

Micro Fiche Scan

Name of device(s) tested:

XXDP

Test description:

XXDP V2 DRVR PROGR GD

MAINDEC Number or Package Identifier (after SEP 1977):

CHQDPA0

Fiche Document Part Number:

AH-FG10A-MC

Fiche preparation date unknown, using copyright year:

1984

Image resolution:

8-bit gray levels, max. quality for archiving

COPYRIGHT (C) 1984 by d|i|g|i|t|a|l

IDENTIFICATION

PRODUCT NAME: AC-U036A-MC
PRODUCT NAME: CHQDPAQ XXDPV2 Drvr Progr Gd
PRODUCT DATE: 8 Oct 1984
MAINTAINER: Low End Diagnostic Engineering

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

Copyright (C): 1984 by Digital Equipment Corporation

The following are trademarks of Digital Equipment Corporation:

Digital, PDP, UNIBUS, MASSBUS, DEC, DECUS, DECTape, DEC/X11

XXDP V2
Driver Programmer's Guide

Manual Revision: 0.1

XXDP Version: 2.0

Maintained by: MSD Diagnostic Engineering

Revision History:

Revision	Date	By	Description
0.0	1-Jun-84	DAL	Original Document
0.1	8-Oct-84	LSP	Reformat

Table of Contents

	Page -----
1.0 Introduction	4
2.0 Device Driver Layout	5
3.0 Device Driver Functions	11
4.0 Writing a Driver	15
5.0 Device Driver Characteristics	17
6.0 Glossary	19
7.0 Bibliography	19

Appendices

- Appendix: A - Driver Example
- Appendix: B - Assembly and Linking Instructions
- Appendix: C - Driver Equates
- Appendix: D - Device Type Codes

1.0 Introduction

This document is intended as a guide to those who need to understand and/or write device drivers for the XXDP+ V2 system. Section 1.0 below describes the basic differences between V1 and V2 drivers. Section 2.0 outlines the physical layout of the driver. Section 3.0 describes the functions performed by drivers while section 4.0 offers advice to those intending to maintain or write a device driver themselves.

Throughout this document there are many references to the mnemonics of the file structure. These are listed in the glossary for convenience. A description of the file structure may be found in the file structure document listed in the bibliography.

1.1 Differences between V1 and V2 Drivers

One major purpose of XXDP+ V2 is to simplify the maintenance of XXDP components. A facet of this simplification is to make drivers as uniform as possible. To this end:

- a) Functionality which seemed more file-oriented than device-oriented (e.g. file search) was migrated to a front-end, which is now incorporated in a version of UPD2 and other utilities.
- b) Read-only and Read-write functionality was recombined so that a single driver may be used both by the Monitor and by utilities.
- c) Some functional aspects of individual drivers were changed. For instance, most drivers will now support two units (previously a different copy was needed for each unit).
- d) The layout of all drivers was made as uniform as possible.
- e) Disk organization has been made uniform (MFD 'variety #1' has been retired).
- f) Some functional aspects of the Utilities were changed. UPD2 will no longer permit an Image copy between devices with differing sizes, and will not copy the Monitor during a File copy.

1.2 Compatibility

Compatibility between V2 and V1 has been maintained, with the following exceptions:

- 1) The V1 DL and DM disk layout did not allow for a 32k Monitor. If the V2 Monitor is installed on a V1 medium, the first file (or two) after the monitor area will be corrupted.
- 2) The MFD variety #1 has been retired for the DB, DD, DU and DY drivers. V2 drivers may be used to read or write V1 media. V1 drivers may be used to read V2 media, but not to write. (Except in the case: V1 MS drivers will not read V2 MS tapes.)
- 3) V2 media will have the octal constant 1002 at octal displacement 14 (the old MFD2 pointer) in the MFD. V1 media will have some other value. The MFD is not currently read by most drivers, so this fact is not used.
- 4) The V1 MM and MS tape layouts each had two Monitors at the tape beginning, selected according to what device was being booted. The V2 layouts have only one Monitor as the first file on the tape.

2.0 Device Driver Layout

This section describes the lexical structure of XXDP Version 2 device drivers. The requisite components are outlined below with descriptions as to their functions and usage. Definitions of terms relating to file structure may be found in (AC-S866A-M0) CHQFSA0 XXDP. File Structure Document.

2.1 Driver Revision History

This section contains a brief history of attributed source code revisions, as is standard for DEC software.

2.2 Symbolic Equates

2.2.1 Device-Independent Equates

This section contains definitions for data structure offsets and other equates which are more or less common to all drivers.

1) DIRBLK Offsets

These equates describe the DIRBLK structure in the driver, discussed below. The DIRBLK contains a description of the (disk) layout.

2) DDB Equates

These equates describe the 'Device Descriptor Block' (DDB), a data structure which is found in the utilities, and a subset of which is found in the Monitor. The DDB provides the driver's data interface. The driver's Parameter Table will overlay or be copied to the DDB.

3) Device Command Codes

These equates are the command codes, issued by a utility or the monitor, to which the driver responds. Some command codes, e.g. WRITE\$, are used by all drivers. Others may be specific to device type (e.g. bad-blocking) or to the device itself (e.g. RFS\$FN- reformat RX02 single density).

4) Parameter Table Equates

When the driver is loaded by a utility, its parameter table is copied into the DDB. These equates are thus actually DDB offsets.

5) Device Returned Status Byte

These equates describe the meaning of the bits in the above-mentioned DVSB byte. They concern disk density and tape drive status.

2.2.2 Device-Dependent Equates

These are equates particular to the device and driver code.

1) Program Equates

These equates are typically mnemonics (e.g. LF or CR) used for convenience in the code.

2) Device Equates

These equates describe internal device codes, status words, commands, and packet formats.

2.3 Data Structures

2.3.1 Device Parameter Table

This data structure begins the driver's actual code. When the the Monitor is CREATED by the UPDATE utility the driver is appended to the end of the monitor and this table overlays the Monitor's DDB. When the driver is loaded by a utility, this table is copied into the utility's DDB, addresses being relocated appropriately. From this time on, the table is referenced largely through this DDB copy; the driver's copy is used only by the driver's INIT routine in anticipation of the next load. All driver routines assume that R5 points to the command register entry in the DDB.

