XXDP V2 Driver Guide

Driver Programmer's Guide

Table of Contents

Appendices

PRODUCT NAME: AC-U036A-MC
PRODUCT NAME: CHQDPAC XXDPV2 DRVP PROGR GD
PRODUCT DATE: 8 Oct 1984

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

Transcription History:
by Ian Hammond (IJH) 29-Mar-2015

Transcription source:
http://bitsavers.org/pdf/dec/pdp11/microfiche/ftp.j-hoppe.de/bw/gh/AH-FG10A-MC__XXDP__XXDP_V2_DRVR_PROGR_GD__CHQDPA0__(C)1984.pdf

The transcription is missing the appendices.

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 VI 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 VI 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 01 has been retired).

f) Some functional aspects of the utilities were changed. UPD2 will no longer permit an image copy between devices of differing sizes and will not copy the monitor during file copy.

1.2 Compatibility

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

a) The V1 DL and DH disk layout did not allow for a 32k Monitor. If the V2 Monitor is installed on a VI medium the first file (or two) after the monitor area will be corrupted.

b) 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. VI 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.)

c) V2 media will have the octal constant 1002 at octal displacement 14 (the old MFD2 pointer) in the MFD. VI media will have some other value. The MFD is not currently read by most drivers, so this fact is not used.

d) 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 V2 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-S666A-M0) CHQFSAO 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' (006). 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 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 parameters have been given INITIAL values and functions which are not related to their functions during execution.)

A parameter table example is:

```
PARAM:   .WORD   DISPAT          ;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:  .WORD   174400          ;COMMAND REGISTER ADDRESS
WCOUNT:  .WORD   0               ;WORD COUNT
BUSADR:  .WORD   0               ;BUS ADDRESS
BLOCK:   .WORD   0               ;BLOCK NUMBER
COMD:    .WORD   0               ;COMMAND
DIRPTR:  .WORD   DIRBLK          ;POINTS TO 1ST DIR BLOCK
ASNAME:  .WORD   0               ;FOR MONITOR COMPATIBILITY
PAREND:
```

1) Dispatch Routine Address
This entry is the address of the dispatch routine, which determines which driver routines to Involve. 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.
NQDIR$	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 DD6 and is used by the driver to access all device registers.

9) UCOUNT, 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 92 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 "RO 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 ODB, 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 four 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. In it may also perform auxiliary 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 Auxiliary Routines

These routines are called by DISPATCH, INIT and DRIVER.

3.0 Device Driver 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, overlayed) 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 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 GTMFDl 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.

[IH> Regarding *corruption*: a diagnostic is permitted to overwrite a low area of the resident XXDP monitor. The monitor checks that area for *corruption* after image exit (using a checksum) and reloads the area from disk if required.]

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 DCB.

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 signaled 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 UP02 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

RFSFN,RFDFN ???
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 bvte in tne DD6 and by a negative first word in DIR6LK; 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 comnon 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 B0T 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 independent 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 two 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 UP02 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 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 – TU58

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
Two units/driver	- Yes
Dispatch	- Table

DM – RK06,07

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 – RL01,02

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        - Inter-record gap. The gap that is written between
             records on magtape.
MFD        - Master File Directory
RAD50      - Radix-50. A method of encoding three ascii characters
             into one 16-bit word.
UIC        - User Identification Code.
```

7.0 Bibliography

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