FOR - 16
MANUAL

MBC MetraByte Corporation

3/85

****

WARRANTY

All products manufactured by MetraByte are warranted against defective materials and workmanship for a period of One Year from the date of delivery to the original purchaser. Any product found to be defective within the warranty period will, at the option of MetraByte, be repaired or replaced. This warranty does not apply to products which have been damaged by improper use.

****

****

!! WARNING !!

MetraByte Corporation 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 of other extremely critical systems.**

****

MetraByte Corporation
440 Myles Standish Boulevard.
Taunton, MA. 02780 U.S.A.
Phone: (508) 880-3000 Telex: 503989

61060A

TABLE OF CONTENTS

1.0 INTRODUCTION

The MetraByte DASH-16 Data Acquisition Fortran library is a comprehensive set of A/D and D/A driver Functions / Subroutines used to extend the Fortran compiler. The DASH-16 Fortran Library also contains a set of general purpose I/O functions (INP, OUT, PEEK & POKE) to write and read bytes or words to or from a user defined I/O port or memory location over the entire 8088/86 address range of 0 to (2^16 - 1). This allows the user to directly drive other MetraByte I/O devices e.g. the PIO-12 Parallel I/O board directly for a variety of control applications and also allows memory mapped devices to be used with Fortran. The DASH - 16 Library follows the linking format as required by the MicroSoft Fortran Compiler Version 3.2, and is outlined in the following sections.

1.10 SOFTWARE INSTALLATION AND BACKUP

The installation of the DASH-16 interface board is outlined in the DASH-16 manual chapter 2. The selection of the BASE address and Interrupt and DMA levels are internally set as noted in chapter 1. (Base Address = Hex 300, DMA = 1, INT = programmable). **A BACKUP COPY SHOULD BE USED FOR PROGRAM DEVELOPMENT AND THE MASTER DISK STORED IN A SAFE PLACE.** The disk format is Single Side Double Density DOS 1.10 format and is read compatible for all versions of PC-DOS. Chapter 5 of the DASH-16 manual shows the hookup of the counter/timers for external trigger of the A/D.

The DAS16FOR.LIB will support DOS 1.10 through DOS 3.00 and MS Fortran compiler versions from 3.0 to 3.2. Programmers should use the MS LINK.EXE which is supplied with your Fortran Compiler to obtain upward compatibility. **Do not** use the LINK.EXE supplied with DOS as several revisions and adjustments have been made in the linker program.

1.11 USING THE LIBRARY

The DASH-16 Fortran library is used at the linker level as most libraries. Once the users Fortran program has been compiled according to the Fortran users guide the linker is ready to produce an run-time EXE file. The Linker will automatically search the Fortran libraries required to link the standard functions. In order to link the DASH-16 library the user will respond with DAS16FOR.LIB to the question of LIBRARY: when asked. The session would be as follows.

```
A>LINK
MicroSoft linker version XX ......

Object modules [.OBJ] filespec
Run File [ FILESPEC.EXE ]: <return>
List Map [NUL.MAP]: <return>
Libraries [.LIB]: DAS16FOR
```

The DAS16FOR.LIB library should be the last library linked during the link session. The data segments used in the DAS16FOR library are labeled DATA and not _DATA as in MS Fortran 3.30. This will still link without errors since the DGROUP combines all data segments labeled DGROUP under one segment. See linker manual.

At this point all will be automatic. The library will be loaded as needed by the Fortran program. When the prompt displays the program may be run by typing the name. The following sections will explain the library functions and the Fortran format.

```
A>FILESPEC
```

This will execute the .EXE file and run the program

2.0 DASH-16 FORTRAN SUBROUTINE LIBRARY DESCRIPTIONS

All the following DASH-16 subroutines follow the Standard Fortran functions/subroutines and may be nested up to the limits of the compiler. Since the following library becomes part of the Fortran library the following function/subroutine names become **RESERVED** names and may not be used as labels. The variable names used for the DASH-16 library functions are considered **INTEGER*2** type for all variables and must be adhered to or else strange errors will occur. Using these function names as labels will introduce bizarre run and linking errors. The library consists of two types of functions, the unique DASH - 16 functions and the general purpose I/O type functions. The following is a list of the functions/subroutines incorporated in the library. The page numbers have been added to this section also for the convenience of the user.

********** DASH - 16 UNIQUE SUBROUTINES/FUNCTIONS **********

********* GENERAL PURPOSE IO FUNCTIONS *********

ADINIT (BASADR, DMALEV, INTLEV, RTNFLG)

This function initializes the DASH-16 identification parameters in order for the library functions to be used. The function does not have to be executed within a Fortran module since the library has default values. If other than the default values are used then this function must be executed. The ADINIT function also allows the user to setup a second board for communications with the system, however only one board is allowed to be operational at a time. If the user wishes to run more than one board in the system, this command should be run for all the boards in the system first. The parameter limits are as follows. All variable names are **INTEGER*2** type (2 Bytes length).

BASADR = Base Address of DASH - 16 board (0100H to 03F0H) This address range is checked before further execution.

DMALEV = DMA Channel number of DASH - 16 board (1 or 3) Only channel 1 or 3 is allowed.

INTLEV = INTERRUPT Level of DASH - 16 board (2 to 7) This level is also checked for range.

RTNFLG = Flag Return Code for current selected function.