(Note: in order to save space, some of the parameters have been given INITIAL values and functions which are not related to their functions during execution.)

A Parameter Table Example is:

```
-----
PARAM:  DISPATCH      ;DISPATCH ROUTINE
        .WORD  "DZ      ;DRIVER NAME
        .BYTE  BBSUP$   ;DEVICE CODE
        .BYTE  44       ;RETURNED STATUS (INITIAL DEVICE
                        ;                               TYPE)
        .WORD  BCODE    ;BOOT CODE OFFSET

UNIT:   .BYTE  0        ;UNIT #          (INITIAL REV #)
ERRB:   .BYTE  0        ;ERROR STATUS   (INITIAL PATCH #)
CMDREG: 174400        ;COMMAND REGISTER ADDR
WCOUNT: 0             ;WORD COUNT
BUSADR: 0             ;BUS ADDRESS
BLOCK:  0             ;BLOCK NUMBER
CMD:    0             ;COMMAND
DIRPTR: DIRBLK       ;POINTS TO 1ST DIR BLOCK.
ASNAME: 0             ;FOR MONITOR COMPATIBILITY
PAREND:
```

1) Dispatch Routine Address

This entry is the address of the dispatch routine, which determines which driver routines to invoke. All driver services are provided through this entry.

2) Driver Name

This entry is the device's two byte mnemonic name.

3) Device Code

This static byte is used to indicate that the device has special features of interest to utilities. Current flags are:

BBSUP\$ - Device provides bad block support.
NODIR\$ - Not a directory device
TAPED\$ - Tape device
REFDN\$ - Supports single/double density reformat.
MULUN\$ - Driver supports 2 units/driver
NOREN\$ - Device does not support file rename.
FLOAD\$ - Device may have floating address.

4) Device Status

This byte is returned by some drivers in response to inquiries concerning disk density or tape status. Current flags are:

DDDEN\$ - Disk is double density
BOTTP\$ - Tape is at physical bot
TMKTP\$ - Tape is at tape mark
EOTTP\$ - Tape is at physical eot

(The INITIAL value of this byte communicates a device type code to the Monitor immediately after the driver is loaded. See appendix D.)

5) Boot Code Offset

This entry contains the displacement to the boot code, i.e. to the end of driver code. This is used by the Monitor and does not further concern the driver itself.

6) UNIT

This byte entry communicates the device unit # to the driver. This is commonly addressed as XDN(R5).
(The INITIAL value of this byte communicates the version number of the driver.)

7) ERRB

This byte entry is used by the driver to communicate errors and (sometimes) attention conditions. It is tested immediately prior to driver exit (as XER(R5)).
(The INITIAL value of this byte communicates the patch number of this driver.)

8) CMDREG

This is the address of the primary device command register. It is the focus of the DDB and is used by the driver to access all device registers.

9) WCOUNT, BUSADR, BLOCK

These entries are used to communicate to the driver, the count, address, and block number of a transfer command.

10) COMD

This entry contains the coded command to be performed by the driver. This code is interpreted in the driver's dispatch routine.

11) DIRPTR

This entry points to the driver data structure DIRBLK, a table which describes the physical layout of a disk. This pointer is the only exception to the rule that local entries in this table (as opposed to their copies in the DDB) are not used. The driver's INIT routine may toggle this pointer for some "two-unit" drivers to point to an alternate DIRBLK structure to be active on the next load. This feature permits one driver to be used with two units with differing densities, etc.

2.3.2 DIRBLK

This data structure communicates particulars of the device's physical layout. Its first several entries mirror the structure of a variety #2 MFD, which is now used for non-bad-blocking devices as well. Note that for non-bad-blocking devices, the data contained in DIRBLK is constant and the MFD need never be actually read. For some drivers which support two units, DIRBLK will be replicated, and DIRPTR will be toggled back and forth by the driver's INIT routine.

2.3.3 Local data

This section contains data structures used internally by the driver to store state information, construct packets, etc. Some unit-dependent local data may be appended to DIRBLK to take advantage of DIRBLK switching for two-unit drivers.

2.3.4 Error Messages

This section contains the error messages printed by the driver. The utilities may append information to such messages, e.g. if the driver prints "RD ERR", the utility will note the error through the error byte XER(R5), and may append, for example, "IN INPUT DIRECTORY".

2.4 Executable Code

2.4.1 DISPATCH Routine

The dispatch routine receives control from the utility or monitor, examines the command code in the DDB, and gives control to subordinate routines. Dispatch may, in addition, perform code sequences common to its subordinates or indeed perform some simple commands. Just prior to exit, the dispatch routine tests the error byte XER(R5) so that the calling utility may make an immediate branch on error. At present, some dispatches are "test and call" and some table driven. In drivers with more than 4 such tests, a table driven approach may save space.

2.4.2 INIT Routine

The init routine receives control from dispatch. Its primary function is to perform any physical initialization and to set local DIRBLK variables to reflect unit characteristics. It is assumed to have been called immediately after the driver is loaded. Init may also perform auxillary functions, such as determining device density.

2.4.3 DRIVER Routine

The driver routine receives control from dispatch. It commonly handles I/O transfers. In many cases, the code in this routine is largely unchanged from that in V1.

2.4.4 Auxillary Routines

These routines are called by DISPATCH, INIT and DRIVER.

3.0 Device Drivers Functions

3.1 All Drivers

There is a minimal set of functions which all drivers are expected to perform:

INIT\$

This function is invoked once per device-unit, either after the Monitor has been loaded or immediately after a utility 'loads' a driver. Note that if a utility finds the requested driver to be already present, it will not load a fresh copy. Before INIT\$ is invoked, parameter table information has been copied (or in the case of the Monitor, overlaid) on to the DDB; in particular DIRPTR has been converted from relative to absolute address (but only on a fresh load).

Tasks to be performed at this time include device initialization (e.g. DU performs an initialization sequence at this time when the value of a local variable signifies that it is a fresh 'load') and initialization of local variables. Disk drivers which support bad-blocking use this occasion to read the disk MFD and set DIRBLK variables accordingly. Some drivers which support two units with differing characteristics (e.g. density) will toggle the (local) pointer DIRPTR at this time so that on the next 'load', a different DIRBLK will be used.

