mode set to transfer on CTR 2 Terminal Count for DAC 0,DAC 1, or BOTH. Then the DMA Enable bit must be enabled (high).
4. Next the host DMA controller (8237) must be programmed to the required mode and setup.
5. Finally the DMA active bit of the ADC control register must be activated (high) until the operation is complete.

D/A Converter Timing Control

Counter 2 is used for Pacing DACs during DMA tranfers to the DAS-20. When using this mode of transfer, Counter 2 should be set to perform as a rate generator with active high terminal count output (Mode D or E).

* * * * *

CHAPTER 7

ADC & DAC NOTES

7.1 ADC (A/D CONVERTER) INPUT CONNECTIONS

8-Channel Differential vs. 16-Channel Single-Ended

The DAS-20 offers optional switch-selected 8-channel differential or 16-channel single-ended input configurations. The toggle switch that controls these options is located as shown in Figure 2-1, of Chapter 2. As instructed in Chapter 2, set the switch to the left side (away from the DAS-20 I/O connector) for 8-channel differential, or to the right (toward the DAS-20 I/O connector) for 16-channel single-ended operation.

Single-ended configuration earns its name from the fact that you have only one input relative to ground. This means the input measurement is the voltage difference between either of the two operational amplifier inputs and ground. A differential input measures voltage difference between two inputs of an operational amplifier. Obviously, operating 16 channels single-endedly provides more input channels; but in this configuration, the inputs are suitable only for what is known as "floating" sources. A floating signal source does not have any connection to ground at the signal source and can be wired to the inputs as shown in Figure 7-1.

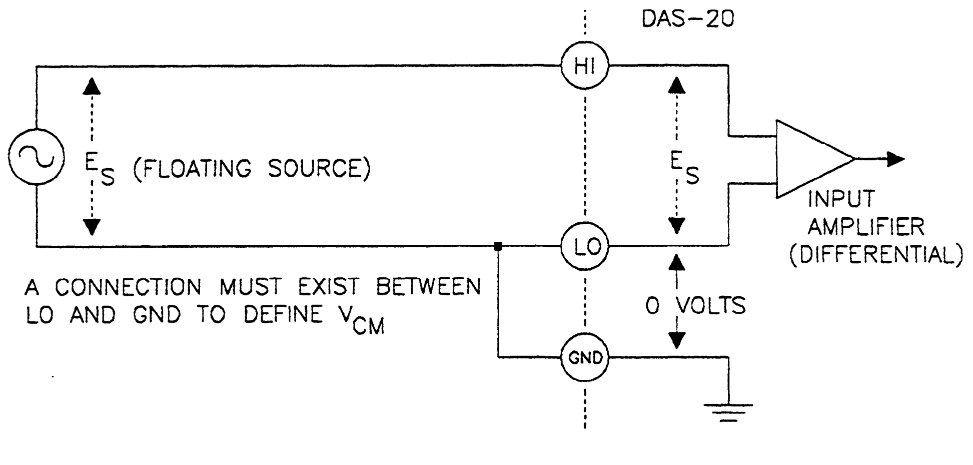
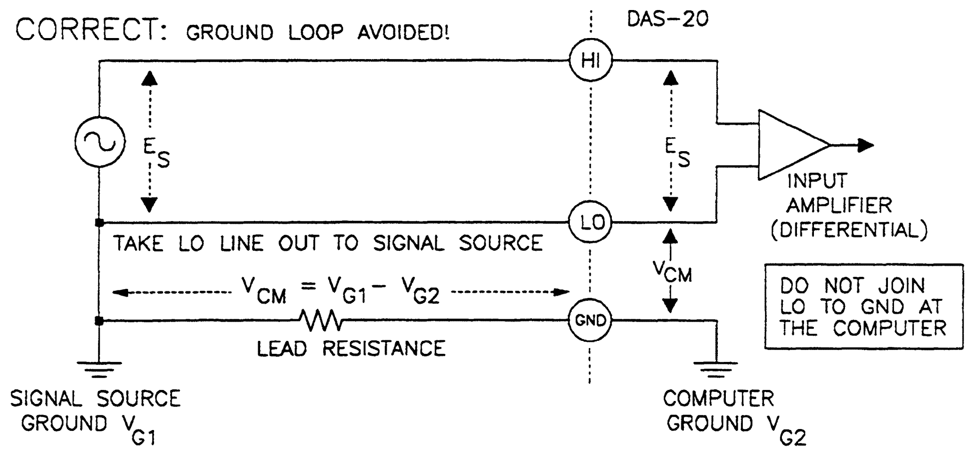

DAS-20
HI
E_S (FLOATING SOURCE)
E_S
INPUT AMPLIFIER (DIFFERENTIAL)
LO
A CONNECTION MUST EXIST BETWEEN LO AND GND TO DEFINE V_{CM}
0 VOLTS
GND

Figure 7-1. Connections to an isolated input source.

If one side of the signal source connects to a local ground, the 8-channel differential configuration should be used. A differential input responds only to difference signals between the high (HI) and low (LO) inputs. In practice, the signal-source ground and the DAS-20 (computer) ground are not at exactly the same voltage because they are connected through the ground returns of the equipment and the building wiring. The difference between the ground voltages (see Figure 7-2) forms a common mode voltage i.e. a voltage common to both inputs (Vcm). A differential input is designed to reject this common mode error up to a certain limit. In the case of DAS-20, the inputs are designed to handle a common mode limit of ±12 volts. If a ground referred signal source is wired to a single ended input as shown in Figure 7-3, the difference in voltage between grounds appears in series with the input signal, causing errors, noise, unreliable readings by what is known as a *ground loop*. For this reason, you should avoid connecting a ground referred signal to a single-ended input unless its ground is the same as the computer's.


CORRECT: GROUND LOOP AVOIDED!
DAS-20
HI
LO
GND
E_S
E_S
INPUT AMPLIFIER (DIFFERENTIAL)
TAKE LO LINE OUT TO SIGNAL SOURCE
$V_{CM} = V_{G1} - V_{G2}$
V_{CM}
DO NOT JOIN LO TO GND AT THE COMPUTER
LEAD RESISTANCE
SIGNAL SOURCE GROUND V_{G1}
COMPUTER GROUND V_{G2}

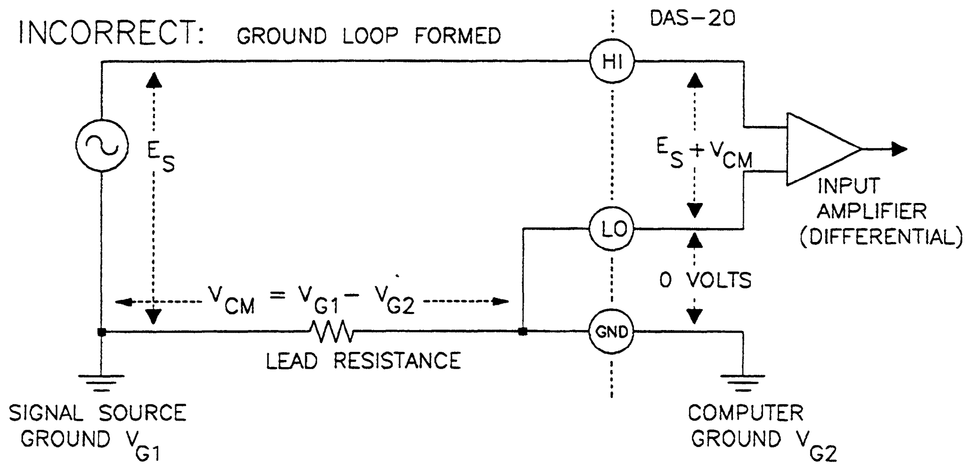

INCORRECT: GROUND LOOP FORMED
DAS-20
HI
LO
GND
E_S
$E_S + V_{CM}$
INPUT AMPLIFIER (DIFFERENTIAL)
0 VOLTS
$V_{CM} = V_{G1} - V_{G2}$
LEAD RESISTANCE
SIGNAL SOURCE GROUND V_{G1}
COMPUTER GROUND V_{G2}

Figure 7-2. Common Mode voltages and differential inputs.

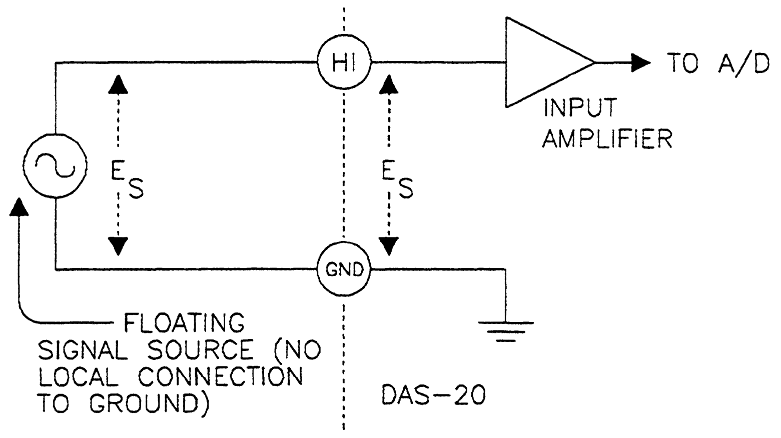


Figure 5-3. Single-ended input connections.

With a combination of floating and ground referred signal sources, the 8-channel differential configuration should be used with the ground referred signals connected as shown in Figure 5-2 and the floating signals connected as shown in Figure 5-4. Note that the jumper between LO and L.L. GND effectively turns the differential input into a single-ended input.

It is important to understand distinctions between the input types, using them effectively, and avoiding ground loops. Misuse of inputs is the commonest difficulty in applying and obtaining the best performance from data acquisition systems.

Measuring Current & 4-20mA Current Loops

DAS-20 interface with process-control current-loop transducers requires the addition of a suitable shunt resistor across the input. Since maximum current is 20mA, the shunt resistor should have a resistance in ohms of 50 * Vfs and should be of low temperature-coefficient metal-film or wire for stability with time and temperature. Using this interface, the 4-20mA working range of the current loop corresponds to 80% of the normal resolution, about 3,277 bits for unipolar ranges and 1,638 bits for bipolar ranges.

Ground-referred currents can also be measured with a suitable shunt. Nonground-referred currents can be measured through an isolation amplifier, a DC, or a Hall-effect current transformer.

Adding More Analog Inputs

MetraByte's EXP-20 allows you to add up to 16 multiplexer channels to any 8/16 analog input. Up to 16 EXP-20's add to one DAS-20 to provide up to 256 channels. The submultiplexer address can be driven by digital outputs DOUT 0-3. A typical scan would use Modes 3, 4, 5 or 6 and then increment the submultiplexer address using Mode 15 and repeat the scan on the next set of sub-multiplex channels, etc. MetraByte's CDAS-2000 adapter cable can connect the first EXP-20 to the DAS-20. Additional EXP-20's can be added with CACC-2000 cables.