0000H = function successful. continue normally.
0001H = System already in use. can't continue.
0003H = BASADR variable range error, <100H,>3F0H
0004H = INTLEV variable range error, <2 or >7.
0005H = DMALEV variable range error, not 1 or 3

EXAMPLE:

```
C      ** SETUP BOARD PARAMETERS AS INTEGER * 2 TYPE **
C
      INTEGER*2 BASADR,DMALEV,INTLEV,FLGRTN
      BASADR = #300
      DMALEV = #3
      INTLEV = #2
      FLGRTN = #0
C
C     ******* EXECUTE FUNCTION CALL *******
C
      CALL ADINIT ( BASADR, DMALEV, INTLEV, FLGRTN )
      IF (FLGRTN .NE. #0) GOTO 10
C     ...... USER CONTINUES PROGRAM .......
C
10    ----- USER ERROR HANDLER, CHECK RTNFLG FOR ERRORS  -----
      END
```

ADCONV (MODE, SCH, FCH, NOS, DATIN(n), RTNFLG)

This function allows the user to collect data via the A/D converter using one of five modes. The user also selects the number of channels Start to Final, and the Number Of Scans for data collection. A Scan is defined as the Start Channel to the Final Channel (SCH to FCH). If the Start Channel (SCH) = 0, and the Final Channel (FCH) = 7, then one scan would collect 8 channels of data into the array DATIN(n). The array size must be large enough to receive the data, at least [NOS*(FCH-SCH +1)]. If SCH = FCH then the Number Of Scans (NOS) will be the actual number of conversions for that channel. If 100 conversions are required on channel 3 then, SCH = FCH = 3, and NOS = 100, The array must be at least DATIN(100) [INTEGER*2 type] in a DIMENSION statement. In MODES 1, 2, 3 and 4 (external trigger modes) typing **Esc** key will terminate the run and execute the next Fortran statement after the ADCONV statement. This will allow termination of data collection with out re- booting the system. All data previously collected before Esc key was pressed will be valid and the return flag code will be HEX 1000. (#1000).

MODE = Data Collection Mode A/D only

0 = Internal start of conversion (start on entry) Immediate start of conversion by software and collect the specified number of conversions to the specified array. This routine is program control only (NO DMA).

1 = External trigger for each conversion. Transfer data to the specified array under program control. The A/D starts with the external trigger for each conversion. The number of conversions is determined by the NOS and the number of channels. This mode is also program control data transfer (NO DMA).

2 = External trigger for each block (SCH-FCH) of channels (NO DMA) under program control. An error code will be returned if the limits are exceeded.

3 = External trigger for each conversion (DMA). This routine collects the data after each external trigger and transfers the data to the array via DMA. The user remains in this routine until all the specified conversions are completed. The user may interrupt the

data collection by pressing the Esc key.

4 = External trigger for each block (SCH-FCH) of channels using DMA. An error code will be returned if the limits are exceeded. Although DMA transfer, this mode can only be driven at interrupt rates.

5 = External Trigger Background DMA Data Transfer. This mode allows the user to collect data in the background while running a secondary program in the foreground. The Background data collection runs at the maximum transfer rate of the A/D converter or the rate of the external trigger. It is the users responsibility to insure the variable data array is not changed during data collection. The user may check the status of the data transfer at any time by the DMASTA function which returns the current number of conversions and the current DASH-16 board status. The user may terminate the data collection before the normal end of transfer by the DMAOFF function.

6 = External Trigger DMA mode Auto-Initialize. This mode allows the user to collect data into the specified array continuously in the background. The data is collected until a DMAOFF function is executed. It is the users responsibility to disable the DMA operation when data collection is no longer required.

NOTE: The output of counter 2 may be internally connected to the A/D trigger input (IP0) by adding 16 decimal (#10 hex) to the mode.

EXAMPLE: MODE = 4+16 Will be mode 4 and counter 2 output will be the Trigger for the A/D converter.

SCH = Start Channel (0 - 15 Single Ended) (0 - 7 Diff.) This channel is automatically reloaded when the FCH (final channel) is reached in the Mux scan register. An error code will be returned if the limits are exceeded.

FCH = Final Channel (0 - 15 Single Ended) (0 - 7 Diff.). This channel is automatically reloaded when the SCH (start channel) reloads the Mux scan register. An error code will be returned if the limits are exceeded.

NOS = Number Of Scans for each group of channels specified by SCH and FCH. NOC (number of conversions) is defined by the equation, NOC = NOS * (FCH - SCH + 1). The number of conversions must be with in the range of NOC max = 32760, NOC min = 1. An error code will be returned if the limits are exceeded.

RTNFLG = Flag Return code for status of function selected.

HEX CODE 0 = Transfer ok
1 = SCH, FCH channel limits exceeded for Differential
2 = SCH, FCH channel limits exceeded for Single Ended
3 = NOC Limit error < 1 or > 32760
4 = A/D DMA mode or Board Busy
5 = Time out. No EOC from convertor
6 = DMA Vector level range error
100 = DMA / Data collection hardware error
1000 = Function Terminated by Esc key sequence

DATIN(n) = Data Transfer variable **INTEGER*2** type only. !!! This variable is used for data transfer and may be a single variable if only a single channel is to be converted. DATIN(n) may be an array of max length less than or equal to 32760 for the data conversion. This is due to the fact of segments of 16 bits and a 16 byte boundary constraint. The variable must be a word (2 bytes) type integer. The size n = NOS*(FCH-SCH+1) minimum.