You will see that, in those drivers which have a GTMFD1 routine to read the MFD, a DIRBLK flag XXMFID is checked before any disk read is done. This flag is raised by the driver loading routine in the utility when a ZERO directive is in progress - in order to avoid reading junk from a disk which is about to be cleared. The DIRBLK structure is updated by the utility during the ZERO execution.

RES\$FN

This function is invoked by the Monitor to read some blocks from the Monitor image, presumably after possible corruption.

At this time the code relocates the requested block number by the starting Monitor block number. The code may assume that this entry in DIRBLK is either a constant or has been updated during INIT\$ processing.

READ\$

This function is used by all drivers except LP:.
It is invoked by the Monitor or the utility to read
a block or series of blocks from the device. The word
count, buffer address and starting block number (for
direct access devices) are found in the DDB.

It is the driver's function to convert the word count
and block numbers if necessary, to initiate the transfer,
and to wait until successful completion. If an error is
detected, the driver may try to effect recovery (e.g.
several disk drivers now have ECC correction routines).
If recovery is impossible, failure is communicated by
setting the XER byte in the DDB to a non-zero value.

WRITE\$

This function is used by all drivers. All comments
concerning READ\$ above are applicable here.

3.2 Disk Drivers

Disk devices are all directory structured. This is signalled to the utility by having a positive first entry in the DIRBLK table. A disk driver may have functions in addition to those above:

RED\$FN

This function requests the read of an absolute cylinder/track/sector from a bad-blocking device. It is invoked by the ZERO command execution in UPD2. UPD2 places the cylinder, track and sector addresses of the bad-block file (determined from DIRBLK) into the DDB and issues the call.

CMP\$FN

The format of the bad-block file is a list of cylinder/track/sectors. The ZERO routine in UPD2 issues a CMP\$FN to convert these to block numbers, which it uses to set the appropriate bit-maps.

DEN\$FN

The ZERO routine in UPD2 needs to know the disk density to find the correct location of the bad-block file.

The driver returns a flag in the DDB status byte DVSB.

0 = single density
1 = double density

RFS\$FN,RFD\$FN

The DY driver performs hardware re-formatting of a disk to single or double density (as communicated to UPD2 through the ZERO command).

3.3 Tape Drivers

Drivers for tape devices (communicated via the device code byte in the DDB and by a negative first word in DIRBLK) provide a variety of functions not needed for disk devices. Tapes are not directory devices - every file is preceded by a header which contains the file name. The logical end of tape is a double EOF. In addition to those functions listed as common to all drivers above:

PRE\$TP

This function is invoked to set up the tape controller for subsequent commands.

REW\$TP

This function is called to rewind the tape.

SPR\$TP

This function is called to backspace the tape.

WHD\$TP

This function is called to write a 7 word header.

RHD\$TP

This function is called to read a header.

SEF\$TP

This function is invoked to skip to an EOF, i.e. to skip the remainder of a file.

WEF\$TP

This function is called to write an EOF on tape.

SET\$TP

This function is called to skip to the logical end of tape, i.e. after all files.

STA\$TP

This function is invoked to return the tape status (at BOT, TMK, physical EOT) through the device status byte in the DDB. The two existing tape drivers, MM and MS approach this differently. MM backspaces the tape and then forward spaces. If BOT was detected during the backspace, this is returned as status. Otherwise the status detected during the forward space is returned. The MS driver interrogates the controller in real time.

4.0 Writing a Driver

The best approach to writing a driver is to model it on existing ones. The drivers that presently exist provide a wide variety from which to choose, and are briefly characterized along several dimensions at the end of this section. Some points to note:

- 1) Much of the driver preamble is device-independent and may be copied wholesale. Look at the preamble of UPD2 to determine the symbolic command codes etc. with which the utilities and drivers communicate.
- 2) The device-dependent components of the preamble follow informal conventions, e.g. control register names are often similar from device to device. You may be able to copy this, with minor changes, from some driver with a similar communications structure.
- 3) The parameter tables of all drivers are quite similar.
- 4) The DIRBLK specifies the physical layout of a disk device. Be careful how you lay out a disk structure - do not lock yourself into a structure which cannot be easily expanded to meet similar but larger devices. For example, you might want to put the Monitor image towards the beginning of the disk, before the UFD and Bitmaps, so that the bootstrap routine doesn't have to contend with these areas as they change from device to device.

An example of a good structure might be:

Block	Purpose
0	Secondary bootstrap
1	MFD1
3	Start of Monitor image
35.	First UFD block
35. * N	First bit map
35. * N * M	# of blocks to preallocate

Remember that, even though they are linked, UFD and bit map space are allocated contiguously by UPD2 at device ZEROing. It is, in fact, this contiguity which results in the possibility that the actual parameters may differ among bad-blocking devices.

- 5) The DDB error byte ERR(R5) is used to communicate failure. The driver must test this byte immediately before exiting. Note that the polarity of this device is used to communicate different flavors of failure: e.g. -1 often means 'device full'.

- 6) If you plan to have your driver support 2 disparate devices at the same time (e.g. bad-blocking devices are disparate because the actual location of some things may change. There is a limit to this: the boot routine may assume a constant location for the Monitor image), you may want to toggle between two DIRBLK's. Be careful, in this case, to remember that the parameter table actually overlays the DDB when the driver is linked with the Monitor; toggle before any changes are made to DIRBLK.
- 7) The DRIVER routine in many drivers disambiguates some of the commands. This is due to historical reasons and commonality of some code.
- 8) Driver code must be location-independent. In particular, this means that if addresses of local data are manipulated, they must be calculated dynamically rather than by the linker. E.g.

```
MOV    #TABLE,RO    ; will not get the address of
                    ; TABLE
```

but

```
MOV    PC,RO
ADD    #TABLE-.,RO  ; will work
```

- 9) All code must be processor independent.
- 10) The disk layout (reflected in DIRBLK) of some bad-blocking devices depends on the medium density. When a driver is 'loaded' as a result of a ZERO command, the MFD refreshed indicator in the DIRBLK is set by UPD2 prior to invoking the INIT function. This is tested in the driver's GTMFD1 routine to bypass an MFD read (the MFD may be junk). The UPD2 ZERO command will issue a DEN\$FN to the driver to determine the disk density, and will compute the bad-block file and bad-block dependent attributes (first UFD, bitmap, and Monitor) accordingly. It will not, however, set up the remaining density-dependent DIRBLK entries: this should be done by the driver's INIT code with consideration that the MFD might not be read.
- 11) The MFD for all devices is written by UPD2 during a ZERO command, and, for bad-blocking devices, must be referenced (because it contains variable information) by the driver during an INIT function to update the DIRBLK. The variables to be updated are starting UFD, Monitor, and bitmap block numbers. Except for this reference, the driver need not concern itself with the MFD variety or structure.