When a sub-multiplexer is added, the DAS-20 cannot operate in the fast DMA sampling mode because it must drive the submultiplexer board address through the digital output port. However, it is often possible to partition your channels into high-speed direct inputs and low-speed indirect inputs through the EXP-20. The EXP-20 is best suited to handling high-gain, slow change inputs such as thermocouples, pressure transducers, etc. EXP-20 includes cold junction compensating circuitry, amplification with switch-selectable gain from 1 to 800, and 16 differential inputs.

EXP-20 cards are designed to cascade with flat cable and connectors. All analog channel connections use screw connectors, and each group of 16 channels can operate at a different gain. For more EXP-20 information, please refer to Appendix G and the EXP-20 manual.

Interface For Transducers, Thermocouples, Etc.

Low-level transducers such as thermocouples and strain-gage bridges (load cells, pressure & force transducers) may require amplification before being applied to DAS-20 inputs. The EXP-20 expansion multiplexer incorporates a stable instrumentation amplifier along with circuitry for cold junction compensation of thermocouples. EXP-20 will handle most interfacing requirements to DC output transducers and also includes spaces for filters, shunts and attenuators.

For temperature measurement in the -50 to +125 °C range, use semiconductor temperature transducers such as the Analog Devices AD590/592 (with an output of 273µA at 0 °C and a scaling of 1µA/°C) and the National Semiconductor LM335 (output of 2.73 volts at 0 °C and and a temperature coefficient of 10mV/°C).

For temperatures to 1800 °C or more, use thermocouples. Base metal thermocouples (types J, K, T, and E) have outputs around 40 µV/°C, while the platinum and tungsten types, used for the highest temperature measurements (types S, B, and R), tend to have outputs in the 6-12 µV/°C range. A complication in the use of thermocouples occurs when terminating the thermocouple wire at the copper EXP-20 connections creates an unwanted thermocouple junction that causes an error in the thermocouple output. The error can be offset by sensing the connector temperature with a semiconductor sensor on another channel and correcting the thermocouple readings in software. This is required only at the highest levels of accuracy since in most cases connector temperature (usually room temperature) varies little. The EXP-20 provides the sensing hardware needed to perform this correction. Thermocouple linearization routines for the DAS-20/EXP-20 are included on the DAS-20 disk. These routines have been written in BASICA, but the general program flow can be generalized to use with other languages.

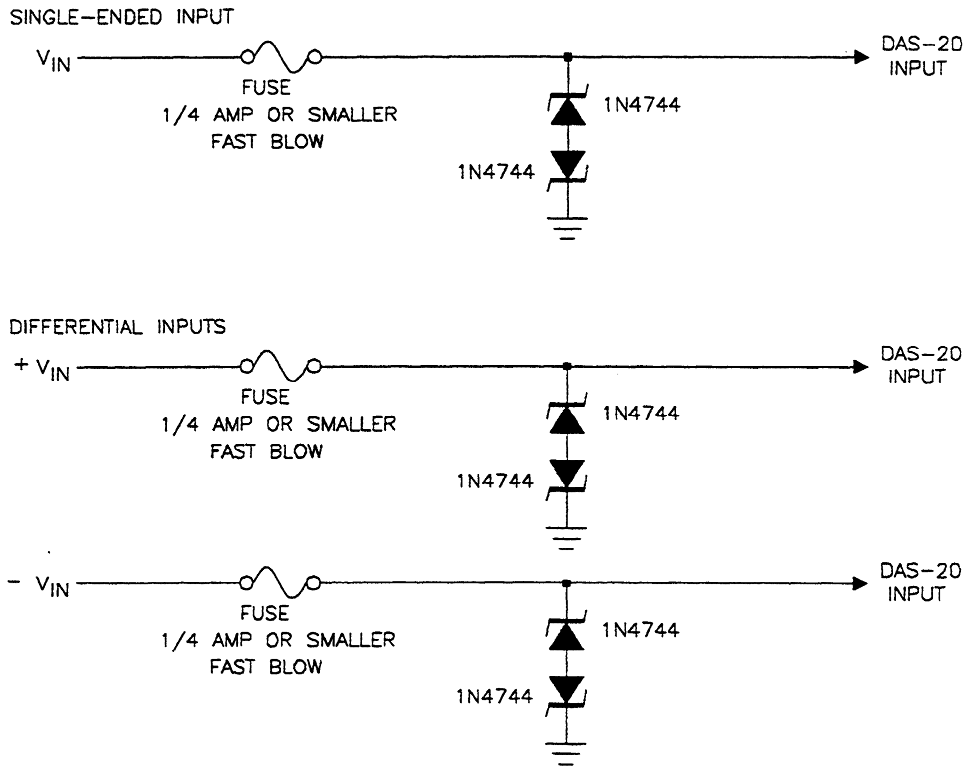
Precautions Against Noise, Groundloops, & Overloads

Unavoidably, data acquisition systems give users access to inputs to the computer. **Do NOT mix these inputs with the AC line.** An inadvertent short can instantly cause extensive and costly damage to 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 by using isolation amplifiers and transformers.
2. Make all connections tight and sound so that signal wires are not likely to come loose and short to high voltages.

The rear connector contains pins for *Digital GND* and *Analog GND*. Digital GND is the noisy or "dirty" ground that carries digital signal and heavy power supply currents. Analog GND is for 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 even when connected to each other.

If there is a high probability that the DAS-20 inputs may inadvertantly come in contact with an input voltage in excess of ±30V, MetraByte recommends the use of input protection circuitry. The following circuits (one for single-ended, the other for differential inputs) will protect the input from most (but not all) overvoltages. These circuits are not recommended for applications with input frequencies in excess of 10kHz.



7.2 DACs (D/A CONVERTERS)

Functional Description

The D/A channels consist of two separate double-buffered, 12-bit D/A converters. Each converter may be set to ±10, ±5, or 0-10 volt full-scale outputs. Writing to the D/A converters can be accomplished in three ways: under program control, under interrupt control, or under DMA control. For further information on programming the D/A converters please refer to Chapter 4.

Since data is 12-bit, it must be written to each D/A in two consecutive bytes. The first byte is the least significant and contains the eight least-significant bits of data (see below or refer to Section 3.2.8 or 3.2.9 for format). The second byte is the most significant and contains the most significant eight bits of data. The least significant byte should be written first and is stored in an intermediate register in the D/A, having no effect on the output. When the most significant byte is written, its data is added to the stored, least-significant data and is presented "broadside" to the D/A Converter thus assuring a single step update. This process is known as double buffering.

Locations of the D/A registers are as follows:

Base Address +8	-	D/A #0 Low Byte
Base Address +9	-	D/A #0 High Byte
Base Address +10	-	D/A #1 Low Byte
Base Address +11	-	D/A #1 High Byte

Data format is:

Lo Byte:	D7	D6	D5	D4	D3	D2	D1	D0
Base +8 or +10	B7	B6	B5	B4	B3	B2	B1	B0
Hi Byte:	D7	D6	D5	D4	D3	D2	D1	D0
Base +9 or +11	x	x	x	x	B11	B10	B9	B8

x = don't care

Output Range Switch Settings

The DAS-20 provides output ranges of ±5V, ±10V, and 0 - 10V. Two toggle switches are used (per channel) to choose the range. The switches are located in the upper-right corner of the board, close to the DAS-20's I/O connector. The D/A range-selection switch locations and settings are as shown in Chapter 2.

PROGRAMMING (without the DAS-20 assembly Driver)

The following example shows how to output data in BASIC. The example translates readily to other languages. Since the D/As have 12-bit resolution, the data (D) should be in the range 0 - 4095 decimal. First split the data into 2 Bytes: DL% (low) and DH%(high).

```
xxx00  DH% = INT(D/256)        'generate High Byte
xxx10  DL% = D - 256*DH%       'derive remainder in Low Byte
```

Next write the data to the D/A. The example assumes D/A #0 with a board base address of Hex 300:

```
xxx30  OUT &H308, DL%     'Low Byte
xxx40  OUT &H309, DH%     'High Byte and load
```

This simple example is written in BASIC, but may be easily generalized into other languages.

A lengthy routine using BASIC OUT's can be avoided by using Modes 7, 8, 9 and 10 of the CALL. Please refer to Chapter 4 and/or example programs EX7.BAS, EX9.BAS, or EX10.BAS for further details on using the Assembly Driver.

* * * * *

CHAPTER 8

A SIMPLE LINEARIZATION ROUTINE FOR THE DAS-20

A problem common to data acquisition is linearizing or compensating for non-linear transducers (thermocouples, flowmeters, tachogenerators etc). With a powerful computer behind the data acquisition system, it is simple to perform the polynomial approximation that serves as a linearizing function.

The starting point for any linearizing algorithm is a knowledge of the calibration curve (or input/output behavior) of the transducer. This may be derived experimentally or be available in manufacturer's data or standard tables e.g. NBS tables for thermocouples. Of the approaches to linearizing, two of the most common are (1) piecewise linearization using look up tables and (2) use of a mathematical function to approximate the non-linearity. Amongst the mathematical methods, polynomial expansion is one of the easiest to implement. The utility program POLYNOM.BAS allows you to generate a fifth-order polynomial approximation which is usually very adequate for linearizing most practical transducers. The transducer signal is digitized by the A/D and made the input (X) to the linearizing polynomial function:

```
Y= f(X) = C1 + C2*X + C3*X^2 + C4*X^3 + C5*X^4 + C6*X^5
```

Before you start the program, have in hand a table with your desired output/input data (or calibration data). To run the program from DOS, type BASICA POLYNOM and answer the first prompt which will ask you how many calibration points you intend to enter. The program will perform a least-squares fit regression analysis to ensure that the approximation is as close as possible over the whole range of input data. A question arises as to how many data points you have to enter to get a good approximation. This is something of a trial and error process: too few points, say less than 6, may be insufficient for an accurate approximation whereas too many, say more than 40, is just tedious to enter and will not significantly improve the accuracy.

When you have provided the number of data points, the program will then request you to enter the data as Y, X pairs (or output vs. input) points. The two values for each data point should be entered with a comma as a delimiter between the values and you may edit with the backspace before you press return. The line on the bottom of the screen will tell you where you are up to. As soon as you have entered the final pair, you will go into a review and edit mode that allows you to change any values that you may have misentered. When all is O.K., type 0 in response to the prompt. This is the point of no return and will either present more data for review or lead into the regression analysis. You will then be requested for the order of the polynomial that you wish to use. Five is usually the best choice although flexibility is provided in case you are interested in examining the effects of a lower order (Note: There is not really much advantage in using an order lower than five from the point of view of subsequent execution time for the polynomial evaluation).

The regression analysis terminates by displaying the coefficients, COEF(1 thru 6), which provide the best least-squares approximation to your data. Just how good this approximation is, can be tested by entering any value you like in the TEST CONFORMANCE section and seeing what output results. When you tire of this, type Q (for quit) and you will then be asked if you want to try the regression analysis again with a different order (this just saves re-entering your data if you do). Type Y (yes) or

N (no) as required. If you intend to save the values of the coefficients either copy them down now or if you have a printer, type PrtSc. before they are scrolled off into oblivion. This is strictly a utility program and does not save anything for you. As with any BASIC program, you can abort execution at any point by typing Ctrl-Break.

