MICROSTAR LABORATORIES™

2265 116th Avenue N.E., Bellevue, WA 98004
Sales & Customer Support: (425) 453-2345
Finance & Administration: (425) 453-9489
Fax: (425) 453-3199
World Wide Web: http://www.mstarlabs.com/

ACCEL Driver for OS/2 2.x

Communication between a Data Acquisition Processor™ and a PC application is provided by interface software supplied by Microstar Laboratories. This interface software is a device driver called the ACCEL driver. The ACCEL driver is responsible for transfer of data between a PC and a Data Acquisition Processor.

The ACCEL driver establishes a link to the DAP through the OS/2 file system. ACOM.SYS installs a series of devices named ACCEL, ACCEL0, ACCEL1,...ACCEL20. Since DOS and OS/2 treat devices as files, a PC program can read from and write to any of the ACCEL devices as though they were files. Therefore, a PC program opens input files and output files with the device name ACCEL(x). Once the files are opened, the PC program can write data to the Data Acquisition Processor by writing to an output file and read data from the Data Acquisition Processor by reading from an input file.

The OS/2 ACCEL driver from Microstar Laboratories supports versions 2.0 and above of the OS/2 operating system.

Installation

The ACCEL driver is in the file ACOM.SYS. Copy this file from the release diskette to the root directory of the OS/2 boot drive. To install the driver, include a statement in the OS/2 file CONFIG.SYS. The statement syntax is as follows:

```
DEVICE=ACOM.SYS Ix DAP:Azzz Mu Pw
```

where

```
x     --  a hexadecimal digit specifies the interrupt vector
zzz   --  specifies the hexadecimal starting I/O address of
          the Data Acquisition Processor
u     --  is two-digit hexadecimal mode specification
w     --  is a hexadecimal number specifying the number of
          paragraphs of memory to reserve for PC communication
          pipes
```

The DAP model option can be repeated to support more than one Data Acquisition Processor in the PC. A detailed explanation of the above options can be found in each Data Acquisition Processor Manual. The following example works with a basic DAP 2400a™ system:

```
DEVICE=ACOM.SYS I3 DAP:A220 M00 P180
```

Once a device installation statement is placed in the file CONFIG.SYS, the ACCEL driver will be automatically installed when the system is booted.

Driver Initialization

Before using the driver, run the program ACOMINIT with a proper communication configuration file ACOM.DAT. (The communication configuration file is explained in the ACCEL driver chapter of the Data Acquisition Processor Systems Manual.)

Running ACOMINIT can be automated by placing a statement in the OS/2 file STARTUP.CMD. The statement is:

```
[PATH]ACOMINIT [PATH]ACOM.DAT
```

[PATH] is the path name of the directory where the two files are located.

If a RAM-based DAP is used in the system, STARTUP.CMD also should include the following line after the driver initialization statement to automate the initialization of the Data Acquisition Processor:

```
[PATH]DAPLINIT [PATH]D*.STD /RESET
```

where D*.STD is the downloadable operating system for the Data Acquisition Processor.

Using the OS/2 ACCEL Driver

The OS/2 driver supports DAPview™ and DAPview for Windows running in DOS compatibility boxes, and native 32-bit OS/2 applications running under OS/2. OS/2 applications must be written in the C programming language. Microstar Laboratories supplies two sets of C libraries for use with either Borland C++ for OS/2 or IBM C++ Tools. These files are included on the release diskette.

Compiling C/C++ programs

There are no special requirements involved in compiling C/C++ programs that will make calls to the driver. Function prototypes are provided in the file dapio.h. Somewhat more extensive driver support can be found in the file c_lib.c, which can be used as an include file. When linking, the library appropriate to the compiler model should be used (IBM_CDAP.LIB for IBM C++ and BC_CDAP.LIB for Borland C++). Applications that use multiple threads should also be linked to the multi-thread libraries from the compiler manufacturers: DDE4MBS for IBMC++ and C2MT for Borland C++

A typical compilation command line for IBM C/C++ Tools has the syntax:

```
ICC /DIBM32BIT /Gm+ DTERM.C IBM_CDAP.LIB
```

The macro definition /DIBM32BIT is required to switch to some IBM C/C++ Tools specific syntax to compile the Data Acquisition Processor example programs.