EXAMPLE:

```
C     **** INITIALIZE VARIABLE'S TYPE FOR USE WITH FUNCTION ****
C
      INTEGER*2 MODE, SCH, FCH, NOS, RTNFLG DATIN
C
C     **** DIMENSION DATA ARRAY FOR (FCH7-SCH0+1)*100 = 800
C
      DIMENSION DATIN (800)
C
C     ***** INITIALIZE VARIABLES ******
C
      MODE0 = 0
      SCH0  = 0
      FCH7  = 7
      NOS100  = 100
      RTNFLG = 0
C
C
C     ******* COLLECT DATA FROM A/D INTO ARRAY *******
C     - THE DATA WILL BE COLLECTED UNDER PROGRAM CONTROL -
C
      CALL ADCONV ( MODE0, SCH0, FCH7, NOS100, DATIN(1), RTNFLG )
      IF (RTNFLG .NE. 0) GOTO 400
C
C     ...... USER CONTINUES PROGRAM HERE .........
C                 .....................
C                 .....................
C     ****** ERROR HANDLER IF RTNFLG NOT ZERO ******
C
400   WRITE (*,401) RTNFLG
401   FORMAT (1X, 'ERROR DURING A/D CONVERSION FUNCTION IS ', I2)
      END
```

D16FIX (DATAX, CHANX)

This function allows the user to condition the A/D data collected from ADCONV in mode 5 or 6. The A/D data is supplied by the variable DATAX and the function returns two values. The first value is the CHANNEL the data was collected on and returns it to CHANX variable. The second value is returned as a function value in the form FIXDATA = D16FIX (DATAX, CHANX), where FIXDATA stores the 12 bit A/D integer data in the range 00 to 4095 for unipolar and 00 +/- 2048 for bipolar settings of the DASH-16.

EXAMPLE:

```
C     ***** ASSUME ARRAY OF DATA DATI0(1000) WAS COLLECTED ****
C
      INTEGER*2 DATI0,DATX,CHANX, J
C
C     **** DECLARE THE FUNCTION AS AN INTEGER FUNCTION ***
C
      INTEGER*2 D16FIX
C
      DIMENSION CHANX(1000),DATX(1000),DATI0(1000)
C
C     MAKE NEW ARRAY OF FIXED DATA AND ASSOCIATED CHANNEL #
C
      DO 100 J=1,1000
      DATX(J) = D16FIX (DATI0(J), CHANX(J))
      100 CONTINUE
C
      STOP
      END
```

DMASTA (RTNSTATUS)

This routine allows the user to monitor the status of the DASH-16 while it is collecting data in mode 5 or 6. The routine may be either a subroutine for just the current status or a function for the status and the current number of conversions. If the function is executed and mode 5 is inactive the returned value will be 0 else the returned value will be the current number of conversions. The status is returned in either case. If the word count = 0 and the DMASTA function is executed then a -32767 or some other negative value will return. The bit assignments for the Status corresponds to the DASH-16 CONTROL register and STATUS register found in sections 3.5 and 3.6 of the DASH-16 manual.

INTEGER BIT ASSIGNMENT

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DASH - 16 CONTROL REGISTER SECTION 3.6 DASH-16 MANUAL								DASH - 16 STATUS REGISTER SECTION 3.5 DASH-16 MANUAL							

EXAMPLE:

```
C     ***** SETUP AD MODE 5 AND ARRAYS ******
C
      INTEGER*2 DATIN, NOC, DAS16STAT, RTNFLG, DMASTA
C
      DIMENSION DATIN (1024)
C
C     --- SETUP COUNTER 0 FOR 1 KHz TICKS ---
C     -- CONNECT CNTR 0 OUT TO TRIG IN, OP0 TO CNT0 GATE IN. --
C
      CALL CNTM0 (2,1000)
C
      IF ( ADCONV( 5, 1, 1, NOS100, DATIN(1), RTNFLG) .NE. 0))
     @ GOTO 400
C
C     --- READ STATUS AND PRINT THE CONVERSION NUMBERS ---
C
100   NOC = DMASTA ( DAS16STAT )
      IF ( NOC .GT. 0 .AND. NOC .LE. NOS100 ) GOTO 200
      GOTO 500
C
200   WRITE (*,201) NOC
```

```
201   FORMAT (1X, 'THE CURRENT CONVERSION IS ',I5)
      GOTO 100
C
400   .... ERROR HANDLER ROUTINE PRINT ERROR ......
500   END
```

DMAOFF

This routine allows the user to terminate the DMA data collection in mode 5 or 6 of the ADCONV function. This function does not have any variables associated with it and may be used any time the user wishes to reset the dma/interrupt hardware to a known inactive state. This routine **MUST** be executed before termination of program if modes 5 or 6 are used.

EXAMPLE:

```
C     ****** INITIATE A TERMINATE DMA ******
C
      CALL DMAOFF
      END
```

DAOUT (DACN, DATOUT(n), RTNFLG)

This function allows the user to transfer data to a selected D/A converter channel or both D/A channels with one command. The D/A converters are 12 bit allowing the range of 0 to 4095 decimal. The DAOUT functions variables are all **INTEGER*2** type. The function DATOUT(n) may be any INTEGER variable name or array. If both D/A's are selected then the DATOUT(n) is expected to be an array of two. The return flag, RTNFLG, will return a value of 0 if all O.K. or a value of 1 if the DAC's have been called by another user task.

DACN = D/A Converter Selected for output

0 = D/A converter channel 0
1 = D/A converter channel 1
2 = Both D/A converters 0 & 1

RTNFLG = Return flag code for multitask

0 = transfer completed all OK
1 = function previously called by another task and data transfer is incomplete. No transfer takes place.