5.0 Device Driver Characteristics

DB - RJP04,5,6

Type - Disk
Bad-blocking - No
Error-recovery - ECC correction, retry
Communications - Device registers
DIRBLK - Constant
Two units/driver - Yes
Dispatch - Table

DD - TUS8

Type - Disk (directory structured tape)
Bad-blocking - No
Error-recovery - Retry
Communications - Packet
DIRBLK - Constant
Two units/driver - Yes
Dispatch - Table

DL - RL01,02

Type - Disk
Bad-blocking - Yes
Error-recovery - Retry
Communications - Device Registers
DIRBLK - Variable according to bad-blocking
and density.
Two units/driver - Yes
Dispatch - Table

DM - RK06,7

Type - Disk
Bad-blocking - Yes
Error-recovery - ECC correction, retry
Communications - Device Registers
DIRBLK - Variable according to bad-blocking
Two units/driver - Yes
Dispatch - Table

DR - RM02,03

Type - Disk
Bad-blocking - Yes
Error-recovery - ECC correction, retry
Communications - Device Registers
DIRBLK - Variable according to bad-blocking
Two units/driver - Yes
Dispatch - Table

DU - UDA 50,RD/RX

Type - Disk
 Bad-blocking - Transparent to driver
 Error-recovery - "
 Communications - MSCP
 DIRBLK - Variable according to drive capacity
 Two units/driver - Yes
 Dispatch - Test and call

DY - RX02,01 (does not boot RX01)

Type - Disk
 Bad-blocking - No
 Error-recovery - Retry
 Communications - Device Registers
 DIRBLK - Variable according to RX01/02
 Two units/driver - Yes
 Dispatch - Table

LP - Line printer

Type - Line printer
 Bad-blocking - Huh?
 Error-recovery - "
 Communications - Device registers
 DIRBLK - Constant 0
 Two units/driver - No
 Dispatch - Test and call

MM - TM02

Type - Tape
 Bad-blocking - "
 Error-recovery - Retry
 Communications - Device registers
 DIRBLK - Constant -1
 Two units/driver - Yes
 Dispatch - Table

MS - TS04/TS11

Type - Tape
 Bad-blocking - "
 Error-recovery - Retry
 Communications - Packet
 DIRBLK - Constant -1
 Two units/driver - Yes
 Dispatch - Table

6.0 GLOSSARY

- IRG - Interrecord gap. The gap that is written between records on magtape.
- MFD - Master File Directory
- RAD-50 - RADIX-50. A method of encoding 3 ASCII characters into one 16 bit word.
- UFD - User File Directory.
- UIC - User Identification code.

7.0 Bibliography

- XXDP- /SUPR USE MAN, CHQUS??. AC-F348F-MC, current
- XXDP- FILE STRUCT DOC, CHQFSA0, AC-S866A-MO, April, 1981

Appedicies

Appendix A - Driver and Boot Example

The following is an example of a working driver (DB:), edited slightly to explicate structure.

```

.NLIST CND
.TITLE RJP04,5,6 - XXDP+ V2 DRIVER
.SBTTL DRIVER REVISION HISTORY
-----
: REV   DATE           CHANGE
-----
: 1.0   31-JUL-78      INITIAL ISSUE
: 1.1   17-NOV-78      MAKE COMPATABLE WITH BIG DRVCOM
:
: 2.0   11-AUG-80      XXDP+ V1.1 COMPATIBLE
:                               REMOVED READ-ONLY CODE
:                               ADDED XER(R5) AS RESULT STATUS
:                               ADDED INIT ROUTINE
:                               REMOVED CLEAR MAPS ROUTINE
:
:       21-FEB-84      CHANGE FOR V2, INCLUDING ECC CORRECT
:       06-MAR-84      TWO UNITS/DRIVER - GOT RID OF GTMFD1
:       18-MAR-84      TABLE DRIVEN DISPATCH
:       25-APR-84      INITIALIZE RETURNED STATUS BYTE
-----

```

.PAGE

```

.NLIST ME,CND
.NLIST MC
.LIST  MEB

```

.SBTTL DEVICE-INDEPENDENT EQUATES

: DIRBLK OFFSETS

```

XDIR    =0      ;1ST DIR BLOCK.
XDIRN   =2      ;# OF DIR BLOCKS.
XMP      =4      ;1MAP BLOCK #.
XMPN    =6      ;# OF MAP BLOCKS.
XMFD1   =10     ;MFD1 BLOCK #.
XVERS   =12     ;XXDP VERSION # (1002 = VERSION 2)
XMXBK   =14     ;MAX BLOCKS WORD.
RSBK    =16     ;# OF BLOCKS TO RESERVE.
ITLVE   =20     ;INTERLEAVE FACTOR.
BOTBK   =22     ;BOOT BLOCK.
MNBK    =24     ;MONITOR CORE IMAGE BLOCK.
XMFID   =26     ;MFD REFRESHED INDICATOR.