A typical compile/link sequence for Borland C++ for OS/2 has the syntax:

```
BCC -C -sm -DBC32BIT DTERM.C BC_CDAP.LIB
```

The macro definition /DBC32BIT is required to switch to some Borland C/C++ specific syntax to compile the Data Acquisition Processor example programs.

Examples

Three application examples are included on the release diskette. The same source files can be used to compile 32-bit applications using Borland C++ for OS/2 or the IBM C/C++ Tools compiler. The examples include:

1. DISPLAY.C — Reads data from the Data Acquisition Processor and writes to the screen.
2. DLOG.C — Reads data from the Data Acquisition Processor and writes to disk.
3. DTERM.C — Puts Data Acquisition Processor in terminal mode and allows interactive input.

DLOG.C Example

```
/*****************************************************************************
DLOG.C  - This program illustrates high speed disk logging using text stream I/O
to write commands to the DAP, and binary device I/O to read data from the DAP.
SetParam is used to set the time between readings in the DAPL command file.
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include "dapio.h"

#define BlockSize 256              /* The size of DAP to Disk data blocks*/

void main ()
   {
   short  Data[BlockSize];
   int     i,
        DapBinIn;
   FILE *DapTextOut,
        *DapTextIn,
        *logfile;
   char lname[20];

   printf ("Enter logging file name: ");
#ifdef IBM32BIT
   fflush(stdout);
#endif
   scanf ("%19s", lname);
   logfile = fopen (lname, "w");     /* Open the log file              */

   /* Open ACCEL1 (binary) as a binary input file used to get data from    */
   /* the DAP, and ACCEL0 (text) as a text output file and as a binary     */
   /* input file used to communicate with the DAP.                        */
   if (((DapBinIn   = open ("ACCEL1",O_RDONLY|O_BINARY)) == -1) ||
      ((DapTextOut = fopen("ACCEL0",OpenTextWrite)) == NULL) ||
      ((DapTextIn  = fopen("ACCEL0","rb")) == NULL))
      {
      printf("Error opening DAP device driver\n");
      exit(1);
      }

   BinaryMode(DapBinIn);            /* Set all files to DOS binary mode   */
   fBinaryMode(DapTextOut);
   fBinaryMode(DapTextIn);
   setvbuf(DapTextOut,NULL,_IONBF,0);/* Turn off output stream buffering   */

   WriteIoCtlStr(DapBinIn,"S,M00");  /* Set up ACCEL device driver mode    */

   fprintf(DapTextOut, "RESET\n");   /* Send a command to reset the DAP    */

   FlushDap(DapBinIn);              /* Flush old DAP data              */
   fFlushDap(DapTextIn);

   /* Configure the DAP using the DAPL command file LOG.DAP               */
   /* Assign 50 to parameter 1 which is used to set the TIME parameter    */
   SetParam(1,"50");
   if (fConfigDap(DapTextOut,"LOG.DAP") >= 200)
      {
      printf("Error while configuring DAP\n");
      exit(2);
      }
```

```
for (i=1; i<=100; i++)          /* Read and log blocks of DAP data    */
   {
   GetDapBuf(Data, sizeof(short), BlockSize, DapBinIn);
   fwrite(Data, sizeof(short), BlockSize, logfile);
   }

fclose(logfile);
fprintf(DapTextOut, "STOP\n");    /* Send a command to stop the DAP     */
WriteIoCtlStr(DapBinIn,"R");      /* Restore ACCEL device driver mode   */
fclose(DapTextOut);               /* Close files                        */
fclose(DapTextIn);
close(DapBinIn);
}
```

Multi-tasking Under OS/2

More than one Data Acquisition Processor application can run in the OS/2 multi-tasking operating system, and multiple threads can run independently within a single process. However, several guidelines should be observed to ensure correct operation of the data acquisition system.