DATOUT(n) = INTEGER*2 (2 byte) Data variable. If DACn = 0 or 1 then DATOUT(n) may be a single INTEGER variable or an Array. If DACn = 2 then the function will expect the data to be an INTEGER Array and n will point to DAC channel 0.

EXAMPLE:

```
C     ***** SETUP INTEGER VARIABLES *****
C
      INTEGER*2 DAC, DATOUT, RTNFLG
C
      DAC = 0
      DATOUT = #400
      RTNFLG = 0
C
C     ***** TRANSFER TO DAC CHANNEL 0 *****
C
      CALL DAOUT ( DAC, DATOUT, RTNFLG )
      IF (RTNFLG .NE. 0) GOTO 10
C
C     ...... USER PROGRAM CONTINUES .......
                    ..............
```

```
                 ..............

10  ...... PROCESS RTNFLG RETURN CODE FOR MULTI-TASK ....
C        --------- IF RTNFLG = 1 THEN -----------
C      OUTPUT RTNFLG FOR SOMEONE ELSE IS USING THE SYSTEM
```

DIGOUT (DATOUT)

This function allows the user to transfer a four bit value (bits 3, 2, 1, 0) in the range of 0 to 15 to the digital output port on the DASH-16. The digital port is limited to 4 bits range, and the function passes only the least significant four bits MOD(2^4 - 1). The variable must be of **INTEGER*2** type.

DATOUT = data to be transferred. (00 to 15) DECIMAL

EXAMPLE:

```
C     ****** SETUP DATA TO BE TRANSFERRED ******
C
      INTEGER*2 DATOUT
      DATOUT = #03
C
C     ***** TRANSFER DATA TO DIGITAL PORT *****
C
      CALL DIGOUT ( DATOUT )
C
C     **** TRANSFER A CONSTANT VALUE FOR DATA ****
C
      CALL DIGOUT ( 15 )
C
C     ** TRANSFER A INTEGER USING A FUNCTION AS A VALUE **
C
      CALL DIGOUT ( IABS(7.45) )
C
C     -- THE DATA TRANSFERRED IS THE INTEGER VALUE OF 7 --
C

      END
```

DIGIN (DATAIN)

This function allows the user to read the four bits of data available at the digital input port on the DASH-16 board. The data range is returned INTEGER*2 format in the range of 0 to 15. The INTEGER variable DATAIN receives the data. This function may also be used directly with conditional statements.

DATAIN = INTEGER*2 data variable for data transfer.

EXAMPLE:

```
C     ***** SETUP VARIABLE FOR DATA TRANSFER *****
C
      INTEGER*2 DATAIN, X, DIGIN
C
      DATAIN = 0
C
C     ****** READ DIGITAL INPUT PORT BITS 0,1,2,3 ******
C
      X = DIGIN ( DATAIN )
C
C     ****** USE FUNCTION IN CONDITIONAL STATEMENT ******
C
      IF ( DIGIN (DATAIN) .EQ. 4 ) GOTO 400
C
      ..... PROGRAM CONTINUES IF NOT EQUAL TO 4 .....
      . DATA IS ALSO PASSED TO VARIABLE DURING EXECUTION
C
C
400   WRITE (*,401) DATAIN
401   FORMAT (1X, 'THE DIGITAL PORT HAS A VALUE OF ,I2 )
C
C     ... DIGITAL INPUT PORT WAS 4 .....

C     *** PASS FUNCTION VALUE TO TWO VARIABLES TOGETHER ***
C
      X = DIGIN ( DATAIN )
C
C     --- BOTH X AND DATAIN HAVE THE DIGITAL PORT DATA ----
C     THE VARIABLE DATAIN  MUST BE A NAME FOR  DATA  TO  BE
C     TRANSFERRED IN. UNKNOWN ERRORS WILL OCCUR IF A CONSTANT
C     IS USED.
C
      END
```

CNTMIn (MODE)

This function allows the user to read the selected counter in one of two modes,Latched or Non-latched. There are three counters available to the user and the selected counter is specified by "n", where n is 0, 1, 2. All three counters are completely independent. The CNTMIn function may be used in conditional statements as shown in the example. The MODE variable must be of the **INTEGER*2** type. The return data is also of the **INTEGER*2** type and the receiving variable must be type matched. The data range returned by the function is in the range of 0 to 65535 (0000 to #FFFF) 16 bits.

MODE = Selects one of two read modes.

0 = UN-Latched read on the fly (dynamic)

>= 1 = Latched, Data is latched prior to reading
This mode is active for any value except 0.

EXAMPLE:

```
C     **** DECLARE VARIABLE TYPES ****
C
      INTEGER*2   MODE,DATIN0,DATIN1,DATIN2,CNTMI0,CNTMI1,CNTMI2
      MODE = 1
C
C     ***** READ COUNTER 0 TO VARIABLE ******
C
      DATIN0 = CNTMI0 ( MODE )
C
C     ***** READ COUNTER 1 TO VARIABLE ******
C
      DATIN1 = CNTMI1 ( MODE )
C
C     ***** READ COUNTER 2 TO VARIABLE ******
C
      DATIN2 = CNTMI2 ( MODE )
C
C     ***** USE COUNTER 1 WITH CONDITIONAL STATEMENTS *****
C
      IF ( CNTMI1 (MODE) .GE. 1000 ) GOTO 400

      ...... CONTINUE PROGRAM, COUNTER IS LESS THAN 1000 ........
                     ................
                     ................

400   WRITE (*,401) CNTMI1(MODE)
401   FORMAT (1X, 'THE CURRENT VALUE OF THE COUNTER IS ',I5)
```

END

CNTMOn (MODE, DATOUT)

This function allows the user to load the selected counter with the value specified by the DATOUT integer value. There are three counters available to the user, where n = 0, for counter 0, 1 for counter 1, and 2 for counter 2. Each counter is independent from the others. The selected counter may be initialized (loaded) in one of six modes for user versatility. The counters may be programmed to be a divider, a programmable event counter, a programmable digital one shot and a programmable real time clock. The counters may be connected in almost any configuration at the 37 pin edge card connector, (refer to the DASH -16 manual for the various connections).

