VAX/VMS
Training

VAX/VMS
Device Driver
Debugging
DEBUGGING

Prepared by Educational Services of Digital Equipment Corporation
Copyright © 1984 by Digital Equipment Corporation
All Rights Reserved

The reproduction of this material, in part or whole, is strictly prohibited.
For copy information, contact the Educational Services Department,
Digital Equipment Corporation, Bedford, Massachusetts 01730.

Printed in U.S.A.

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.

The software described in this document is furnished under a license
and may not be used or copied except in accordance with the terms of
such license.

Digital Equipment Corporation assumes no responsibility for the use
or reliability of its software on equipment that is not supplied by Digital.

The manuscript for this book was created using DIGITAL Standard
Runoff. Book production was done by Educational Services
Development and Publishing in Nashua, NH.

The following are trademarks of Digital Equipment Corporation:

<table>
<thead>
<tr>
<th>digital™</th>
<th>DATATRIEVE</th>
<th>DEC</th>
<th>DECMate</th>
<th>DECnet</th>
<th>DECS</th>
<th>DECS-10</th>
<th>DECS-20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DECtape</td>
<td>DECUS</td>
<td>DECwriter</td>
<td>DIBOL</td>
<td>MASSBUS</td>
<td>PDP</td>
<td>P/OS</td>
</tr>
<tr>
<td></td>
<td>Rainbow</td>
<td>RSTS</td>
<td>RSX</td>
<td>UNIBUS</td>
<td>VAX</td>
<td>VMS</td>
<td>VT</td>
</tr>
</tbody>
</table>
1. Must test all code threads, all processor types

2. START with Q10W
   all function codes, ensure illegal ones trapped

3. Use 1056

4. Use Q10s to queue 1056

5. BECANCEL
   image run down
   ↑ Y Control Y

6. Handle all characters

7. RMS/COPY
INTRODUCTION

Since device drivers run in kernel mode, at high interrupt priority levels, any error is considered serious, and may cause the system to crash. Before a driver is put into operation, it must be thoroughly tested.

VMS offers several tools to aid in driver debugging. Among these are debuggers (DELTA and XDELTA) and a system dump analyzer (SDA). This module discusses the various debugging tools available, and how those tools can be used. In addition, this module presents some general techniques/suggestions for identifying where problems occur in driver routines. Also, some general driver testing hints are presented.

OBJECTIVES

Upon completion of this module, given the lecture notes, you will be able to:

- Observe the operation of a driver by examining various driver-related data structures using the executive debugger (XDELTA) and the system dump analyzer (SDA).
- Determine the cause of a driver-related crash by using XDELTA commands, analyzing console bugcheck listings and using the system dump analyzer.
- Describe what must be done to fully test a driver.

RESOURCES

1. Guide to Writing a Device Driver for VAX/VMS
2. VAX-11 Software Installation Support Manual
DEBUGGING

TOPICS

I. Debugging Tools Overview
II. Analyzing Console Bugcheck listings
III. XDELTA Command Summary
IV. CHMK - Accessing Kernel mode with DELTA
V. SDA Command Summary
VI. Driver Testing Hints
**VAX/VMS DEBUGGING**

Under the VAX/VMS operating system, there are several ways to "debug" or analyze a problem that is occurring in a program. The method of analysis depends on the "environment" the program is running under and the nature of the analysis you wish to do (i.e., monitoring only, stepping through a program). The table below describes the type of analysis, program environment, and suggested method of analysis (the debugger).

<table>
<thead>
<tr>
<th>Problem/Environment</th>
<th>Method of Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program IPL=0, User mode, Examine perprocess memory</td>
<td>VAX/VMS SYMBOLIC DEBUGGER (linked with image or included at run time, User mode only)</td>
</tr>
<tr>
<td>Program IPL=0, User to Kernel mode, Examine process and system memory</td>
<td>DELTA DEBUGGER (linked with image or included at run time, User to Kernel mode) Nonsymbolic</td>
</tr>
<tr>
<td>Examine active system</td>
<td>System Dump Analyzer (SDA) Activated from DCL, can examine active system</td>
</tr>
<tr>
<td>Examine a Crash file</td>
<td>System Dump Analyzer (SDA) Activated from DCL</td>
</tr>
<tr>
<td>Program IPL &gt; 0</td>
<td>XDELTA DEBUGGER (linked with VMS, run from console terminal only) Nonsymbolic</td>
</tr>
</tbody>
</table>
BUGCHECK OUTPUT

While debugging a driver, the system will probably crash several times. The Bugcheck output appearing on the system console (and in the SDA dump file) often can be used to locate the instruction in the driver causing the crash. The bug (causing the identified instruction to crash the system) may actually occur prior to the instruction's execution. By locating the instruction, you will at least have a handle on how far an I/O request got before the crash occurred.