The physical ACCEL driver is logically installed as numbered ACCEL devices with the device names ACCEL0 through ACCEL20, each of which is associated with a corresponding numbered communication pipe if that communication pipe has been defined in the file ACOM.DAT. Each ACCEL device can only be opened by one OS/2 process. If two different processes attempt to open the same ACCEL device, the driver will report and error and the second process will be unable to run.

The directed ACCEL device (named ACCEL) is also installed. It can be used for I/O to and from any of the communication pipes. The currently active pipe is set by C library function calls. Multitasking is not allowed if the directed ACCEL device is used. Since the directed ACCEL device has access to all numbered pipes, any process that opens the directed ACCEL device has the potential to interact with any other process that opens a numbered ACCEL device. To eliminate such conflicts, only one process will be allowed to open ACCEL devices if the directed ACCEL is one of the devices opened.

The device protection mechanisms described above not allow more than one application to have an open handle to the PC com pipe connected to SYSIN. Since DAPL commands can only be sent to the DAP over SYSIN, only one application can control the DAP at a time. Other applications can affect the DAP's operation by changing modes, which is accomplished by IOCTL calls to the driver (see the ACCEL driver section of the Systems Manual). C programs that access the DAP are often critically dependent on a particular mode. Thus no mode changes should occur when multitasking DAP-related applications. Because the driver does not enforce this rule, programmers should be especially aware of the consequences of mode changes.

ACCEL Driver File Handles

The ACCEL Driver does not allow its file handles to be duplicated. If the ACCEL file handles are opened in the current OS/2 session, and the current OS/2 session tries to spawn another OS/2 or DOS session, a system error will occur. This is because the operating system tries to duplicate the ACCEL file handles when it spawns another session, and since we do not allow this, a system error occurs. An example error message would be ‘The ACCEL0 device is not functioning’.

You need to use the *O_NOINHERIT* option to open the ACCEL file handles if the OS/2 session with the ACCEL file handles has to spawn other sessions. The example C program below shows how to do this. The example uses the C function *system* to spawn a DOS session and execute a DOS batch file. It also reads text and binary data from the Data Acquisition Processor.

Please note that this example was written for the Borland C/C++ compiler for OS/2. It may or may not work with the IBM C/C++ Tools compiler.

Below are the necessary files to run the NINHERIT.C example.