When you want to include a linearization routine in your own programs and evaluate the polynomial, you can do it in a neat little subroutine (this one is in BASIC) as follows:

```
xxx00    COEF(1) = Value
xxx10    COEF(2) = Value
xxx20    COEF(3) = Value
xxx30    COEF(4) = Value
xxx40    COEF(5) = Value
xxx50    COEF(6) = Value
xxx60    Y = COEF(1)
xxx70    FOR CNT% = 5 TO 1 STEP -1
xxx80    Y = Y + COEF(CNT%+1) * X ^ CNT%
xxx90    NEXT CNT%
xx100    RETURN
```

* * * * *

CHAPTER 9

CALIBRATION & TEST

To maintain full accuracy of the DAS-20, you are urged to perform periodic recalibration at an interval based on the nature of the board's service and environment. For an environment with frequent large changes of temperature and/or vibration, a 3 month recalibration interval is recommended. For laboratory or office conditions, 6 months to 1 year is acceptable.

A 4.5-digit digital multimeter and a voltage calibrator or a stable noise free D.C. voltage source are the minimum requirements for performing a satisfactory calibration. In addition, a card extender, such as Vector part # 3690-22 (available from Vector dealers or Vector Electronic Co. Inc., Box 4336, Sylmar, Ca. 91342; Phone: 213/365-9661) is recommended. The card extender is an inexpensive device that greatly improves accessibility to the board during calibration, and it will probably be useful with other boards too.

Calibration is easily performed using the CAL20.EXE program in conjunction with a BASICA program. To activate CAL20.EXE, type **BASICA CAL20** at the DOS prompt. This program will lead you through the calibration and set up procedures with a variety of prompts and graphic displays showing you settings and adjustment trimpots. The explanatory material in this section is brief and is intended as an aid for the calibration programs.

The calibration program also serves as a useful test of the DAS-20 A/D & D/A functions and can aid in troubleshooting if problems arise. Simple tests are included for the digital I/O and programmable timer.

* * * * *

CHAPTER 10

INSTRUCTIONS FOR PRODUCT RETURN

10.1 ATTEMPTING YOUR OWN REPAIRS

Most of the components that interface to outside connections and are thus vulnerable to external shorts, transients, overloads, etc. are in sockets. These parts are readily serviced simply by unplugging and replacing them with a new part. MetraByte can service your board, but if you wish to attempt your own repairs, use the following as a guide to locating and fixing faults:

Board located at wrong I/O address:	Error code -3 received on initialization usually indicates that the BASE ADDRESS switch setting does not correspond to the address used in your program. Run INSTALL, check that the DAS20.ADR file is correct. If there is no discrepancy, the board has a serious hardware fault and should be returned to MetraByte for repair.
A/D Converter nonoperational, nonlinear, missing bits, excessive noise:	All channels dead, error code -1 in mode 3. First check that other functions on the board are working e.g. digital I/O, control register etc. If other functions are OK, replace the HI-774AJD.
Missing channels opens & shorts:	Suspect multiplexer chips. These are Harris HI-508A-5.
Gain wrong, drift, excessive noise, unable to zero:	These effects may be due to a faulty input instrumentation amplifier. Replace U5 and/or U13, HA5170 Operational Amplifiers.
D/A converters: missing bits, non-linearity:	D/A #0, replace D/A, U7 DAC-811 D/A #1, replace D/A, U8, DAC-811.
Digital outputs:	Faulty or blown outputs. Replace U20, part # 74LS794.
Digital inputs:	Faulty or blown inputs. Replace U26, part # 74LS373.
Counter/Timer:	Replace U6, AMD-9513.

10.2 USING FACTORY REPAIR

Before returning any equipment for repair, please call 508/880-3000 to notify MetraByte's technical service personnel. If possible, a technical representative will diagnose and resolve your problem by telephone. If a telephone resolution is not possible, the technical representative will issue you a Return Material Authorization (RMA) number and ask you to return the equipment. Please reference the RMA number in any documentation regarding the equipment and on the outside of the shipping container.

Note that if you are submitting your equipment for repair under warranty, you must furnish the invoice number and date of purchase.

When returning equipment for repair, please include the following information:

1. Your name, address, and telephone number.
2. The invoice number and date of equipment purchase.
3. A description of the problem or its symptoms.

Repackage the equipment. Handle it with ground protection; use its original anti-static wrapping, if possible.

Ship the equipment to

Repair Department
Keithley MetraByte Corporation
440 Myles Standish Boulevard
Taunton, Massachusetts 02780

Telephone 508/880-3000
Telex 503989
FAX 508/880-0179

Be sure to reference the RMA number on the outside of the package!

* * * * *

APPENDIX **A**

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. Sixteen bits of data is equivalent to values from 0 t0 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	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**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 is D7; 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 8253 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 complement 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:

```
50,000  (Hex C350)  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
```

* * * * *

APPENDIX **B**

UNDERSTANDING DMA

B.1 WHAT IS DMA?

Consider the problem of moving a large amount of data to or from an I/O device. This requirement is commonly encountered in the operation of many peripheral devices (for example disk drives) that are constantly moving large amounts of data into and out of memory. An obvious way of making the transfer would be a short program which for an input operation might read a byte from the I/O port into the accumulator (AX register) of the 8088 processor and then move the data from the AX register to a memory location to store it. In addition, we have to keep track of the memory locations where the data is going. By far the simplest way of handling this is to lay the data down in contiguous locations within a block of memory, using one of the processor's index registers to control the address. Each time a byte is transferred, the index register (usually the DI or Destination Index register) is incremented or decremented to point to the next location. A typical example assembly language program to do this might be as follows:

SETUP:

```
MOV AX,SEGMENT              ;setup segment of memory for transfers
MOV DS,AX
MOV DI,OFFSET;setup start address within segment
MOV CX,COUNT ;setup number of bytes to transfer
MOV DX,IOPORT;DX = I/O port address
```

READ:

```
IN AL,DX  ;read byte from I/O port     (8)
MOV [DI],AL  ;store data                   (10)
INC DI ;increment index              (2)
LOOP READ ;continue reading until CX=0 (17)
```

CONT:

```
........ ;yes, continue with program
```

The opposite operation of transferring data from memory to an I/O port is essentially similar. Numbers in parentheses following *READ:* are the processor clock cycles required to execute each instruction. On a standard IBM PC, the clock is 4.77MHz corresponding to a clock cycle period of 210 ns, so that the loop takes 8 + 10 + 2 + 17 = 37 cycles or 7.8 µs to execute. This would be the fastest we could transfer each byte. Note also the following:

1. The processor is tied up 100% of the time in transferring data, it cannot execute any other part of the program while the transfer is underway.
2. The rate at which the data is transferred is controlled by the processor clock and may not correspond to the rate at which the I/O device wants to handle the data. This can be circumvented by polling the I/O device to see if it is ready, or having the I/O device generate a hardware interrupt, but either of these adds further code to the routine which will slow down transfer rates even further. In practice because of all the stacking of registers that occurs when a

hardware interrupt is processed, it is impossible to handle data transfer rates much above 5-10kHz using interrupts.

3. If the processor has to handle an interrupt from one device while it is involved in handling a data transfer to another device then the delays involved may cause it to miss data, or at least will cause a discontinuity in the data flow.

It would be nice to have a way of transferring data without involving the processor so it can be freed up as much as possible to attend to execution of the program. It would also be a plus if we could speed the transfer rate up compared with the example programmed transfer above, and be able to control the rate easily. Since all we want to do is move a byte directly to/from an I/O port from/to memory without an sort of processing on the way, we are better off not using the processor at all, but instead providing special hardware that will acomplish this commonly required task. The process of connecting an I/O device directly to memory is known as DMA (Direct Memory Access) and the hardware that controls this process is known as the DMA controller, which in the case of the IBM PC is the function of the 8237 chip on the system board.

B.2 THE MECHANICS OF A DMA TRANSFER

What happens when a device wants to transfer data to/from memory? The first step is for the device to send a signal known as a DMA REQUEST (DREQ for short) to the DMA controller. The processor normally has control of the computer's address and data busses as well as control signals such as the memory read/write (MEMR & MEWW) and I/O read/write (IOR & IOW) lines. To accomplish a DMA transfer, control of these lines must be passed to the DMA controller. On receipt of the DREQ, the DMA controller in turn issues a HOLD REQUEST to the processor. As soon as it is able to, when it has partially completed the instruction it is currently executing, the processor issues a HOLD ACKNOWLEDGE signal to the DMA controller, and simultaneously disconnects itself from the address, data and control busses. This process is technically known as "tri-stating" as the connections to the processor assume a third open circuit state compared to their usual binary states of 1's and 0's or highs and lows.

On receipt of the HOLD ACKNOWLEDGE, the DMA controller goes to work. It releases its own connections to the address and control busses from their tristate condition, asserting a valid memory address from an internal counter and then issues a DMA ACKNOWLEDGE (DACK) signal to the I/O device followed by a simultaneous IOW and MEMR for a data output, or IOR and MEMR for input. The peripheral in turn responds to the DACK and IOR or IOW signals by placing or receiving data on the data buss effecting a transfer directly to/from memory. On completion of the MEMR/IOW or MEMW/IOR from the DMA controller, the controller removes DACK, releases HOLD REQUEST, tristates its own address and control lines, and increments or decrements its internal address counter ready for the next transfer. The processor in turn regains control of the busses, continuing execution of the next instruction. From the assertion of DREQ to completion of the cycle takes about 2-5 microseconds, depending on the length of the instruction that the processor happens to be engaged in on receipt of the DREQ. The actual amount of time between instructions that the processor loses the buss to the DMA controller is even less, about 1 microsecond. The effect on program execution is minimal even when transferring data at very high rates which can approach 350,000 bytes/sec on the IBM PC. To prevent the DMA controller from "hoggingó the busses if the DREQ is held constantly high, the controller always allows the processor to perform at least part of an instruction between each DMA transfer, so that even operating "flat-out", DMA cannot grab much more than 30% of the buss bandwidth.

In order to perform DMA operations the peripheral must include hardware that generates the DREQ and responds to the DACK. The DAS-20 includes this special hardware. The DMA controller on the other hand is a system component that is a standard feature of the IBM PC architecture. The only member of the IBM PC family that does not include an 8237 DMA controller is the PCjr which also lacks expansion slots. Most compatibles also include the DMA controller with the exception of the TI Professional. Without an 8237 controller, the DAS-20 is reduced to operation as a standard I/O interface.

It is important to appreciate that the DMA controller sets the dynamics of the DMA transfer, there is nothing in the peripheral I/O device that can alter the maximum speed at which the controller will handle data. There are some surprising side effects of this, in particular the IBM PC/AT which is generally 3 times faster than a standard PC or PC/XT is actually slower on DMA transfers because its controller operates at 3MHz instead of the 4.77MHz on the PC. On the other hand it can also perform word (16 bit) transfers on its extended data buss as well as byte (8 bit) transfers on the PC compatible section of its data buss which with the right hardware can make up for the slower transfer rate. Although it will operate in DMA mode in the PC/AT, the DAS-20 performs word transfers by making two sequential byte transfers and so is unable to take advantage of the PC/AT's extended buss, this being the price paid for PC compatibilty.