MODE = Selects the current operating mode for the counter. The mode value range is 0 to 5. any attempt to load a value less than 0 will load 0, and any value larger than 5 will load 5.

0 = **Output goes high on terminal count.** The output remains high until Re-loaded. The output will be set low upon execution and starts counting. If this mode is entered while the counter is counting the counter will stop until the new count value is loaded and then start a new count with the new values entered.

1 = **Programmable one-shot.** Output will go low on the count following the rising edge of the gate input. The output will go high on the terminal count. If a new value is loaded while the output is low it will not affect the duration of the one-shot pulse until the succeeding trigger.

The one-shot is retriggerable, hence the output will remain low for the full count after any rising edge of the gate input.

2 = **Rate Generator.** Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts specified by the DATOUT variable. Reloading will change the rate on the next count cycle. The counter will start counting at execution of this function.

The gate input, when low, will force the output high. When the gate input goes high, The counter will start from the initial count.

3 = **Square Wave Rate Generator.** Similar to mode 2 except that the output is a square wave (50% duty cycle). The counter will remain in the square wave state and at the rate programmed until reloaded. The frequency of the square-wave is defined as, 1,000,000 / DATOUT.

The gate input, when low, will force the output high. When the gate input goes high, The counter will start from the initial count.

4= **Software Triggered Strobe.** The output goes low upon execution of function and starts counting at the number value of DATOUT. On terminal count, the output will go low for one input clock period, then go high again. If the counter is reloaded between output pulses the present period is not affected, but the subsequent period will reflect the new value. The count will be inhibited while the gate input is low.

5 = **Hardware Triggered Strobe.** The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will go low until the full count after the rising edge of any trigger.

DATOUT = Data value to load counter with. The variable is expected to be INTEGER*2 type. The range of the variable for all modes is 2 to 65535 (16 bits). Any attempt to load a number less than 2 will automatically load the value 2. Any attempt to load a number larger than 65535 will then load MOD(2^16 - 1).

NOTE:

Counters 1 and 2 of the DASH-16 are Cascaded to make up a 32 bit counter. The counters 1 and 2 have a 1 MHz input clock and Counter 0 has a 100 KHz clock input. To use counters 1 and 2 as a rate timer (real time clock) then the output ticks will be defined by:

FREQUENCY = 10**6 / (CNTM1 * CNTM2)

where:

CNTM1 and CNTM2 are the data values loaded into the counter. All other modes are available to the user. Counter 0 as a

real time clock is defined by:
FREQUENCY = 10**5 / CNTM0 data
For more information on the counter timers on the DASH-16 refer to Chapter 5 of the DASH-16 manual.

EXAMPLE:

```
C     ***** SETUP VARIABLE TYPES ******
C
      INTEGER*2 DATO0, DATO1, DATO2, MODE1, MODE2, MODE3, MODE5
      INTEGER*2 BASADR
C
      BASADR = #300
      DATO0 = 1000
      DATO1 = 4
      DATO2 = 250
      MODE5 = 5
      MODE3 = 3
      MODE2 = 2
      MODE1 = 1
C
C     **** SETUP COUNTER 0 FOR EXTERNAL HARDWARE TRIGGER ****
C
      CALL CNTMO0 (MODE5, DATO0)
C
C     **** SETUP COUNTER 1 FOR DIVIDE BY N COUNTER ****
C
      CALL CNTMO1 ( MODE2, DATO1 )
C
C     **** SETUP COUNTER 2 FOR DIVIDE BY N COUNTER ****
C
      CALL CNTMO2 ( MODE2, DATO2 )
C
C     * Cascade Counter 1 and Counter 2 to generate a 1 KHz wave *
C
C     SETUP COUNTER 0 TO USE THE INTERNAL 100KHz ON BOARD
C     REFERENCE.  TO DO THIS OUTPUT A BYTE #02 TO THE COUNTER
C     CONTROL REGISTER AT BASE ADDRESS + 10
C
      CALL OUTB ( (BASADR+10),#02)
C
C     COUNTER 0 IS NOW INTERNALLY REFERENCED TO 100,000 Hz
C
      END
```

INPB (PORT)

This function allows the user to input data from a specified I/O port. The data transferred is in BYTE format (0 to 255). The variable PORT is a **INTEGER*2** type and has the full range of the 8086/8088 processor of MOD(2^16 - 1), (0 to 65535). This function may be used with conditional statements as shown in the examples. INPB performs the same function as the IN Byte instruction in assembly language. The data is transferred using MOD(2^8 - 1) format.

PORT = I/O Address in the range of MOD(2^16 - 1)
[0 to 65535].