```

```

:-----
: DEVICE DESCRIPTOR BLOCK (DDB) EQUATES
: DDB OFFSETS FOR R/W DRIVER
: DDB OFFSETS FOR MONITOR ARE A SUBSET
:-----

```

```

XREW   = -50   ;INDEX TO INHIBIT REWIND INDICATOR
XWCTR  = -46   ;INDEX TO WRITE COUNTER
XWILD  = -44   ;INDEX TO WILDCARD INDICATOR
XFLCNT = -42   ;INDEX TO FILE COUNT
XSVMAP = -40   ;
XSVBLK = -36   ;
XSVDAT = -34   ;
XBKLG  = -32   ;
XLSTBK = -30   ;
XBUF   = -26   ;
XSVCNT = -24   ;
XSVNAM = -22   ;
XSVEXT = -16   ;
X1STBK = -14   ;
XSV    = -12   ;INDEX TO SERVICE ROUTINE (DRIVER)
XDN    = -2    ;DRIVE NUMBER INDEX
XER    = -1    ;RESULT STATUS
XCM    = 0     ;INDEX TO COMMAND REGISTER
XWC    = 2     ;INDEX TO WORD COUNT
XBA    = 4     ;INDEX TO BUS ADDRESS
XDT    = 6     ;INDEX TO BLOCK NUMBER OR TAPE SKIP #
XCO    = 10    ;INDEX TO COMMAND
XDR    = 12    ;INDEX TO 1ST DIR BLOCK POINTER
XXNAM  = 14    ;INDEX TO ASCII NAME IN DDB
XBC    = 16    ;INDEX TO REQUESTED BLOCK COUNT
XNB    = 20    ;INDEX TO LAST BLOCK # ALLOCATED
XCKSUM = 22    ;CHECKSUM CALCULATION IN SEARCH

SVC    = XSV   ;ALTERNATE NAME

```

```

:-----
: DEVICE COMMAND CODES
:-----

```

```

INIT$  = 0     ; INITIALIZE DEVICE AND BRING ON LINE
READ$  = 1     ; READ
WRITE$ = 2     ; WRITE
RES$FN = 3     ; RESTORE FUNCTION FOR MONITOR
DIS    = SVC   ; DISPATCH TABLE

```

```

:-----
: CODE BYTE
:-----

```

```

MULUN$ = 100  ; DRIVER PERMITS MULTIPLE DEVICES

```

.PAGE

.SBTTL DEVICE-DEPENDENT EQUATES

SEQ 0022

```
-----  
: RJP04 FUNCTION EQUATES  
-----
```

```
RPWC   = 2   ;RJP04 WORD COUNT REGISTER  
RPBA   = 4   ;RJP04 BUS ADDRESS REGISTER  
RPDA   = 6   ;RJP04 DESIRED SECTOR/TRACK REGISTER  
RPCS2  = 10  ;RJP04 CONTROL STATUS REGISTER 2  
RPER1  = 14  ;RJP04 ERROR REGISTER 1  
RPOF   = 32  ;RJP04 OFFSETT REGISTER  
RPDC   = 34  ;RJP04 DESIRED CYLINDER REGISTER  
RPEC1  = 44  ;RJP04 ECC POSITION  
RPEC2  = 46  ;RJP04 ECC PATTERN  
  
RJREAD = 71  ;RJP04 READ COMMAND  
RJWRITE = 61 ;RJP04 WRITE COMMAND  
DONE   = 200  
ERROR  = 100000
```

.PAGE

SEQ 0023

.SBTTL XXDP DEVICE DRIVER PARAMETER TABLE

```

-----
: DEVICE-DRIVER PARAMETERS
: THESE PARAMETERS ARE USED IN COMMUNICATION WITH THE UTILITY
: PROGRAM
-----

```

```

PARAM:  DISPAT      ;DISPATCH ROUTINE
         .WORD      "DB      ;DRIVER NAME
         .BYTE      MULUN$   ;DEVICE CODE
         .BYTE      11      ;RETURNED DEVICE STATUS (INT DEVICE TYPE)
         .WORD      BCODE    ;BOOT CODE OFFSET

UNIT:   .BYTE      'A      ;UNIT #                (INITIAL REV # A )
ERRB:   .BYTE      '1      ;ERROR STATUS         (INITIAL PATCH # 1)
CMDREG: 176700      ;COMMAND REGISTER ADDR
WCOUNT: 0          ;WORD COUNT
BUSADR: 0          ;BUS ADDRESS
BLOCK:  0          ;BLOCK NUMBER
CMD:    0          ;COMMAND
DIRPTR: DIRBLK    ;POINTS TO 1ST DIR BLOCK.
ASNAM:  0          ;FOR MONITOR COMPATIBILITY
PAREND:

```


.PAGE

.SBTTL DIRBLK TABLE

: PARAMETERS USED TO DEFINE DISK LAYOUT

DIRBLK: 3	:1ST UFD BLOCK ADDR
170.	:NUMBER OF UFD BLOCKS
173.	:1ST BIT MAP BLOCK ADDR
50.	:NUMBER OF MAP BLOCKS
1	:MFD1 BLOCK ADDR
1002	:VERSION 2 FLAG (NOT UPDATED)
48000.	:MAX NUMBER OF BLOCKS ON DEVICE
255.	:# OF BLOCKS TO PREALLOCATE AT ZERO
1	:INTERLEAVE FACTOR
0	:BOOT BLOCK #
MONBLK: 223.	:MONITOR CORE IMAGE BLOCK #
0	:MFD REFRESHED FLAG. 0=NO, NON 0=YES

.SBTTL LOCAL DATA

: LOCAL DATA AND ERROR MESSAGES

ECCPAT: .WORD 0,0 ;STORAGE FOR ECC CORRECTION

.SBTTL ERROR MESSAGES

MWTERR: .ASCIZ <40><40>'? WT ERR'
 MRDERR: .ASCIZ <40><40>'? RD ERR'
 ILLERR: .ASCIZ <40><40>'? ILLEGAL CMND ERR'
 .EVEN

.PAGE

SEQ 0025

.SBTTL MAIN DISPATCH ROUTINE

:*****

: DISPATCH ROUTINE FOR DRIVER

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

:

THIS ROUTINE RECEIVES CONTROL FROM A UTILITY
OR DRVCOM. IT EXAMINES THE COMMAND CODE IN
XCO(R5) IN THE DDB, AND CALLS THE APPROPRIATE
LOCAL FUNCTION.