Note that this description of DMA is fairly specific to the PC and Intel 8086/88/286 family. On other processors, it is not uncommon for the DMA to be interleaved with instructions so that DMA transfers take place duriod periods of instruction execution when the processor is not using memory. This makes DMA even more transparent, the end effect is however the same.

B.3 DMA LEVELS

Although we have discussed the operation of a single device using DMA, it is customary to cater to the needs of several devices by providing several DMA channels, each one dedicated to a particular device. The 8237 provides four separate DMA channels known as Levels 0 thrã 3. Correspondingly, there are 4 DMA request lines, DREQ0-3, and 4 corresponding acknowlege lines, DACK0 - 3. The lines are prioritized according to a two possible protocols set by a bit in the controller command register, either fixed priority where lower DMA levels have higher priority than higher levels, or rotating priority where each level takes a turn at having the highest priority. The PC BIO– sets the 8237 to operate in fixed priority mode on power up, and it is inadvisable to change this for reasons that will be explained later. The PC/AT design added to the number of DMA channels by using two 8237 DMA controllers. Since one channel is used to cascade one controller into the other, this actually adds an additional 3 channels all of which appear on the 16 bit additional expansion connectors of the PC/AT and are dedicated to 16 bit transfers.

Apart from its uses for high speed data transfer, the DMA controller includes counter hardware that cycles through the memory addresses, so as a byproduct of its design, it can also be used to refresh dynamic memory saving the cost of a separate memory refresh controller. This is what IBM chose to do in the PC, and on all models Level 0 performs this function with its DREQ being driven from counter 1 of the internal 8253 timer at a 15 microsecond interval. The complete assignments for the levels are as follows:

For all machines (except PCjr) these levels are capable of 8 bit transfers:

Level 0:	Memory refresh.
Level 1:	Not assigned and usually available. Certain local area network interfaces may use this level.
Level 2:	Used by the floppy disk controller and not free for any other purpose.
Level 3:	May be used by the hard disk controller on some PC/XT models. On floppy disk only, PC/AT and some PC/XT machines, this level is free for other uses.

For the PC/AT only, these levels are capable of 16 bit transfers:

Level 4:	Used for cascading.
Levels 5-7:	Available on AT special connectors.

Although the extended DMA capabilities of the PC/AT have been mentioned here for the sake of completness, for compatability reasons the DAS-20 has not been designed to utilize these extra levels.

In practice the only levels that a user is able to use are either Level 1 or 3 with the latter choice possibly eliminated in the case of a hard disk equipped PC/XT. The DAS-20 can select either of these possible levels through software control. In certain circumstances, e.g.level 1 being used by a network interface and level 3 by the hard disk, it may be impossible to operate the DAS-20 unless the network software is capable of sharing a DMA level in a sequential fashion. The DAS-20 does include tri-state drivers which can be sequentially enabled and disabled to share one DMA level amongst many DAS-20's or other devices provided their software and hardware has equal capabilities.

If there is any doubt about whether level 3 is available in your particular machine, there are a couple of ways of finding out. Some hard disk controller boards use hardware DMA to transfer data to DOS's disk buffers and others use block moves (software) which is just as fast. The transfer method is controlled by the fixed disk BIOS which is located on a peripheral ROM on the fixed disk controller board. Although the fixed disk BIOS calls are of necessity functionally identical, the type of transfer method is controlled by the manufacturer of the controller board. The best method of checking whether the fixed disk makes use of level 3 is to enter DEBUG and load the 8237 DMA controller level 3 transfer address and word count
registers at I/O locations 6 & 7 with some garbage numbers, exit DEBUG, run CHKDSK on your fixed disk, re-enter DEBUG and read the transfer address and word count registers to see if they have changed. If they have, level 3 is used by the hard disk, otherwise it is not. Another method is to connect a scope to the DREQ3 bus line and look for activity when yoã do a CHKDSK. If pulses are apparent, level 3 is in use.

B.4 THE DMA CONTROLLER

This section provides a more detailed description of the Intel 8237 DMA controller used on the PC. A full description is contained on the 8237 component data sheet available from Intel [Intel Corp., Lit. Dept., 3065 Bowers Avenue, Santa Clara, CA 95051].

Apart from the command register, the 8237 includes several other registers using a total of 16 I/O addresses, with the base I/O location in the IBM PC and compatibles starting at zero. A description of the registers follows:

BIOS

BIOS initializes the command register with byte 00H (zero). Apart from many of the bits controlling signal polarity and timing features that must be observed for correct operation of the internal hardware, Level 0 is used for memory refresh and Levels 2 & 3 may be used by the disk drives. All these functions are critical, so it is inadvisable to alter BIOS's initialization of the command register byte after power up.

The key features initialized by the BIOS are:

1. Memory to memory transfers using Levels 0 & 1 are not possible. (Level 0 is used for memory refresh).
2. Normal timing, late write and fixed priority are selected.
3. The DREQ's are active high, and the DACK's are inverse, active low.
4. In the case of the PC/AT which has 2 DMA controllers, both are initialized as above.

COMMAND REGISTER - I/O Location 8

This is an 8-bit Write only register that controls overall operation of the 8237 as follows:-

```
D7   D6   D5   D4   D3   D2   D1   D0
 I    I    I    I    I    I    I    I
 I    I    I    I    I    I    I    I    0 - Memory/memory disable
 I    I    I    I    I    I    I    I__  1 - Memory/memory enable
 I    I    I    I    I    I    I
 I    I    I    I    I    I    I         0 - Level 0 address hold disable
 I    I    I    I    I    I    I_____    1 - Level 0 address hold enable
 I    I    I    I    I    I
 I    I    I    I    I    I              0 - Controller enable
 I    I    I    I    I    I_________1 - Controller disable
 I    I    I    I    I
 I    I    I    I    I                   0 - Normal timing
 I    I    I    I    I____________  1 - Compressed timing
 I    I    I    I
 I    I    I    I                        0 - Fixed priority
 I    I    I    I________________   1 - Rotating priority
 I    I    I
 I    I    I                             0 - Late write selection
 I    I    I____________________    1 - Extended write selection
 I    I
 I    I                                  0 - DREQ active high
 I    I________________________    1 - DREQ active low
 I
 I                                       0 - DACK active low
 I____________________________     1 - DACK active high
```

MODE REGISTER - Location B Hex

This is a group of 4 write only 6 bit registers, each one controlling the characteristics of each DMA channel and selected by the lower 2 bits of the byte. The bit functions are as follows:

```
D7   D6   D5   D4   D3   D2   D1   D0
|         |    |    |         |
|         |    |    |         |         |00 - Channel 0 select
|         |    |    |         |_______|01 - Channel 1 select
|         |    |    |                   |10 - Channel 2 select
|         |    |    |                   |11 - Channel 3 select
|         |    |    |
|         |    |    |                   |00 - Verify transfer
|         |    |    |______________|01 - Write transfer
|         |    |                        |10 - Read transfer
|         |    |
|         |    |_________________|0  - Autoinitialize disable
|         |                             |1  - Autoinitialize enable
|         |
|         |____________________|0  - Address increment select
|                                       |1  - Address decrement select
|
|                                       |00 - Demand mode select
|_____________________________|01 - Single mode select
                                        |10 - Block mode select
                                        |11 - Cascade mode select
```

A mode register is associated with each channel and unlike the command register, can safely be programmed by the user at least for the available levels 1 or 3. Some explanation of the bit functions follows:

D0 & D1	The 2 lowest bits select the mode register for the channels being programmed. Make sure these are correct so that you do not interfere with memory refresh etc.
D2 & D3	Control the direction of transfer. A read transfer moves data from memory to an I/O device, whereas a write transfer performs the reverse. A verify transfer is a pseudo transfer made without read or write pulses and is used in certain types of system testing, otherwise it is of no value.
D4	Controls auto-initialization (see below).
D5	The address counter in the 8237 can count up or down from its starting value. This bit controls the count direction.
D6 & D7	Control the type of DMA transfer (see below). 00 = Demand mode select. 01 = Single mode select. 10 = Block mode select. 11 = Cascade mode select.

If bit D4 is zero, a DMA transfer will start at the memory address specified in the 8237 address register and proceed for the number of bytes specified in the word count register. On reaching the specified word count, further DREQ's are masked out terminating the DMA transfer. If D4 is one, auto-initialize mode is enabled. On reaching the final word count, the address counter is re-initialized with its starting address so that in this mode, DMA proceeds continuously to a memory buffer area of location specified by the starting address, and length specified by the word count. On reaching the end of the buffer, transfers continue at the beginning of the buffer. In effect the buffer is circular in auto-initialize mode. This can be useful when outputting data of a cyclic nature such as driving a D/A converter to generate a periodic waveform.

D6 and D7 control the DMA transfer modes. BIOS initializes D6 and D7 on every channel to select the single transfer mode. In this mode, each DREQ will result in a single byte transfer of data followed by return of bus control to the main processor. The DAS-20 has been designed for operation of the DMA controller in this mode. The block and demand modes are almost similar to each other. In block mode, a single DREQ will make the DMA controller perform multiple transfers one after another until the final word count is reached. Once the DMA process becomes active, the DREQ can be taken low without affecting continuation of the DMA block transfer i.e. the DREQ is ignored once transfers start. Demand mode is similar except that the transfers can be halted at any time during the block transfer by taking DREQ low. Transfers will continue from the point where they stopped on taking DREQ high. Both of these transfer modes can block the processor's use of the bus and hold off service of other DMA channels while they are active. Since Level 0 is used for memory refresh, any hold off of service on this level may result in data loss and a system crash. For this reason, it is inadvisable to select either block or demand transfer modes and the DAS-20 has not been designed to support them. Cascade mode allows allows connection of another 8237 hold request and hold acknowledge signals to the DREQ and DACK of the first DMA controller. This allows expansion of the number of DMA channels as in the PC/AT but apart from its use in hardware expansion has no other function.

MASK REGISTER - I/O locations A Hex & F Hex.

Each channel has associated with it a mask bit which can be set to disable the incoming DREQ. A mask bit is also set when not in the autoinitialize mode by the terminal count, T/C, produced at the end of the DMA transfer.

The mask register including all the mask bits for all the channels can be written to as a whole through the port at I/O location F hex. Apart from initializing the DMA controller which the BIOS performs on power up, it is best to avoid this method of setting and resetting mask bits, since yoã may disturb the mask bit on another channel inadvertently as the mask register is write only and it is impossible to read its current contents.

The I/O location at address A hex provides a way of setting and clearing an individual mask register bit without disturbing the mask bits of the other DMA levels. The protocol for doing this is as follows:

```
D7   D6   D5   D4   D3   D2   D1   D0
                         |    |
     Don't care      |   |          | 00 - Select level 0 mask bit
                         |    |_____| 01 - Select level 1 mask bit
                         |               | 10 - Select level 2 mask bit
                         |               | 11 - Select level 3 mask bit
                         |
                         |__________| 0 - Clear mask bit (enable DMA)
                                     | 1 - Set mask bit (disable DMA)
```

Before performing any setup of the DMA controller, the mask bit of the level being set should be set to disable any spurious DMA transfer that might otherwise occur.