EXAMPLE:

```
C     *** SETUP PORT VARIABLE ***
C
      INTEGER*2 PORT, PRTDAT, INPB
      PORT = #3F8
C
C     **** READ PORT ****
C
      PRTDAT = INPB(PORT)
C
C     **** READ PORT WITH CONSTANT AS VARIABLE ****
C
      PRTDAT = INPB (#3F8)
C
C     **** USE FUNCTION WITH CONDITIONAL STATEMENTS ****
C
      IF ( INPB( PORT ) .EQ. #80 ) GOTO 400
C
           ..... CONTINUE NOT EQUAL TO #80 .....
                     ...............
                     ...............
C
C     ---- IF PORT IS #80 EXECUTE THIS PROGRAM ----
C
400   WRITE (*,401) INPB(PORT)
401   FORMAT (1X, 'THE VALUE AT THE PORT IS ',I3)
      END
```

INPW (PORT)

This function allows the user to input data from a specified I/O port. The data transferred is in WORD format (0 to 65535). The variable PORT is a **INTEGER*2** type and has the full range of the 8086/8088 processor of MOD(2^16 - 1), (0 to 65535). This function may be used with conditional statements as shown in the examples. INPW performs the same function as the IN Word instruction in assembly language. The data in is Low Byte from PORT and High Byte form PORT+1. The data is transferred using MOD(2^16 - 1) format.

PORT = I/O Address in the range of MOD(2^16 - 1)
[0 to 65535].

EXAMPLE:

```
C     *** SETUP PORT VARIABLE ***
C
      INTEGER*2 PORT, PRTDAT, INPW
      PORT = #3F8
C
C     **** READ PORT ****
C
      PRTDAT = INPW(PORT)
C
C     **** READ PORT WITH CONSTANT AS VARIABLE ****
C
      PRTDAT = INPW (#3F8)
C
C     **** USE FUNCTION WITH CONDITIONAL STATEMENTS ****
C
      IF ( INPW( PORT ) .EQ. #8000 ) GOTO 400
C
            ..... CONTINUE NOT EQUAL TO #8000 .....
                        ...............
                        ...............
C
C     ---- IF PORT IS #8000 EXECUTE THIS PROGRAM ----
C
400   WRITE (*,401) INPW(PORT)
401   FORMAT (1X, 'THE VALUE AT THE PORT IS ',I5)
      END
```

OUTB (PORT, DATOUT)

This function allows the user to output data to a specified I/O port. The data transferred is in BYTE format (0 to 255). The variable PORT is a **INTEGER*2** type and has the full range of the 8086/8088 processor of MOD(2^16 - 1), (0 to 65535). This function may be used with conditional statements as shown in the examples. OUTB performs the same function as the OUT Byte instruction in assembly language. The data is transferred using MOD(2^8 - 1) format.

PORT = I/O Address in the range of MOD(2^16 - 1)
[0 to 65535].

DATOUT = Byte Data to output. The data is MOD (2^8 - 1).

EXAMPLE:

```
C     *** SETUP PORT VARIABLE ***
C
      INTEGER*2 PORT,DATOUT
      PORT = #3F8
      DATOUT = #F3
C
C     **** WRITE DATA TO PORT hex 3F8
C
      CALL OUTB (PORT, DATOUT)
C
C     **** WRITE PORT WITH CONSTANT AS VARIABLE ****
C     **** OUTPUT hex F3 TO PORT hex 3F8 ****
C
      CALL OUTB ( #3F8, #F3 )
      END
```

OUTW (PORT, DATOUT)

This function allows the user to output data to a specified I/O port. The data transferred is in WORD format (0 to 65535). The variable PORT is a **INTEGER*2** type and has the full range of the 8086/8088 processor of MOD(2^16 - 1), (0 to 65535). This function may be used with conditional statements as shown in the examples. OUTW performs the same function as the OUT Word instruction in assembly language. The output is Low Byte to PORT and High Byte to PORT+1. The data is transferred using MOD(2^16 - 1) format.

PORT = I/O Address in the range of MOD(2^16 - 1)
[0 to 65535].

DATOUT = Byte Data to output. The data is MOD (2^16 - 1).

EXAMPLE:

```
C     *** SETUP PORT VARIABLE ***
C
      INTEGER*2 PORT,DATOUT
      PORT = #3F8
      DATOUT = #10F3
C
C     **** WRITE DATA TO PORT hex 3F8
C
      CALL OUTW (PORT, DATOUT)
C
C     **** WRITE PORT WITH CONSTANT AS VARIABLE ****
C     **** OUTPUT hex 10F3 TO PORT hex 3F8 ****
C
      CALL OUTW ( #3F8, #10F3 )
      END
```

PEEKB (MSEG, MOFF)

This function allows the user to READ a byte from any memory location by defining the SEGMENT and OFFSET. The byte is written to location MSEG:MOFF. This function performs the same as the MOV ES:[OFFSET reg], reg in 8086/88 assembly language. The variables are INTEGER*2 type.

EXAMPLE:

```
C     **** DECLARE VARIABLES ****
C
      INTEGER*2 NUMB, MSEG, MOFF, PEEKB
C
C     *** DECLARE SELECTED ADDRESS ***
C
      MSEG = #F000
      MOFF = #0000
C
C     **** GET BYTE AT MEMORY LOCATION ****
C
      NUMB = PEEKB (MSEG, MOFF)
C
      WRITE (*, 100) NUMB
100   FORMAT (1X, 'THE MEMORY BYTE IS ', I5)
      END
```

PEEKW (MSEG, MOFF)

This function is the same as the PEEKB function except it returns a 16 bit integer word to the variable. All variables are INTEGER*2 type also.