Examine the Bugcheck output below.

```
* RUN WRITEV
BEFORE ASSIGNMENT MADE
ASSIGNMENT SUCCESSFUL

**** FATAL BUG CHECK, VERSION = SS5UXCEPT, Unexpected system service exception
CURRENT PROCESS = VIKP

REGISTER DUMP

R0 = 7FFEFE35
R1 = 800084E7
R2 = 8005BF30
R3 = 00000002
R4 = 80059A50
R5 = 80059240
R6 = 7FFE1DC0
R7 = 00000020
R8 = 8005A39C
R9 = FFFFFFFB
R10 = 00000030
R11 = 80059A8E
AP = 7FFEEDB4
FP = 7FEEDB8C
SP = 7FEEDB84
PC = 80084E8
PSL = 00020000

KERNEL/INTERRUPT STACK

Condition Handler

MECHANISM ARRAY

CONDITION HANDLER

MECHANISM ARRAY

Example 7-1 Sample Bugcheck (Console Output)
DEBUGGING

Use the following information to analyze the previous crash.
Base Address of Driver

$MCR SYSGEN
SYSGEN> SHOW/DEVICES=PT

--- Driver --- Start --- End --- Dev --- DDB --- CRB --- IDB --- Unit --- UCB ---
PTDRIVER 8005A320 8005A7E0 PTA 80055810 80055850 80055880 0 80059240

Part of driver . MAP file

<table>
<thead>
<tr>
<th>MODULE NAME</th>
<th>IDENT</th>
<th>BYTES</th>
<th>FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTDRIVER</td>
<td>V03</td>
<td>1206</td>
<td>DRA01TVK, RUG1PTDRIVER, ORJ14R</td>
</tr>
<tr>
<td>SYS</td>
<td>.STB1</td>
<td></td>
<td>0 DRA01SYSXFSYS, STR11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P-SECT NAME</th>
<th>MODULE(S)</th>
<th>BASE</th>
<th>FIND</th>
<th>LENGTH</th>
<th>ALIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>$88105PROLOGUE</td>
<td>PTDRIVER</td>
<td>00000000</td>
<td>000000050</td>
<td>00000051</td>
<td>81.</td>
</tr>
<tr>
<td>$88115DRIVER</td>
<td>PTDRIVER</td>
<td>00000000</td>
<td>000000050</td>
<td>00000051</td>
<td>81.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>VALUE</th>
<th>SYMROL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DYNSC_CRB</td>
<td>00000005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNSC_DDB</td>
<td>00000006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNSC_DPT</td>
<td>0000001E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNSC_UCB</td>
<td>00000010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERLASDEVICFPR</td>
<td>8000A3ER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERLASDEVICMG</td>
<td>8000A3EC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;ARORTIO</td>
<td>800092FF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;ALOCRUF</td>
<td>800093B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;BUFFROQDTA</td>
<td>8000978F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;FINISHLOC</td>
<td>8000930C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;IOPOFK</td>
<td>800095C4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;IOQACPPKT</td>
<td>80009329</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;IOQRVPKT</td>
<td>80009124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;READCK</td>
<td>80008055</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXE&amp;WRITECK</td>
<td>80008080</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOC&amp;CANCELIO</td>
<td>8000A551</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOC&amp;REGCOM</td>
<td>8000A6B3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOC&amp;RETURN</td>
<td>8000A869</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOC&amp;WFFPCH</td>
<td>8000A96A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCB&amp;PHD</td>
<td>00000064</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCB&amp;BYCTCT</td>
<td>00000042</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB&amp;JPL</td>
<td>00000112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PBUS&amp;MOUNT</td>
<td>00000111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT&amp;DDT</td>
<td>00000054-R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KEY FOR SPECIAL CHARACTERS ABOVE:**

- * - UNDFPINF
- I - UNIVERSAL
- R - PLOOCURALF
- W - WEAK

**LINK=W-USRTFP, Image "PTDRIVER" has no user transfer address**

Example 7-1 Sample Bugcheck (Cont) (driver.MAP)

7-7
DEBUGGING

Example 7-1 Sample Bugcheck (Cont)
(Driver Listing File)
The following steps are keyed to the sample Console Bugcheck Output in Example 7-1.

1. The AP points to a condition handler. (Note that sometimes the AP does not contain such a pointer; a signal and mechanism array may still appear on the top of the kernel/interrupt stack.)

2. The mechanism array holds the values of R0 and R1 at the time of the crash. (Other registers are in the Register Dump section.)

3. The signal array contains the reason for the crash (an $S$S$Serror$S$S_code), and the PC and PSL at the time of the crash. The System Services Reference Manual contains descriptions of what the other parameters mean for various $S$S$error_code$s. For an access violation, $S$S$ACCVIO$, the other information includes the type of access violation, and the virtual address referenced (which caused the fault).

The most important piece of information is the PC. If the PC indicates an address in your driver (i.e., lies in the range defined by the SYSGEN SHOW/DEVICES command for your driver), the following procedures can be used.