ADDRESS REGISTERS - Location (2 x Level #)

There are four sixteen bit address registers, one associated with each DMA level. The address register should be loaded with the starting address of the memory area reserved for DMA Since these are 16 bit registers, they are loaded by 2 sequential outputs to the same I/O port:

For example, for Level 3:

```
                    OUT 6, Low byte of address
                    OUT 6, High byte of address
```

The address read back from the same location is the current address i.e. the starting address plus or minus the number of DMA transfers depending on whether increment or decrement is selected by the mode register.

BYTE COUNT REGISTERS - Location (2 x Level #) + 1

There are four sixteen bit byte count registers (called word count on the Intel data sheet) that control the number of DMA transfers. These registers are loaded with the number of bytes required in a transfer minus one. The registers decrement to zero and on reaching FFFFH generate a terminal count, T/C, on the expansion bus. In the non-autoinitialize mode further DMA transfers cease, whereas in the autoinitialize mode both the byte count and address registers are automatically reloaded with their initial values and DMA proceeds continuously. The combination of T/C (terminal count) and the selected DACK can provide an interrupt to signal the end of thee DMA transfer. The DAS-20 can generate this type of interrupt on on any level if required. A second method of determining if the DMA transfer has ended is to poll (read) the byte count register until it has reached FFFFH. This also provides data on the number of DMA transfers that have taken place and is the method used in mode 12 of the DAS-20.BIN driver.

B.5 THE DMA PAGE REGISTERS

Since the 8237 DMA controller can only supply a 16 bit address, and 20 bits are used to address the 1 Mbyte of RAM and ROM memory in the PC architecture, it is only possible to perform DMA within a 16 bit addressable area of memory at a time. This amount to 64 Kbytes and is known as a page. There are 16 pages in the whole 1 Mbyte address space.

When performing DMA, selection of the active page is made through a set of three 4 bit page registers. These are write only registers and cannot be read back. The page register locations and functions are as follows:

I/O LOCATION	FUNCTION
83 Hex	Level 0 & level 1 page register
81 Hex	Level 2 page register
82 Hex	Level 3 page register

Note that level 0 and level 1 share the same page register. Since the memory refresh performed by level 0 only requires cycling of the lower order address bits A0 thru A6 or A7, the selected page (A16-A19) is irrelevant for refresh. On early IBM PC's (the type with 64K of memory on the mother board using 16K dynamic RAM chips) writing anything other than zero to the level 1 page register would cause a memory parity error and hang up the computer. This was due to a design defect that was corrected in later versions. The problem can be circumvented on these older machines by disabling the NMI (non-maskable interrupt) which is used to detect memory parity errors before writing to the level 1 page register. To disable the NMI perform:

```
OUT &HA0, &H00
```

To re-enable the NMI after writing 0 to the level 0/1 page register:

```
OUT &HA0, &H80
```

B.6 SETTING UP A DMA TRANSFER

Setup of a DMA transfer entails several steps which must be performed in the correct sequence and requires a good understanding of the action of the hardware. Modes 6, 10 and 27 of the DAS-20 driver performs this setup and avoids a lot of programming, the source listing of mode 1 (see DAS-20.ASM) gives an example of the controller setup. If you do not want to use the driver and wish to write your own setup program you should proceed in the following sequence:

1. Disable any active DMA on the chosen level by setting the appropriate mask register bit.
2. Write to the appropriate mode register of the 8237 DMA controller to set up the transfer characteristics.
3. Load the address and byte count registers with appropriate data.
4. Write the page register to set the DMA page.
5. Install any interrupt handler.
6. Write the DAS-20 DMA and interrupt control registers to enable DMA and possibly interrupts.
7. If you are using the on board timer, it is a good idea to reload it so that the first DMA transfer will occur at a fixed delay from this point.
8. Enable interrupt mask register (if using interrupt).
9. Last step, enable DMA mask register.

The main pitfall to avoid when doing the DMA setup is enabling the 8237 DMA controller before DMA is enabled on the DAS-20 through bit 5 of the A.D.C. control register. Until the DAS-20 is enabled, the DREQ line is tri-state or high impedance and picks up a lot of crosstalk from other lines on the expansion bus. If the 8237 mask register bit is enabled, many spurious rapid DMA transfers will be generated, and it is not unusual to find that the 8237 signals that transfers are ended before the DAS-20 is later enabledí Always enable the DAS-20 first followed by enabling the 8237 controller.

B.7 DMA TRANSFER TIMING

On receipt of a DMA request (DREQ) from a peripheral, the 8237 DMA controller immediately sends a HOLD request to the 8088 processor. The 8288 bus interface unit responds to the HOLD as soon as possible according to the following rules (times apply to an 8088 with 4.77MHz clock):

1. The response is close to immediate if no bus cycle (transfer to or from memory or I/O) is in progress. This time can be as little as 0.32 microseconds.
2. If a bus cycle is in progress, it will be completed before hold acknowledge is enabled. In this case a delay of up to 1.74 microseconds may take place assuming memory generates no wait states as is normal on the IBM PC.
3. If a bus cycle involving a word (2 byte) transfer to an odd byte memory boundary is taking place, both bus cycles will be completed before hold acknowledge is enabled. In this case the delay can extend to 2.58 microseconds.
4. If the hold request occurs at the beginning of an interrupt acknowlege sequence then hold acknowledge can be delayed up to 3.01 microseconds.
5. If an instruction is executed with a LOCK prefix, the bus interface unit will insure that the whole following instruction will be executed before issuing a hold acknowledge. In the case of an IDIV (integer division) instruction this could be as long as 40 microseconds.

In general condition 5 is unlikely to arise, since the LOCK prefix is used by programmers to insure total instruction execution in multi-processor systems and the IBM PC is a single processor system. There is therefore no reason for a programmer to use the LOCK prefix in programs for the PC, though of course there is nothing to prevent its use.

Once the DMA transfer is under way, the DMA controller will take 5 clock cycles or 1.05 microseconds to effect the transfer. Assuming that instructions are not LOCKed, the worst case transfer timing would appear to be condition 4 plus the DMA cycle time, about 4.06 microseconds. This neglects the effect of higher level DMA channels of which the level 0 memory refresh occurring every 15 microseconds is the main culprit. If a memory refresh bumps the DMA transfer on level 1 or 3, it can add a further 0.84 microseconds (refresh DMA's are slightly shorter than I/O DMA's). All told the worst case latency after the DREQ can amount to 4.9 microseconds for a single byte transfer.

There are a number of conclusions that can be drawnabout DMA latency. If you are performing clocked DMA transfers i.e. regulated by the timer, it will limit transfer speeds to the worst case delay time corresponding to about 200,000 bytes/sec or 120,000 words/sec. However with less stringent timing requirements you can expect speeds of up to 350,000 bytes/sec or 200,000 words/sec. These figures apply to the standard IBM PC with a 4.77MHz clock, higher clock rates will reduce these times.

B.8 USING INTERRUPTS WITH DMA

The DAS-20 can generate a hardware interrupt on any level from 2 thru 7 from one of the following four sources:

1. An A/D converter End of Conversion pulse.
2. A periodic interrupt from the timer # 2.
3. A terminal count (T/C) from the DMA controller.

4. An End of Queue pulse.

This provides several options in programming. The DMA terminal count interrupt can be usedd to signal the end of a DMA transfer or set up another transfer or move data from the DMA buffer to disk etc. The external interrupt can be generated by an external device that requires DMA service, or to signal that it has transferred all its data etc. The periodic interrupt can be used to perform some repetitive operation like sending a block of data to a peripheral.

B.9 DETERMINING STATUS OF A DMA TRANSFER

Once a DMA transfer is under way, it is often necessary to find out how many transfers have taken place or whether the DMA transfer has completed. This can be determined by reading the appropriate byte count register of the 8237 DMA controller e.g.:

```
10 XL% = INP(7)           'low byte for level 3
20 XH% = INP(7)           'high byte for level 3
30 B = 256 * XH% + XL%    'B = number of transfers left to complete
```

Alternatively, mode 12 of the DAS20.BIN driver will perform a similar function, returning the number of bytes or words transferred according to the setup of other DMA controlled modes.

* * * * *

APPENDIX **C**

THE EXP-20

CONTENTS

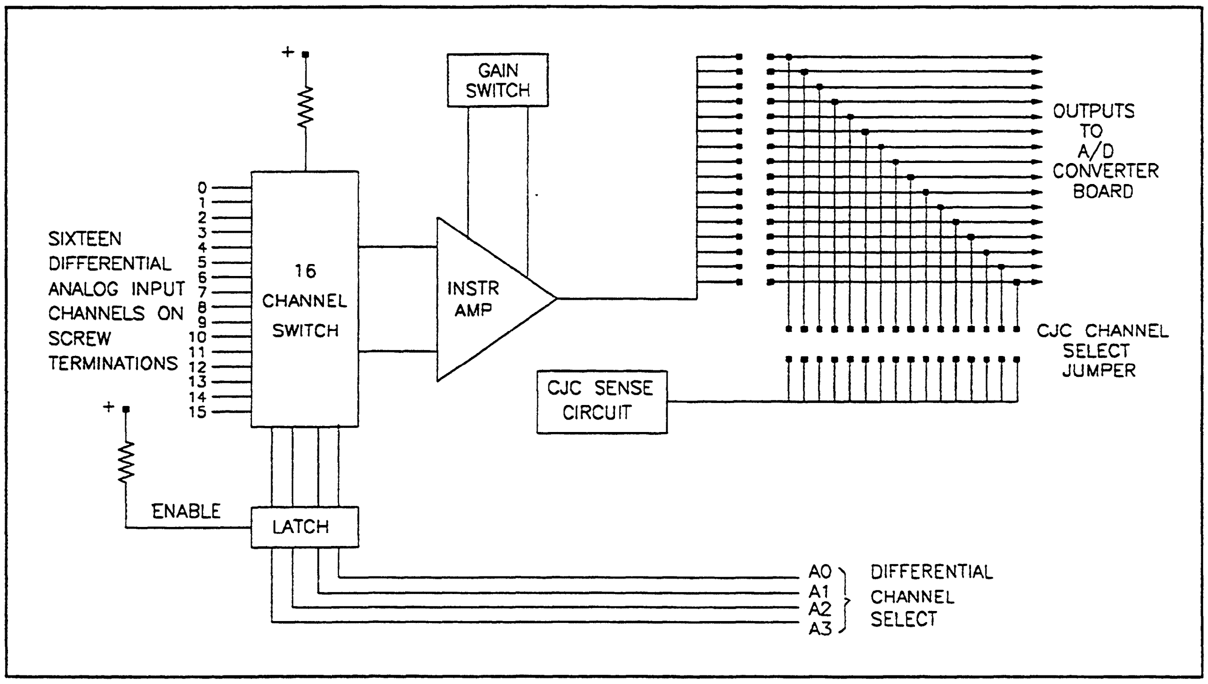
* * * * *

Section 1
INTRODUCTION

1.1 GENERAL

The EXP-20 is a high speed, analog signal multiplexer and instrumentation amplifier designed for use with MetraByte's PC-bus compatible DAS-20 analog input module. It is configured as 16 differential, Bipolar input channels with nine DIP switch selectable gains of 1, 10, 100, 200, 300, 500, 600, 700, and 800 covering full-scale voltage spans from ±12.5mV through ±10 VDC. The EXP-20 also supports all common thermocouples such as J, K, T, R, S, B, and E with built-in Cold Junction Compensation. Output from the EXP-20 may be any one of 16 single-ended or one of eight differential channels. Simple, potentiometer adjustments for all calibration and other parameters is standard. Cold junction compensation may be 20 mV/° C, or 200 mV/° C. The EXP-20 is a passive amplifier/active multiplexer design with a minimum MUX settling time of 2us (no filter).