INPUT:

XCO(R5)

OUTPUT:

CALLS APPROPRIATE INTERNAL FUNCTION.
TESTS ERROR BYTE BEFORE RETURN

REGISTERS CHANGED:

NONE

:*****

```
DISPAT: MOV    R0,-(SP)          ;SAVE
        MOV    R1,-(SP)
        MOV    R2,-(SP)
        MOV    R3,-(SP)
        MOV    R4,-(SP)

        MOV    PC,R1           ;TRUE ADDRESS
        SUB    #.,R1          ;DIFFERENCE BETWEEN TRUE &
                                ;APPARENT
        MOV    #TABLE-2,R0    ;DO A TABLE SEARCH
        ADD    R1,R0          ;GET REAL ADDRESS
10$:    TST    (R0)+           ;TO NEXT FUNCTION
        TST    (R0)          ;END OF TABLE ?
        BMI    110$          ;MI = YES
        CMP    (R0)+,XCO(R5) ;IS IT OUR FUNCTION ?
        BNE    10$           ;NE = NO
        ADD    (R0),R1       ;ELSE GET REAL ADDRESS
        JSR    PC,(R1)       ;AND DO IT
        BR    240$          ;AND LEAVE
```

; HERE IF ILLEGAL FUNCTION

```
110$:  $ABORT #ILLERR        ;NOT LEGAL COMMAND
        MOV    #-1,XER(R5)  ;SIGNAL

240$:  MOV    (SP)+,R4       ;RESTORE
        MOV    (SP)+,R3
        MOV    (SP)+,R2
        MOV    (SP)+,R1
        MOV    (SP)+,R0
        TSTB  XER(R5)       ;Set error indicator
        RTS    PC
```

;FUNCTION TABLE - FIRST ELEMENT IS FUNCTION, SECOND IS ROUTINE

```
TABLE: .WORD  INIT$,INIT    ;INITIALIZE
        .WORD  RES$FN,RESTOR ;MONITOR RESTORE
        .WORD  READ$,DRIVER ;BLOCK READ
        .WORD  WRITE$,DRIVER ;BLOCK WRITE
        .WORD  -1           ;END OF TABLE
```

```
.PAGE  
.SBTTL MAIN ROUTINE: INIT  
;*****  
;ROUTINE TO INITIALIZE THE DEVICE  
;  
;INPUTS:  
;   NONE  
;  
;OUTPUTS:  
;  
;ROUTINES CALLED:  
;  
;REGISTERS CHANGED: NONE  
;--*****  
INIT:  CLR  XER(R5)      ;ASSUME GOOD RESULT  
       RTS  PC
```

.PAGE

SEQ 0027

```

.SBTTL MAIN ROUTINE: RESTORE
:*****
: ROUTINE TO READ PART OF THE MONITOR CORE IMAGE
:
: CALL AS FOLLOWS:
:   PUT BLOCK NUMBER RELATIVE TO MONITOR IN XDT(R5)
:   PUT NUMBER OF WORDS TO READ IN XWC(R5)
:   PUT ADDRESS TO READ INTO IN XBA(R5)
:   PUT REWS$FN IN XCO(R5)
:   JSR PC,@DIS(R5)
:
: GOOD RETURN: DATA READ
: ERROR RETURN: DIS TESTS XER(R5) BEFORE RETURN
:
: ROUTINES CALLED: DIS(R5)
:
: REGISTERS CHANGED: NONE
:-----
RESTOR: ADD    MONBLK,XDT(R5) ;MAKE BLK NUMBER RELATIVE TO 0
        MOV    @READ$,XCO(R5) ;DO A READ FUNCTION
        JSR    PC,@DIS(R5)   ;LET DRIVER DO IT
        RTS

```

```

.PAGE
.SBTTL  MAIN ROUTINE: DRIVER
:*****
:  READ-WRITE DRIVER FOR THE RJP04
:
:  CALLED FROM DISPATCH
:    PERFORMS READ$ AND WRITE$ FUNCTIONS
:
:  GOOD RETURN:
:    TRANSFER EFFECTED, XER(R5) CLEARED
:  ERROR RETURN:
:    MESSAGE TYPED, XER(R5) NONZERO
:
:  REGISTERS CHANGED:
:    R0,R1,R2,R3,R4
:-----

```

```

DRIVER:
        CLR      XER(R5)          ;ASSUME SUCCESSFUL RESULT
        MOV      #11.,R4         ;# OF TIMES TO RETRY ON ERRORS
RPDRV1: DEC      R4              ;SHOULD WE CONTINUE?
        BLE     33$             ;NO,SO REPORT ERROR
        MOV     (R5),R3         ;DEVICE ADR
        MOV     XDN(R5),R0      ;GET UNIT NUMBER
        BIC     #177400,R0      ;STRIP OFF ANY JUNK
        MOV     R0,RPCS2(R3)    ;LOAD RESULT INTO RPCS2
        MOV     #10000,RPOF(R3) ;SET 16 BIT FORMAT IN RPOF REG
        MOV     #23,(R3)       ;DO A PACK ACK TO SET VV BIT
        MOV     XWC(R5),RPWC(R3) ;WORD COUNT
        NEG     RPWC(R3)       ;TWO'S COMPLEMENT OF WC
        MOV     XBA(R5),RPBA(R3) ;BUS ADR
        MOV     XDT(R5),R1     ;BLOC NUMBER
        MOV     #22.,R2       ;22 SECTORS PER TRACK
        CLR     R0
1$:     SUB     R2,R1          ;DIVIDE BY SECTOR SIZE
        BLO     2$
        INC     R0            ;UP TRACK COUNT
        BR     1$
2$:     ADD     R2,R1          ;WENT TOO FAR
        MOV     R1,-(SP)       ;PUT SECTOR # ON STACK
        CLR     R1
        MOV     #19.,R2       ;19 TRACKS PER CYLINDER
3$:     SUB     R2,R0          ;DIVIDE BY TRACKS PER CYL
        BLO     4$           ;TO GET TRACK AND CYL #
        INC     R1            ;UP CYL COUNT IN R1
        BR     3$           ;R0 IS HOLDING TRACK #
4$:     ADD     R2,R0          ;MAKE UP FOR GOING TOO FAR
        SWAB    R0            ;MOVE TRACK # TO LEFT
        BIS     (SP),R0       ;OR IN RIGHT SIDE (SECTOR)
        MOV     R0,RPDA(R3)    ;TO DSK ADR REG
        MOV     R1,RPDC(R3)    ;TO DSK CYL ADR REG
        CMP     #READ$,XCO(R5) ;IS A READ ?
        BNE     10$          ;NE = NO, MUST BE A WRITE
        MOV     #RJREAD,(R3)  ;ELSE START IT
        BR     30$
10$:    MOV     #RJWRIT,(R3)  ;START WRITE