EXAMPLE:

```
C     **** DECLARE VARIABLES ****
C
      INTEGER*2 NUMB, MSEG, MOFF, PEEKW
      MSEG = #F000
      MOFF = #0000
C
C     **** GET WORD AT MEMORY LOCATION ****
C
      NUMB = PEEKW (MSEG, MOFF)
C
      WRITE (*, 100) NUMB
100   FORMAT (1X, 'THE MEMORY WORD IS ', I7)
```

```
      END
```

POKEB (MSEG, MOFF, DATOB)

This function allows the user to write a byte any where in memory by defining the SEGMENT and OFFSET. The byte is written to location MSEG:MOFF. This function is the same as MOV ES:[OFFSET reg], reg. in 8088 assembly language. The variables are INTEGER*2 type.

EXAMPLE:

```
C     **** DECLARE VARIABLES ****
C
      INTEGER*2 NUMB, MSEG, MOFF
C
C     *** DECLARE SELECTED ADDRESS ***
C
      MSEG = #F000
      MOFF = #0000
      DATOB = #1A
C
C     **** WRITE BYTE AT MEMORY LOCATION ****
C
      CALL POKEB (MSEG, MOFF, DATOB)
      END
```

POKEW (MSEG, MOFF, DATOW)

This function is the same as the POKEB function except it writes a 16 bit integer word to the selected memory location. All variables are INTEGER*2 type also.

EXAMPLE:

```
C     **** DECLARE VARIABLES ****
C
      INTEGER*2 NUMB, MSEG, MOFF
      MSEG = #F000
      MOFF = #0000
      DATOW = #10A2
C
      CALL POKEW (MSEG, MOFF, DATOW)
      END
```

LOCATE (ROW, COL)

This subroutine allows the user to locate the cursor to a row, column location. The variables are expected to be INTEGER*2 type. If the Row, Col limits are exceeded the max limits of the screen will be set by default.

EXAMPLE:

```
C
C     ****** DECLARE VARIABLES ******
C
      INTEGER*2 ROW, COL
C
C
C     LOCATE THE CURSOR ON ROW 5, COLUMN 10
C
      ROW = 5
      COL = 10
C
      CALL LOCATE (ROW, COL)
C
C
C     LOCATE THE CURSOR ON COLUMN 3, ROW 19
C
      CALL LOCATE (19,3)
      END
```

CLRSCN (FG, BG)

This subroutine allows the user to clear the screen and set the Foreground and Background color. The FG,BG color variables are expected to be INTEGER*2 type. the color selection is shown below. The color selection in the background is limited to primary colors only.

COLOR TABLE FOR FG,BG

HEX	COLOR	HEX	COLOR
00	BLACK	08	GRAY
01	BLUE	09	LIGHT BLUE
02	GREEN	0A	LIGHT GREEN
03	CYAN	0B	LIGHT CYAN
04	RED	0C	LIGHT RED
05	MAGENTA	0D	LIGHT MAGENTA
06	BROWN	0E	YELLOW
07	WHITE	0F	HIGH INTENSITY WHITE

EXAMPLE:

```
C
C     **** CLEAR THE SCREEN SET BG = BLUE, FG = WHITE
C
      CALL CLRSCN (#07, #01)
      END
```

3.00 LIBRARY MEMORY MAP (GLOBALS)

The following is a memory map of the DAS16FOR.LIB and the associated GLOBAL Externals associated with each function.

Number	Size	Name	Segment Name	Date Version
1	195H	ADINIT	ADINIT_CODE	12 JAN 1985
2	10A8H	ADCONV	ADCONV_CODE	11 DEC 1986
3	A8H	DI6FIX	D16FIX_CODE	12 JAN 1985
4	128H	DMASTA	DMASTA_CODE	12 JAN 1985
5	109H	DMAOFF	DMAOFF_CODE	12 JAN 1985
6	C7H	CNTMI0	CNTMI0_CODE	12 JAN 1985
7	C7H	CNTMI1	CNTMI1_CODE	12 JAN 1985
8	BEH	CNTMI2	CNTMI2_CODE	12 JAN 1985
9	DEH	CNTMO0	CNTMO0_CODE	12 JAN 1985
10	DEH	CNTMO1	CNTMO1_CODE	12 JAN 1985
11	DCH	CNTMO2	CNTMO2_CODE	12 JAN 1985
12	13CH	DAOUT	DAOUT_CODE	12 JAN 1985
13	B8H	DIGIN	DIGIN_CODE	12 JAN 1985
14	B5H	DIGOUT	DIGOUT_CODE	12 JAN 1985
15	8FH	INPB	INPB_CODE	12 JAN 1985
16	8DH	INPW	INPW_CODE	12 JAN 1985
17	95H	OUTB	OUTB_CODE	12 JAN 1985
18	95H	OUTW	OUTW_CODE	12 JAN 1985
19	9EH	PEEKB	PEEKB_CODE	12 JAN 1985
20	9CH	PEEKW	PEEKW_CODE	12 JAN 1985
21	A4H	POKEB	POKEB_CODE	12 JAN 1985
22	A4H	POKEW	POKEW_CODE	12 JAN 1985
23	BCH	LOCATE	LOCATE_CODE	12 JAN 1985
24	B3H	CLRSCN	CLRSCN_CODE	12 JAN 1985
25	1CBH	DAS16EXT	DATA	12 JAN 1985

GLOBAL VARIABLES (EXTERNALS)

PUBLIC VARIABLES DEFINED BY: DAS16EXT

BASADR
BCNTR
BLKCNT

DABUSY
DMACNT
DMAFLG
DMAINT
DMAON
DMASF
DMAVEC
DMINXT

FNCH
INTLEV
KBDOFF
KBDSEG
MOD00

NOS00
STCH
STFNCH
SYSBSY
TMPCTL

ALL PUBLIC VARIABLES ARE 2 BYTES   [INTEGER*2]

4.00 SERVICE PERFORMANCE REPORT

This section is used for service of the DASH-16 Fortran library. If any problems occur during operation please write them down and mail the form to MetraByte Corporation. Be sure to enclose your return address and telephone number where you can be reached during the day.