Two 50-pin headers permit daisy-chain expansion to a maximum of 16 EXP-20s (256 analog/thermocouple signals). EXP-20 power is supplied by the computer.


GAIN SWITCH
SIXTEEN DIFFERENTIAL ANALOG INPUT CHANNELS ON SCREW TERMINATIONS
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16 CHANNEL SWITCH
INSTR AMP
OUTPUTS TO A/D CONVERTER BOARD
CJC CHANNEL SELECT JUMPER
CJC SENSE CIRCUIT
ENABLE
LATCH
A0
A1
A2
A3
DIFFERENTIAL CHANNEL SELECT

1.2 FEATURES

- 16 differential, bipolar inputs
- Nine DIP switch selectable gains
- On-board DC to DC converter
- Choice of Single-ended or Differential outputs
- Built-in Cold Junction Compensation
- Supports J, K, T, R, S, B, and E thermocouples
- Pin compatible with DAS-20
- Cascadable to 256 Analog channels

Section 2

HARDWARE

2.1 GAIN SELECTION

The EXP-20 supports nine standard gains, which are selectable via Switch S1. Since gain selection affects the entire board, it should be performed only after determining the largest full-scale input range. Gain selection is often a compromise between full-scale range for the largest signal and resolution for the smallest signal. For example, different thermocouple types may be combined on the same EXP-20 if the maximum signal from the "larger" thermocouple is not expected to exceed the upper voltage limit for the gain chosen. Use the NBS "mV output vs temperature" curves to determine the maximum expected voltage. Switch settings and resultant gains are as shown.

SWITCH 1				GAIN	BIPOLAR RANGE	SETTLING TIME	THERMOCOUPLE TYPE
1	2	3	4				
0	0	0	0	X1	±10V	4uS	
1	0	0	0	X10	±1V	2uS	
0	1	0	0	X100	±100mV	3uS	E
0	0	1	0	X200	±50mV	5uS	J, K
0	1	1	0	X300	±33mV	9uS	
0	0	0	1	X500	±20mV	11uS	S, T, R
0	1	0	1	X600	±16.7mV	21uS	
0	0	1	1	X700	±14.2mV	33uS	B
0	1	1	1	X800	±12.5mV	52uS	

2.2 OUTPUT CHANNEL SELECTION

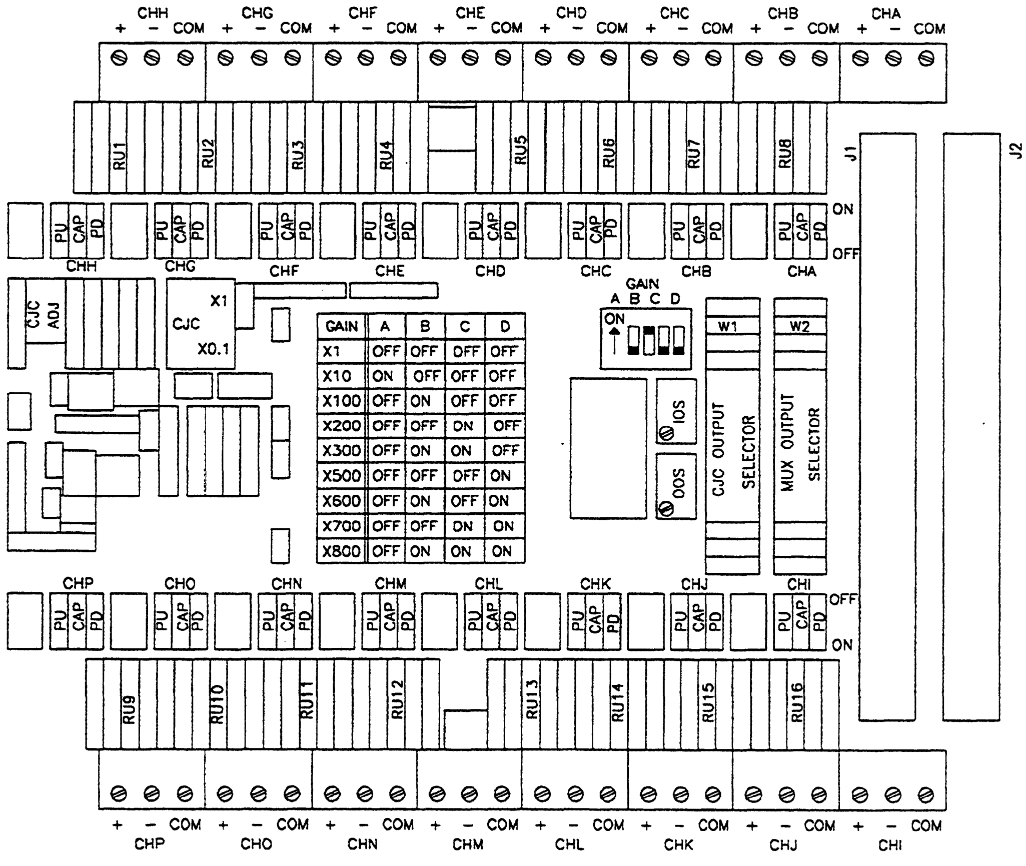
The output of the EXP-20's instrumentation amplifier can be jumped to any one of eight differential (16 single-ended) DAS-20 analog input channels using jumper block W2 (MUX OUTPUT SELECTOR; see board layout diagram). The W2 block contains three columns of pins in 16 rows. W2-block columns are labelled *SIG, OUT,* and *COM* while the rows are numbered *0* through *15.* To select an output for a particular DAS-20 channel, the user must plug a jumper across the two pins of the SIG/OUT columns at the row number that corresponds to the **DAS-20 Input Channel** (for example, to select for DAS-20 Channel 5, place a jumper across the SIG/OUT pins at Row 5). If the output is to be single-ended, no other jumpers are required on the W2 block. If the output is to be differential, the user must plug an additional jumper onto the W2 block across the two pins of the COM/OUT columns at the row number corresponding to the value **DAS-20 INPUT CHANNEL + 8**; for example, for DAS-20 Input Channel 5, the jumper must link the COM/OUT pins at 5 + 8, which is Row 13. Use the following table as a guide for jumper placement.

2.3 COLD JUNCTION COMPENSATION

Cold-Junction Compensation of the EXP-20 can be jumpered to any one of eight differential (16 single-ended) DAS-20 analog input channels using Jumper Block W1 (CJC OUTPUT SELECTOR; see board layout diagram). The W1 block contains three columns of pins in 16 rows. W1-block columns are labelled *SIG*, *OUT*, and *COM* while the rows are numbered *0* through *15*. To select an output for a particular DAS-20 channel, the user must place a jumper across the two pins of the SIG/OUT columns at the row number that corresponds to the **DAS-20 Input Channel** (for example, to select for DAS-20 Channel 7, place a jumper across the SIG/OUT pins at Row 7). If the output is to be single-ended, no other jumpers are required on the W1 block. If the output is to be differential, the user must plug an additional jumper onto the W1 block across the two pins of the COM/OUT columns at the row number corresponding to the value **DAS-20 INPUT CHANNEL + 8**; for example, for DAS-20 Input Channel 7, the jumper must link the COM/OUT pins at 7 + 8, which is Row 15. Be sure that the DAS-20 channel selected for the CJC is different from the one selected for the output.

2.4 SIGNAL INPUT CONNECTIONS

Once you have set the gain for the full-scale input range and you have chosen an output channel, you may make signal connections. Note that the EXP-20 employs a differential-input configuration (see the illustrations below). Therefore, if one or more of your input signals are single-ended, simply wire the signal to the "+" terminal and tie the "-" and COM terminals together for the selected channel. If the input signal is from a grounded source, do not connect the EXP-20 "COM" terminal to the "-" low terminal. If, however, your signal is from a floating (non-grounded) source (such as thermocouples), it is best to tie the "COM" and "-" terminals together. The label on the EXP-20 shows the position of all jumpers, switches, and input terminals along with the output channel assignments for each. Refer to this label when connecting signals, etc. to the EXP-20.


CHH CHG CHF CHE CHD CHC CHB CHA
+ - COM
RU1 RU2 RU3 RU4 RU5 RU6 RU7 RU8
J1 J2
PU CAP PD
ON OFF
CJC ADJ
X1 CJC X0.1
GAIN A B C D
ON
W1 W2
IOS OOS
CJC OUTPUT SELECTOR
MUX OUTPUT SELECTOR
CHP CHO CHN CHM CHL CHK CHJ CHI
RU9 RU10 RU11 RU12 RU13 RU14 RU15 RU16

GAIN	A	B	C	D
X1	OFF	OFF	OFF	OFF
X10	ON	OFF	OFF	OFF
X100	OFF	ON	OFF	OFF
X200	OFF	OFF	ON	OFF
X300	OFF	ON	ON	OFF
X500	OFF	OFF	OFF	ON
X600	OFF	ON	OFF	ON
X700	OFF	OFF	ON	ON
X800	OFF	ON	ON	ON

2.5 PU, PD, & CAP JUMPERS

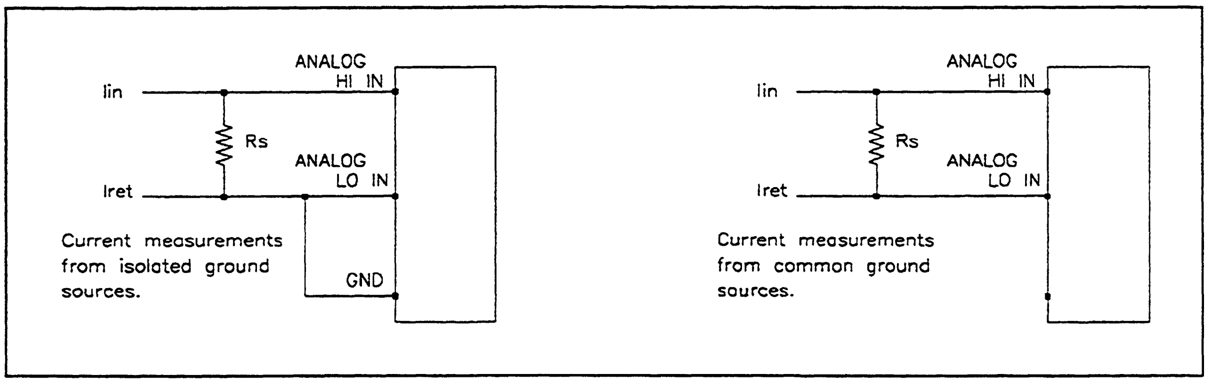
Each of the EXP-20 board's input channels contains a 9-pin jumper block for selecting functions entitled *PU* (Pull Up), *CAP* (Capacitor), and *PD* (Pull Down), as shown in the board's layout diagram. Each of these three functions uses a 3-pin row as follows: the outer pin (towards the outer edge of the board) is for ON, the center pin is common, and the inner pin is for OFF. Thus a jumper across the ON and common pins turns the function On, while a jumper across the OFF and common pins turns the function Off.