1. NINHERIT.C

```
/*****************************************************************************
NINHERIT.C  -  This C example demonstrates how to open ACCEL file
handles using the O_NOINHERIT option.  It uses system to spawn a DOS
session and run a DOS batch file.  It also shows how to read text and
binary data from the Data Acquisition Processor.

_open is obsolete and should be replaced with _rtl_open .  The Borland
C/C++ compiler version 2.0 for OS/2 did not include the library file
that supports _rtl_open , which made linking impossible.  For this
reason, _open is used in this example until a newer version of the
Borland C++ compiler supports _rtl_open .
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include "dapio.h"

#define BlockSize 100

void main ()
{
        short Data[BlockSize], i;
        char textmsg[256];

        int iDapBinIn, iDapTextIn, iDapTextOut;
        FILE *DapBinIn, *DapTextIn, *DapTextOut;

        if      (((iDapBinIn   = _open("ACCEL1",O_RDONLY|O_BINARY|O_NOINHERIT)) < 0) ||
                 ((iDapTextIn  = _open("ACCEL0",O_RDONLY|O_BINARY|O_NOINHERIT)) < 0) ||
                 ((iDapTextOut = _open("ACCEL0",O_WRONLY|O_TEXT|O_NOINHERIT))   < 0))
        {
                printf("Error opening ACCEL handles.\n");
                exit(1);
        }

        if      (((DapBinIn   = fdopen(iDapBinIn,  "rb")) == NULL) ||
                 ((DapTextIn  = fdopen(iDapTextIn, "rb")) == NULL) ||
                 ((DapTextOut = fdopen(iDapTextOut,"wt")) == NULL))
        {
                printf("Error associating streams with ACCEL handles.\n");
                exit(2);
        }

        fBinaryMode(DapBinIn);                                          /* Set all files to DOS binary mode. */
        fBinaryMode(DapTextOut);
        fBinaryMode(DapTextIn);

        setvbuf(DapTextOut, NULL,_IONBF, 0);            /* Turn off output stream buffering. */

        fWriteIoCtlStr(DapBinIn, "S,M00");              /* Set up ACCEL device driver mode. */

        fprintf(DapTextOut, "RESET\r\n");                                       /* Send a command to reset the DAP. */

        fFlushDap(DapBinIn);                                                            /* Flush old DAP data. */
        fFlushDap(DapTextIn);

        /* Configure the DAP using the DAPL command file NINHERIT.DAP. */
        if (fConfigDap(DapTextOut, "NINHERIT.DAP") >= 200)
        {
                printf("Error while configuring DAP.\n");
                exit(3);
        }

        fprintf(DapTextOut, "HELLO\r\n");               /* Send hello to the DAP. */

        while(fGetDapAvail(DapBinIn)==0);                                       /* Wait until some data are available. */
        printf("Reading DAP data...\n");
```

```
        fGetDapBuf(Data, sizeof(short), BlockSize, DapBinIn);
        for(i=0; i<BlockSize; i++)
                printf("%s %hd %s %hd\n", "Sample", i+1, "=", Data[i]);

        system("start /c /win NINHERIT.BAT");                   /* Spawn a DOS session and run a DOS

          batch file. */

        /* Scan DapTextIn for messages. */
        printf("\nBelow is a text message from the DAP, in response to the DAPL
        command HELLO:\n");

        if(fGetDapAvail(DapTextIn)>0)
        {
                fgets(textmsg, sizeof(textmsg), DapTextIn);
                printf("%s", textmsg);
        }

        fprintf(DapTextOut, "STOP\r\n");                                    /* Send a command to stop the DAP. */

        /* This stops all active tasks. */

        fWriteIoCtlStr(DapBinIn,"R");                                       /* Restore ACCEL device driver mode. */

        fclose(DapTextOut);                                                 /* Close all files. */
        fclose(DapTextIn);
        fclose(DapBinIn);
}
```

2. NINHERIT.DAP

```
;*************************************************************************
;NINHERIT.DAP  -  This DAPL file samples one input channel and sends
;the data to the PC through the binary communication pipe.
;*************************************************************************

RESET
IDEF A 1
      SET IP0 S0
      TIME 1000
      COUNT 100
      END
PDEF B
      MERGE(IP0,$BINOUT)
      END
START
```

3. NINHERIT.BAT

```
@echo off
rem ************************************************************************
rem NINHERIT.BAT  -  This DOS batch file is run by the OS/2 session.
rem ************************************************************************

echo The OS/2 session spawned a DOS session to
echo run this batch file, NINHERIT.BAT.
pause
```

Text or Binary File Streams

When a file is opened in text mode, line feed (\n) is converted to carriage return and line feed (\r\n) by the operating system. However, when a file is opened in binary mode, that conversion does not take place and \r\n should be used instead.

In DLOG.C, DapTextOut (called DapOut in DTERM.C and C_LIB.C) was opened as a text file. However, in NINHERIT.C, DapTextOut was opened as a binary file. You will notice that DLOG.C uses \n but NINHERIT.C uses \r\n when used with DapTextOut.

Example in DLOG.C

```
fprintf(DapTextOut, “RESET\n“);
```

Example in NINHERIT.C

```
fprintf(DapTextOut, “RESET\r\n“);
```