```

```

30$: BIT #DONE!ERROR,(R3) ;DONE OR ERROR?
      BEQ 30$ ;NEITHER
      BPL 20$ ;DONE
      BIT #100000,RPER1(R3) ;WAS A DATA CHECK ERROR?
      BEQ 32$ ;EQ = NO
      BIT #100,RPER1(R3) ;YES, IS IT CORRECTABLE?
      BNE 32$ ;NE = NO
      JSR PC,ECCOR ;ELSE CORRECT IT
31$: MOV #40,RPCS2(R3) ;CLEAR ERROR CONDITION
      TSTB (R3) ;WAIT TILL DONE
      BPL 31$
      BR 20$ ;AND LEAVE

32$: MOV (R3),R0 ;SAVE ERROR INFORMATION
35$: MOV #40,RPCS2(R3) ;CONTROLLER CLEAR
      TSTB (R3) ;DONE?
      BPL 35$
      BIT #40000,R0 ;WAS IT HARD ERROR?
      BEQ RPDRV1 ;NO

33$: DECB XER(R5) ;INDICATE ERROR
      CMP XCO(R5),#READ$ ;WAS ERROR ON READ?
      BEQ 36$ ;YES
      .FRCTYP #MWTERR ;PRINT WRITE ERROR
      BR 20$ ;RETURN TO CALLER
36$: .FRCTYP #MRDERR ;PRINT READ ERROR
20$: RTS PC

```

.PAGE

SEQ 0030

.SBTTL ROUTINE ECCCOR

```

:*****
: CORRECT A SOFT ECC ERROR
: (ALGORITHM ADAPTED FROM THAT IN CZR6PD)
: USES HARDWARE ERROR BURST PATTERN TO CORRECT A FAULTY
: SEQUENCE OF UP TO 11 BITS
:
: CALLED BY DRIVER
:
: GOOD RETURN:
: DATA CORRECTED IN BUFFER
:
: REGISTERS CHANGED:
: R0,R1,R4
:*****

```

```

ECCCOR: MOV     RPEC2(R3),ECCPAT ;ERROR BURST PATTERN
        CLR     ECCPAT+2      ;WILL SHIFT INTO THIS
        MOV     R3,-(SP)      ;SAVE
        MOV     RPEC1(R3),R1  ;ERROR BURST POS COUNT
        MOV     XBA(R5),R3    ;BUFFER ADDRESS
        MOV     XWC(R5),R4    ;WORD COUNT
        ASL     R4            ;NOW BYTE COUNT
        MOV     R3,-(SP)      ;CALCULATE END OF
        ADD     R4,(SP)       ;TRANSFER
        DEC     R1            ;CONVERT TO BIT DISPLACEMENT
        MOV     R1,R0         ;SAVE
        ASR     R1            ;COMPUTE BYTE DISPLACEMENT
        ASR     R1
        ASR     R1
        BIC     #1,R1         ;WORD DISPLACEMENT
        CMP     R1,R4         ;ERROR WITHIN TRANSFER?
        BHIS   10$           ;HIS = NO, RETURN
        ADD     R1,R3         ;COMPUTE BUFFER ADDRESS OF ERR
        BIC     #177760,R0    ;STARTING BIT DISPLAC IN WORD
        BEQ    5$            ;EQ = ON WORD BOUNDARY

3$:     ASL     ECCPAT         ;SHIFT PATTERN 1 BIT LEFT
        ROL     ECCPAT+2      ;POOR MAN'S ASHC
        DEC     R0            ;DECREMENT COUNT
        BNE    3$            ;UNTIL DONE

5$:     MOV     (R3),R0        ;CORRECT FIRST WORD
        MOV     ECCPAT,R1     ;WITH XOR OF PATTERN
        BIC     ECCPAT,(R3)   ;POOR MAN'S XOR
        BIC     R0,R1
        BIS     R1,(R3)+
        CMP     (SP),R3       ;CHECK IF SECOND WORD IS
        BEQ    10$           ;IN BUFFER, EQ= NO, ALL DONE
        MOV     (R3),R0      ;ELSE DO NEXT WORD
        MOV     ECCPAT+2,R1
        BIC     ECCPAT+2,(R3)
        BIC     R0,R1
        BIS     R1,(R3)

10$:    TST     (SP)+         ;BUMP TEMP STORAGE
        MOV     (SP)+,R3
        RTS    PC

```

:SECONDARY BOOT CODE AREA
:
:BCODE:

.PAGE
.SBTTL BOOTSTRAP REVISION HISTORY
:-----
: REV DATE CHANGE
:-----
: 1.0 12-JUL-78 INITIAL ISSUE
: 1.1 17-NOV-78 MAKE COMPATABLE WITH XXDP.
: 1.2 12-JUL-82 MODIFIED TO SUIT VAX ASSEMBLER
: 1.3 29-MAR-83 WHEN TRYING TO BOOT TO UNIT OTHER
: THAN 0 AND UNIT 0 NOT ON BUSS, A
: HALT AT 216 OCCURS
: 21-FEB-84 V2 CHANGE STACK AND MON SIZE
:-----