LIBRARY: ______________________________

Date Purchased _______________________

Fortran Version Used ___________________________________

Description of Problem :

APPENDIX A   SAMPLE PROGRAM A/D MODE 5

```
C    *** METRABYTE CORP. DASH - 16 FORTRAN LIBRARY ***
C
C    This routine will collect 1024 points of channel 2 using
C    Counter 1 and 2 as an clock trigger and then display the
C    data in 5 columns of 20 rows each. This program is on the
C    disk supplied. The routine will display the current
C    conversion on the screen in the foreground while data is
C    transferred in the background.
C
C    ------ DECLARE VARIABLE TYPES FOR ROUTINE --------
C
     INTEGER*2 DATI0, SCH, FCH, MODE, RTNFLG, BASADR, DMALEV
     INTEGER*2 INTLEV, I, J, K, CNT0, CNT1, CNT2, NOS, NOC
     INTEGER*2 DMASTA, DI6STATUS, DMAOFF, D16FIX, CHANX, DATX
C
C    ------- DIMENSION  DATA ARRAY ----------------
C
     DIMENSION DATI0(1024)
C
C    ------- SET UP VARIABLE(S) DATA -------------
C
     MODE = 5
     SCH = 2
     FCH = 2
     NOS = 1024
     NOC = 0
     BASADR = #300
     DMALEV = 3
     INTLEV = 2
     RTNFLG = 0
     CNT0 = 0
C
C    ---- CLEAR THE SCREEN BACKGROUND = BLUE,FOREGROUND = WHITE
C
     CALL CLRSCN (7,1)
C
C    ----- INITIALIZE DASH-16 BOARD TO KNOWN STATE -----
C
     CALL ADINIT (BASADR, DMALEV, INTLEV, RTNFLG)
     IF (RTNFLG .EQ. 0) GOTO 20
     CNT0 =1
     CALL ERROR (RTNFLG, CNT0)
     GOTO 40
C
C   -- SETUP COUNTER 1 AND 2 FOR A TICK EVERY 10 MILLI-SEC --
C    TIME = (CNT1*CNT2)/10**6
```

```
C
20    CNT1 = 1000
      CNT2 = 100
C
C     ----- SETUP COUNTERS 1 AND 2 MODE 3 ----
C
      CALL CNTMO1 (3,CNT1)
      CALL CNTM02 (3,CNT2)
C
C     ---- TURN OFF DIGITAL OUTPUT PORT 0  IF USED FOR GATE ----
C
      CALL DIGOUT(0)
C
C     --- SETUP A/D FOR COLLECTION IN THE BACKGROUND (DMA) ---
C
      CALL ADCONV (MODE, SCH, FCH, NOS, DATI0(1), RTNFLG)
      IF (RTNFLG .EQ. 0) GOTO 22
      CNT0 = 2
      CALL ERROR (RTNFLG, CNT0)
      GOTO 40
C
C     ---- DATA IS BEING COLLECTED IN THE BACKGROUND WHILE WE
C     LOOK AT THE STATUS IN THE FOREGROUND AND PRINT IT ON THE
C     SCREEN
C

 22   CALL LOCATE (2,10)
      WRITE (*,'(A\)') ' IS THE NEXT CONVERSION NUMBER'
C
 24   NOC = DMASTA (D16STATUS)
      CALL LOCATE(2,2)
      WRITE (*,'(I5\)') NOC
      IF (NOC .LE. #3F8) GOTO 24
C
C     ---- SETUP TO DISPLAY DATA ON SCREEN ----
C
 30   CALL CLRSCN(7,0)
      CALL LOCATE (1,1)
      WRITE (*, '(A\)') ' POINT#  DATA'
      CALL LOCATE (1,18)
      WRITE (*, '(A\)') 'POINT#  DATA'
      CALL LOCATE (1,34)
      WRITE (*, '(A\)') 'POINT#  DATA'
      CALL LOCATE (1,50)
      WRITE (*, '(A\)') 'POINT#  DATA'
      CALL LOCATE (1,66)
      WRITE (*, '(A\)') 'POINT#  DATA'
      J = 1
 34   I = 2
      K = 1
```

```
35    IF (I .NE. 22) GOTO 36
      I = 2
      GOTO 35
36    CALL LOCATE (I,K)
      WRITE (*,'(I4\)') J
      CALL LOCATE (I, (K+8))
      DATX = D16FIX ( DATIO(J), CHANX )
      J = J+1
      IF (J .GE. 1025) GOTO 40
      I = I+1
      IF (I .NE. 22) GOTO 35
      K = K+16
      IF (K .GT. 70) GOTO 34
      GOTO 35
C
C     ------- END OF RUN -----
C
40    CALL LOCATE (23,1)
      STOP
      END
C
C
C     ------ SUBROUTINE FOR ERROR HANDLING ------
C
      SUBROUTINE ERROR (X,Y)
      INTEGER*2 X, Y
C
      CALL LOCATE (23,1)
      WRITE (*,51) X,Y
51    FORMAT (1X, 'ERROR IN DATA COLLECTION: RETURN FLG = ', 2I5)
      RETURN
      END
```