To enable detection for an open thermocouple on a particular channel, place jumpers across the ON/common pins of both the PU and PD functions for that channel. This setting will give a minus full-scale reading for an open thermocouple (the software must be designed to recognize minus full scale as an open thermocouple). If detection for an open thermocouple is not needed, place the PU and PD jumpers across the OFF/common pins or remove them from the board.

Placing a CAP jumper across the ON/common pins enables a low-pass filter for noise rejection. Implementing the filter (via CAP) causes an increase in settling time.

2.6 MEASURING CURRENT

MilliAmp signals may be measured on any or all inputs of the EXP-20 provided a shunt resistor is wired in parallel with the signal input leads. A shunt is a relatively low resistance bypass for otherwise damaging current. Provisions have been made for wiring shunt resistors directly onto the board for each input channel (RU1 - RU16 are for this purpose). Also, be certain to check the gain setting for the Full Scale Range prior to wiring current signals to it. The value of the shunt may be calculated as shown below.


ANALOG HI IN
Iin
Rs
ANALOG LO IN
Iret
Current measurements from isolated ground sources.
GND
ANALOG HI IN
Iin
Rs
ANALOG LO IN
Iret
Current measurements from common ground sources.

Rs = E(max)/I(actual)

Where: Rs = value of shunt resistor
E(max) = Max voltage of gain range
I(actual) = Actual current being measured

2.7 CJC GAIN SELECTION

In addition to the selection capabilities already described (signal and CJC output channel selection, signal gain, PU, PD, and CAP), a value for CJC gain is also selectable. This value is jumper-selectable from options of X1 and X0.1 on a jumper block labelled *CJC*. The X1 option gives 200mV per °C; X0.1 gives 20mV per °C. The higher gain may be used when highly accurate reference temperature (for CJC) readings are required.

2.8 CASCADING MULTIPLE EXP-20'S

Up to 16 EXP-20s (256 analog channels) may be cascaded together by simply interconnecting them via the 50 pin connectors on each EXP-20 (use cable #CACC-2000). Note that each EXP-20 connected to the same analog board (DAS-20) must have its own distinct output channel. However, multiple EXP-20's measuring thermocouples do not require seperate CJC channels.

Section 3

SOFTWARE

The EXP-20 is supplied with a (DAS-20 compatible) software device driver for MUX channel access. This device driver is relocatable in memory and may be used with virtually all PC compatible machines operating under MS-DOS. It is a self-contained, CALLable routine compatible with BASIC.

Distribution software furnished with the DAS-20 includes to example programs for the EXP-20. One of these programs, EXP20.BAS, works with analog inputs while the other, THERM.BAS, works with thermocouple inputs.

* * * * *

Section 4
CALIBRATION

4.1 PERIODIC CALIBRATION/ADJUSTMENTS

The EXP-20, like all MetraByte products, has been precalibrated and fully tested prior to shipment. However, periodic recalibration and adjustment may be required to compensate for component aging, etc. The frequency of adjustment varies widely depending upon the EXP-20 operating environment. A dust-free, air-conditioned, laboratory environment may indicate EXP-20 adjustments once each year or even once every two years. Whereas EXP-20 usage on a shop floor or where large temperature gradients are prevalent may mean more frequent recalibration cycles. A good rule of thumb is, "If you think it needs adjustment, it probably does." Recalibrating the EXP-20 is a simple yet important procedure. It is good to bear in mind that the quality of any measurement is a direct reflection of the care and precision taken during the recalibration process.

There are three adjustment points on the EXP-20. They are Input Offset Adjustment (IOS), Output Offset Adjustment (OOS), and CJC adjustment. Make all adjustments with the gain settings you would normally use for your application and while monitoring a single channel while adjusting the relevant potentiometer.

4.2 CJC ADJUSTMENT

An accurate, stable room temperature reading is required for the following procedure:

1. Set the CJC Gain to the desired value for your application.
2. Monitor the CJC channel via software while adjusting the CJC ADJ potentiometer to read the actual room temperature.

NOTE Gain of X1; one degree equals 20 mV. 25 Deg C = 0.5 Volts.

Gain of X10; one degree equals 200 mV. 25 Deg C = 5.0 Volts.

4.3 INPUT (IOS) AND OUTPUT (OOS) OFFSET ADJUSTMENT

1. Set the signal output gain to X1.
2. Wire the EXP-20 "+", "-", and "COM" terminals together.
3. Monitor the channel via software while adjusting OOS pot for best 0.
4. Set the gain to what you would normally use.
5. Monitor the channel while adjusting IOS pot for best 0.

6. Monitor the channel while adjusting OOS pot for best 0.
7. Recheck the IOS for best 0 reading.

* * * * *

Section 5

EXP-20 SPECIFICATIONS

Analog Inputs:	16 differential pairs.				
Thermocouple Compatibility:	J, K, T, R, S, B, and E.				
Cold Junction Compensation:	User selectable +20 mV/°C, +200 mV/°C (0.012207 °C/bit).				
Max Input Voltage Range:	± 10 Vdc.				
Max Voltage Offset Drift:	± (5+100/Gain)μV/°C.				
Instrument Amplifier Gains:	1, 10, 100, 200, 300, 500, 600, 700, 800 DIP switch selectable/board.				
Gain Error:	X1 0.002%	(typ),	0.04%	(max)	
	X10 0.01%	(typ),	0.1%	(max)	
	X100 0.02%	(typ),	0.2%	(max)	
	X200 0.04%	(typ),	0.4%	(max)	
	X500 0.1%	(typ),	1.0%	(max)	
Gain Nonlinearity (F.S.):	X1 ±0.001	(typ),	±0.01	(max)	
	X10 ±0.002	(typ),	±0.01	(max)	
	X100 ±0.004	(typ),	±0.02	(max)	
	X200 ±0.006	(typ),	±0.02	(max)	
	X500 ±0.01	(typ),	±0.04	(max)	
Typical MUX settling time:	X1 4μS	(0.1%),	5μS	(0.01%)	
(Times quoted are without	X10 2μS	(0.1%),	3μS	(0.01%)	
input filter)	X100 3μS	(0.1%),	4μS	(0.01%)	
	X200 5μS	(0.1%),	7μS	(0.01%)	
	X500 11μS	(0.1%),	16μS	(0.01%)	
Common Mode Rejection:	X1 90dB				
(1 KOhm source imbalance)	X10 104dB				
	X100 110dB				
	X200 110dB				
	X500 110dB				
Ouput Configuration:	16 single-ended, 8 differential; user selectable via jumper plugs for CJC and Amplifier Output				

* * * * *

APPENDIX **D**

THE SSH-4

CONTENTS

* * * * *

Section 1

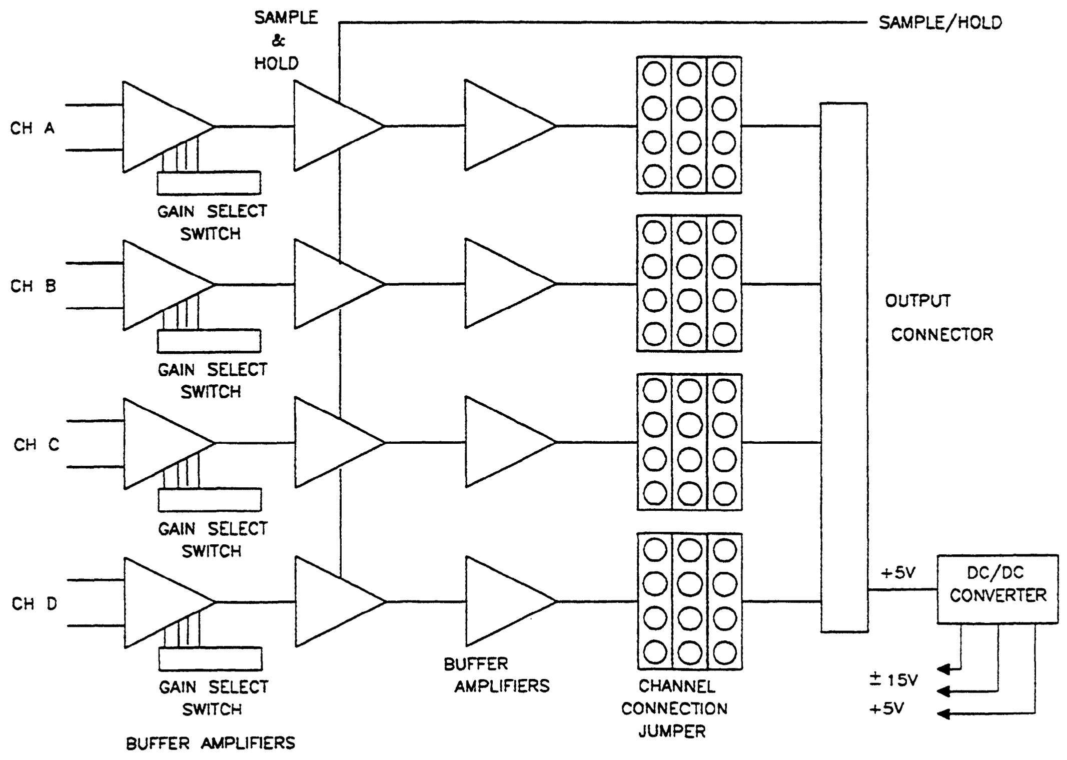
INTRODUCTION

1.1 GENERAL

The SSH-4 is a 4 channel, simultaneous sample & hold front end for use with MetraByte's DAS-20. It permits simultaneous sampling of four input signals, after which the signals may be read by the analog board one at a time. The SSH-4 is configured as four bipolar, differentially measured inputs with DIP switch selectable gains of 1, 10, 100, 200, 300, 500, 600, 700, and 800. Maximum voltage input is ±10 Vdc. Output from the SSH-4 is jumper selectable as any 4 of 16 single-ended or 4 of 8 differential channels. Potentiometer adjustments for input and output offset (IOS, OOS) make calibration a fast, easy procedure. The signal acquisition period may be reconfigured for your specific application. Data collection is triggered via a Sample and Hold pulse (TTL) of at least 10 µS duration. The DAS-20 provides this signal (S/H) at its output connector.