.PAGE

SEQ 0032

```

.SBTTL  BOOTSTRAP

.NLIST  CND
.LIST   MEB

RBCS1   = 0
RBWC    = 2
RBBA    = 4
RBDA    = 6
RBCS2   = 10
RBDS    = 12
RBDC    = 34
BEGIN   = 1046
MONCNT  = 20000-256.    ;SKIP BOOT BLOCK

RBOOT:  NOP
        BR      START    ;START BOOT ROUTINE
        .WORD   6
        HALT    ; TRAP CATCHER
        .WORD   12      ;RESERVED INSTRUCTION ERR
        HALT    ; TRAP CATCHER
        .BLKB   4

RBCSA:  .WORD   176700    ;RJPO4 DEFAULT CSR ADDRESS

START:  NOP
        BR      START1
        .BLKB   12
        .WORD   0,0
        .BLKB   24

START1: MOV    #60000,SP    ;SET UP STACK
        MOV    RBCSA,R5    ;GET RBCS1 ADDRESS
        MOV    #23,(R5)    ;DO PACK ACK TO SET VV BIT
        MOV    RBCS2(R5),R2 ;GET UNIT NUMBER
        BIC    #177770,R2
5$:     MOV    #40,RBCS2(R5) ;CLEAR CONTROLLER
        MOV    R2,RBCS2(R5) ;SET UNIT NUMBER
10$:    BIT    #100200,(R5) ;READY?
        BEQ    10$         ;NO
        BMI    25$         ;ERROR
15$:    TSTB   RBDS(R5)    ;DRIVE READY?
        BPL    15$         ;NO
        MOV    #-MONCNT,RBWC(R5) ;SET UP WORD COUNT
        MOV    #1000,RBBA(R5) ;LOAD AT LOCATION 1000
        MOV    #5003-1,RBDA(R5) ;BLOCK # OF MONITOR
        MOV    #0,RBDC(R5)    ;CYL #
        MOV    #71,(R5)      ;DO READ COMMAND
20$:    BIT    #100200,(R5) ;DONE OR ERROR?
        BEQ    20$         ;NOT DONE
        BPL    30$         ;DONE
25$:    MOV    (R5),R0      ;SAVE STATUS
        MOV    RBCS2(R5),R1 ;AND ANY ERRORS
        HALT   ;HALT ON ERROR
        BR    5$          ;OK, TRY AGAIN
30$:    MOV    R5,R1      ;PUT CSR ADDRESS IN DRIVER TABLE
        JMP   @#BEGIN    ;START UP HIMON

.END

```

Appendix: B - Assembly and Linking Instructions

The Driver and Boot must be merged together and then assembled as a .MAC file. They should be maintained separately as shown in appendix A, that is they have their own revision blocks. Assembling them together helps to eliminate double references that would otherwise occur. References to an absolute location by the BOOT code must be done via an offset from BCODE:, which will be at absolute zero during the boot operation.

Command file for DB under VMS

```
! Command file to create a XXDP V2 DB DRIVER
!
! MCR MAC DB,DB/CRF/-SP=MACROM.MAC,DB.MAC
!
! Set the address limits for the driver and create
! a binary file
!
! MCR TKB
DB/NOMM/NOHD/SQ,DB/-SP=DB
/
PAR=DUMMY:0:3200
STACK=0
/
$ WRITE SYS$OUTPUT " Now type TKBBIN <CR> , "
$ WRITE SYS$OUTPUT " When prompted for the file name enter DB."
$ WRITE SYS$OUTPUT " will create a driver called DB.BIN ."
```

Appendix: C - Driver Equates

```

: **
: XXDP* Version 2      Equate Definitions
: --
:
:  DEVICE COMMAND CODES
:
INIT$  = 0      ; INITIALIZE DEVICE and BRING ON LINE
READ$  = 1      ; READ
WRITE$ = 2      ; WRITE
RES$FN = 3      ; RESTORE FUNCTION for XXDP-SM
RFS$FN = 100    ; REFORMAT SINGLE DENSITY
RFD$FN = 101    ; REFORMAT DOUBLE DENSITY
PRE$TP = 200    ; TAPE - PREPARE
REW$TP = 201    ; TAPE - REWIND
SPR$TP = 202    ; TAPE - REVERSE SPACE
WHD$TP = 203    ; TAPE - WRITE HEADER
RHD$TP = 204    ; TAPE - READ HEADER
SEF$TP = 206    ; TAPE - SKIP to EOF
WEF$TP = 207    ; TAPE - WRITE EOF
SET$TP = 210    ; TAPE - SKIP to EOT
STA$TP = 211    ; TAPE - RETURN STATUS CODE
DEN$FN = 374    ; RETURN DENSITY (0 = LOW, 1 = HIGH)
CMP$FN = 375    ; COMPUT BLOCK # from SECTOR
WRT$FN = 376    ; WRITE absolute SECTOR
RED$FN = 377    ; READ absolute SECTOR
:
:  DEVICE CODE BYTE
:
BBSUP$ = 2      ; BAD BLOCK SUPPORT
NOREN$ = 4      ; TAPE CANNOT RENAME FILE
NODIR$ = 10     ; NOT A DIRECTORY DEVICE
TAPED$ = 20     ; IS A TAPE DEVICE
REFDN$ = 40     ; SUPPORTS SINGLE/DOUBLE DENSITY FORMAT
MULUN$ = 100    ; DRIVER SUPPORTS MULTIPLE UNITS/DRIVER
:
:  DEVICE RETURNED STATUS BYTE
:
BOTTP$ = 2      ; TAPE IS AT BOT
TMKTP$ = 4      ; TAPE IS AT TAPE MARK
EOTTP$ = 10     ; TAPE IS AT EOT

```

Appendix: D - Device Type Codes

The Device Type Code (DTC) is placed into byte location 41 by the monitor every time a binary file is run. This byte is then designated the "load medium indicator". DTC's are assigned as follows:

DTC	DEVICE Type	XXDP+ Version	Notes
0	paper tape or ACT11	1.3	
1	TU56 (DECTape)	1.3	
2	RK05 (disk)	1.3	
3	RP02/RP03 (disk)	1.3	
4	TM10 (magtape)	1.3	
5	TA11 (cassette)	1.3	
6	TU16/TM02 (magtape)	1.3	2.0
7	not used		
10	RX01 (floppy disk)	1.3	
11	RP04/R505/RP06 (disk)	1.3	2.0
12	RS03/RS04 (disk)	1.3	
13	RK06/RK07 (disk)	1.3	2.0
14	RL01/02 (disk)	1.3	2.0
15	RX02 (disk)	1.3	2.0
16	RM02/RM03 (disk)	1.3	2.0
17	TU58 (cassette)	1.3	2.0
20	TU58/PDT11 (cassette)	1.3	
21	TS04 (tape)	1.3	2.0
22	TM78 (tape)	1.3	
23	UDA (disk MSCP)	1.3	2.0 1
24	TR79 (tape)	1.3	
25	RD/RX50 (disk)	1.3	2.0 1
26	RC25 (disk)	1.3	2.0 1
27	TK50 (tape MSCP - TMSCP)	1.3	2.0

NOTES:

1. These are MSCP class devices and under XXDP V2 are handled by one driver which uses DTC = 23