Two 50-pin headers allow easy daisy-chain expansion when more than four channels must be collected simultaneously as well as for connection to the DAS-20 or VMECAI-16 (use cable #CACC-2000). The SSH-4 draws a maximum of 125mA @ 5Vdc from the bus supplies. A single DC to DC converter for ±15 Vdc assures full-scale coverage of all input ranges.

1.2 BLOCK DIAGRAM



1.3 FEATURES

- 4 differential, bipolar analog inputs.
- Nine DIP switch selectable gains.
- On-board DC to DC converter.
- Simple calibration procedure.
- Choice of output configurations.
- Pin compatible with DAS-20 and VMECAI-16.

Section 2
HARDWARE

2.1 GAIN SELECTION

The SSH-4 supports nine standard gains. Gain selection is accomplished via Switch SA through SD. Resultant gain and full-scale input ranges for the switch settings are shown below.

SWITCH 3				GAIN	BIPOLAR RANGE	SETTLING TIME	
1	2	3	4				
0	0	0	0	X1	10V	4ms	
1	0	0	0	X10	1.0V	2ms	0 = OFF
0	1	0	0	X100	100mV	3ms	1 = ON
0	0	1	0	X200	50mV	5ms	
0	1	1	0	X300	33mV	9ms	
0	0	0	1	X500	20mV	11ms	
0	1	0	1	X600	16.7mV	21ms	
0	0	1	1	X700	14.3mV	33ms	
0	1	1	1	X800	12.5mV	52ms	

Since both the SSH-4 and all of the compatible Analog Input modules have provisions for increased gain, avoid setting a combined gain higher than 8000 as this may cause excessive noise and therefore erroneous readings.

2.2 OUTPUT CHANNEL SELECTION

The SSH-4 samples four signals simultaneously then transfers these data to four of sixteen single-ended or four of eight differential analog channels. Output channel selection is accomplished by a series of jumpers. For single-ended outputs, only the SIGNAL/OUTPUT jumper for each block need be connected. Differential outputs require both the SIGNAL/OUTPUT and OUTPUT/ACOMMON jumpers be set as shown below.

SIGNAL-OUTPUT	OUTPUT-ACOMMON
Chan 0	8
4	12
1	9
5	13
2	10
6	14
3	11
7	15

For example, suppose the CHA output is connected to a differential channel 0 input on the DAS-20. The analog "+" portion of the signal should be connected by the SIGNAL-OUTPUT jumper while the reference portion of the signal is connected via the OUTPUT-ACOMMON jumper (CH8).

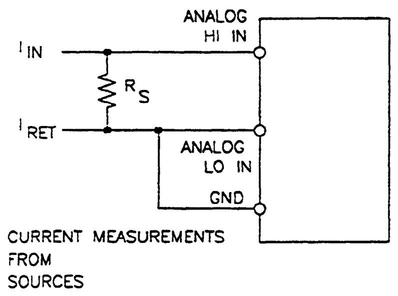
WARNING: When cascading SSH-4's from a single Analog board, bear in mind that each SSH-4 must have (4) distinct, unused output channels. Channel conflicts are not only an inconvenience but may also cause damage to the SSH-4.

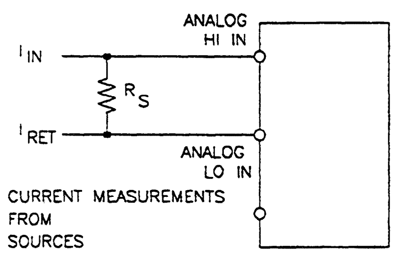
2.3 INPUT SIGNAL CONNECTIONS

Once you have set the gain for the full-scale range of your input signals, you may make signal connections. Note that the SSH-4 employs a differential input configuration. Therefore, if one or more of your input signals is single-ended, simply wire the signal lead to the "+" terminal and tie the "-" and "G" terminals together. If your signal source is grounded, do not connect the SSH-4 "G" terminal to the "-" terminal. If, however, your signal source is floating (non-grounded), it is best to connect the "-" and "G" terminals together. The label on the SSH-4 shows the position of all jumpers, switches, and input terminals along with the output channel assignments for each. This label should be consulted when connecting signals or configuring the SSH-4.

2.4 MEASURING CURRENT

The SSH-4 measures milliAmp signals using a shunt resistor wired in parallel with the signal input leads. A shunt is a relatively low resistance bypass for otherwise damaging current. The value of the shunt may be calculated as shown below. The following diagram may also be of help when measuring current signals.





R_S = 2/FULL-SCALE INPUT CURRENT

For example, if Full Scale Input Current is to be ±20mA, then

R_S = 2/0.020 = 100 ohms.

The following list shows shunt resistors for some common current inputs:

±200mA - 10 Ohms, 1 Watt	±200uA - 10 KOhms
±20mA - 100 Ohms	±20uA - 100 KOhms
±2mA - 1000 Ohms	

Resistors should be ±1% or better metal trim.

2.5 CASCADING MULTIPLE SSH-4s

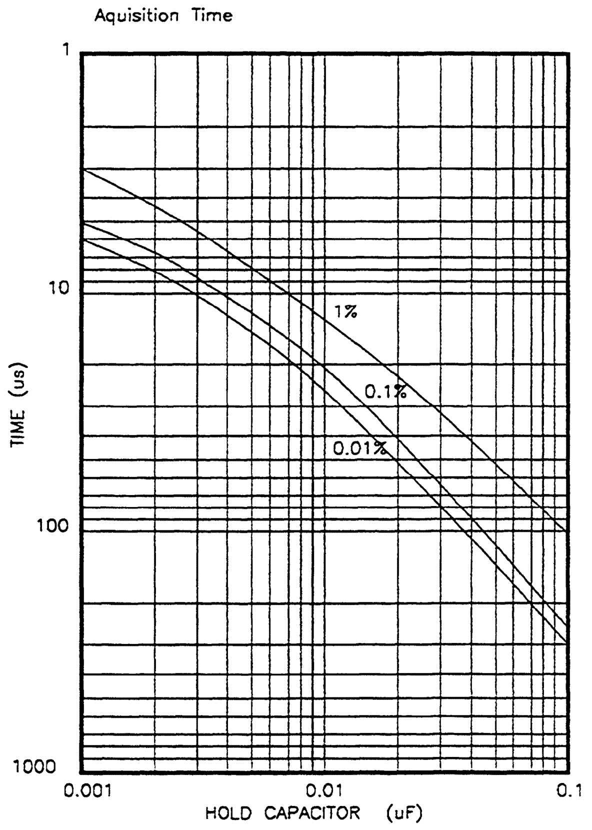
Multiple SSH-4's may be daisy-chained to a maximum of 4 boards (16 channels) by simply interconnecting them via one of the 50-pin headers on each SSH-4 (use cable #CACC-2000).

WARNING: Each SSH-4 connected to a single analog board must have four unused, distinct addresses. Channel address conflicts are not only an inconvenience, but may result in damage to the board. Also, data from the SSH-4 output channel must match the input channel of the analog board being used.

2.6 USER CAPACITOR

The SSH-4 comes configured with a 5600 pF capacitor allowing a 10 µS signal acquisition window. Longer signal acquisition times may be set by simply adding a second capacitor for the relevant channel. Note that a larger capacitor will lower the output droop rate proportionally. The following graph is reproduced from the National Semiconductor LF 398 and may be used as a guideline for addition of other cap values. The standard 5600 pF capacitor is the smallest value that can be used on the SSH-4 and that additional capacitance is added in parallel.

Vin = 0 to 10V; Tj = 25°C.



* * * * *

Section 3

SOFTWARE

The SSH-4 is supplied with a (DAS-20 compatible) software device driver. This device driver is relocatable in memory and may be used with virtually all PC-compatible machines operating in a variety of environments (MS-DOS, CP/M. etc.).

An example of programming the SSH-4 (in conjunction with the DAS-20 board is provided in the SSH4.BAS program which has been included on the DAS-20 disk. The example illustrates DAS-20 programming using the supplied SSH-4 driver (MODE 29). It is written in BASICA as implemented on the IBM PC.

* * * * *

Section 4
PERIODIC CALIBRATION/ADJUSTMENTS

The SSH-4, like all MetraByte products, has been precalibrated and fully tested prior to shipment. However, periodic readjustment may be required due to component aging, etc. The frequency of adjustment varies widely depending upon the SSH-4 operating environment. A dust-free, air-conditioned, laboratory environment may indicate SSH-4 adjustment once each year or two. Whereas SSH-4 usage on a shop floor or where large temperature gradients are prevalent may mean more frequent adjustments. A good rule of thumb is; "If you think it needs adjustment, it probably does." Adjusting the SSH-4 is quite easy with only two trim pots per channel. It is good to bear in mind that the quality of any measurement is a direct reflection of the care and precision taken during readjustment.

The two adjustments are Input Offset Adjustment (IOS) and Output Offset Adjustment (OOS). No special tools or instruments are required. All adjustments should be done using the gain settings that you would normally use for your application. Adjustments may be made by monitoring a single channel while turning the relevant trim pot.

Input OffSet and Output OffSet Adjustment:

1. Set the signal gain to X1.
2. Connect the "+", "-", and "G" terminals together for each channel.
3. Monitor each channel via software and adjust the OOS pots for best 0.
4. Set the gain for each channel to what you would normally use.
5. Monitor each channel while turning the IOS pots for best 0 reading.
6. Monitor each channel while turning the OOS pots for best 0.
7. Recheck the IOS (adjust as necessary) for best 0 reading.

* * * * *

Section 5

SSH-4 SPECIFICATIONS

Analog Inputs:	4 differential				
Max Input Voltage Range:	±10 Vdc				
Max Voltage Offset Drift:	±(5+100/Gain)µV/°C				
Instrument Amplifier Gains:	1, 10, 100, 200, 300, 500, 600, 700, 800 DIP switch selectable/board				
Gain Error:	X1	0.002%	(typ),	0.04%	(max)
	X10	0.01%	(typ),	0.1%	(max)
	X100	0.02%	(typ),	0.2%	(max)
	X200	0.04%	(typ),	0.4%	(max)
	X500	0.1%	(typ),	1.0%	(max)
Gain Nonlinearity (F.S.):	X1	±0.001	(typ),	±0.01	(max)
	X10	±0.002	(typ),	±0.01	(max)
	X100	±0.004	(typ),	±0.02	(max)
	X200	±0.006	(typ),	±0.02	(max)
	X500	±0.01	(typ),	±0.04	(max)
Typical MUX settling time:	X1	4µS	(0.1%),	5µS	(0.01%)
(Times quoted are without	X10	2µS	(0.1%),	3µS	(0.01%)
input filter)	X100	3µS	(0.1%),	4µS	(0.01%)
	X200	5µS	(0.1%),	7µS	(0.01%)
	X500	11µS	(0.1%),	16µS	(0.01%)
Common Mode Rejection:	X1	90dB			
(1 KOhm source imbalance)	X10	104dB			
	X100	110dB			
	X200	110dB			
	X500	110dB			
Ouput Configuration:	4 single-ended or 4 differential User-selectable Ch # via jumper plugs.				

* * * * *

KEITHLEY METRABYTE

440 Myles Standish Boulevard
Taunton, MA 02780
508-880-3000