4. Find your driver's starting address (using information from the SYSGEN SHOW/DEVICES command, which you typed before testing your driver).

5. Find the base address for the $S$$S$115$S$$S$ DRIVER PSECT in your driver.MAP file.

6. Add the two numbers found in steps 4 and 5. This gives you the starting address of your driver code.
DEBUGGING

7. Subtract the number computed in step 6 from the PC found in step 3. This gives you the offset into your driver listing file (driver.LIS) where you will find the instruction causing the crash.

For example

A. Driver start address = \( \text{8005A320} \)
   \[ \text{+ $$$115\_DRIVER PSEC base = 54} \]
   \[ \text{8005A374} \] (Note: addition is done in hex.)

B. \( \text{PC = 8005A402} \)
   \[ \text{- A = 8005A374} \]
   \[ \text{8E (hex)} \]

The instruction causing the system to crash is (see the driver listing file):

\[ \text{MOVL R2,IRPSL\_SVAPTE(R3)} \]

In this case, there is nothing wrong with the instruction itself. The error occurred because R3 had not been saved prior to the call to EXE\$ALLOCBUF (which destroys the contents of R3). As a check, note that the register dump shows that R3 contains the value 2 (clearly not an IRP address). The offset IRPSL\_SVAPTE has a value of 2C, so the faulting virtual address in the signal array also makes sense (2C + 2 = 2E).

When the faulting PC is a system address, but not a driver address, the system map (SYS\$SYSTEM:SYS\$MAP) must be consulted to see which module in the system caused the crash. From this information, it is sometimes possible to determine where in the driver the problem occurred (e.g., a crash caused by module SYSQIOREQ might indicate that FDT routines are exiting with registers R3-R5 modified, possibly following a MOVC instruction for which R3-R5 were not saved and restored). If examination of the system map reveals no clues, or the faulting PC is not part of the executive/driver, you must place breakpoints in your driver (using XDELT), and step through your driver code until you identify the faulting instruction(s).

In general, XDELT is used as a last resort. The prime reason for using XDELT is to set breakpoints to identify how far an I/O request went before a crash occurred. SDA (and console bugcheck listings) are better tools for looking at data structures after crashes. XDELT is used dynamically, while SDA is used statically.

7-10
DEBUGGING

DELTA/XDELTA

**DELTA**

**Usage**
User images

Terminal used for control
Any TTY

IPL
= 0

How activated
Linked or included at run time

Access Mode
All modes

**XDELTA**

Operating System Drivers

Console only (OPA0:)

> 0

Included at boot time

Kernel mode only

Both debuggers are:

- Nonsymbolic
- Use same command syntax
- No visible prompt
- Error message is "Eh?"
BOOTSTRAPPING VMS WITH XDELTA

VAX-11/780

>>> DEPOSIT R3 0   ; Unit number 0 in R3
SYSBOOT> SET BUGREBOOT 0
SYSBOOT> CONTINUE

VAX-11/750

>>> B/7 DMA0   ; For B/F, if F=7 include XDELTA
               ; see DD manual for more information
SYSBOOT> SET BUGREBOOT 0
SYSBOOT> CONTINUE

VAX-11/730

>>> D/G/L 3 1   ; Deposit unit number 1 in R3
>>> @DQAXDT
SYSBOOT> SET BUGREBOOT
SYSBOOT> CONTINUE

; Boot from DQA1 with XDELTA
DEBUGGING

Bootstrapping VMS with XDELTA (Cont)

P
>>>HALT

HALTED AT 80008462

>>>@DB0XDT
DBzXDT for RM03 or RP06
DMzXDT for RK07

!

DB0 CONVERSATIONAL/DEBUG BOOT COMMAND FILE - DB0XDT

BOOT FROM DB0 AND STOP IN SYSBOOT AT ALTER PARAMETERS
.
.
.

DEPOSIT R5 7 ! SOFTWARE BOOT FLAGS (CONVERSATIONAL/DEBUG)
.
.
.

SYSBOOT> SET BUGREBOOT Ø

SYSBOOT> CONT

VAX/VMS Version 3.0 27-MAY-82 10:30

1 BRK AT 800017D ;P
DEBUGGING

Invoking XDELT A

To invoke XDELT A, enter the following commands at the console in the order given.

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;control-P&gt;</td>
<td>Activates XDELT A at the console</td>
</tr>
<tr>
<td>HALT</td>
<td>Halts the processor (780 only)</td>
</tr>
<tr>
<td>D/I 14 5</td>
<td>Deposits &quot;5&quot; into internal register 14 (software interrupt request register) to generate an interrupt at IPL 5.</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Resumes system execution</td>
</tr>
</tbody>
</table>

After issuing the CONTINUE command, the following happens:

- The IPL 5 interrupt occurs when the processor IPL drops below 5.
- XDELT A gains control at IPL 5, raises IPL to 31, and suspends execution at a breakpoint.
- At the breakpoint, XDELT A can accept an XDELT A command entered at the console and act accordingly.

Example:          >>>H

                      HALTED AT 800082EE
                      >>>D/I 14 5
                      >>>CONTINUE
                      1 BRK AT 8000B17D
                      ip
DEBUGGING

Notice that the procedure described on the previous page works only for driver FDT routines running at IPL 2. For other driver routines, running at fork IPL or above, the XDELTA interrupt will not be granted, since XDELTA is invoked at IPL 5. To invoke XDELTA at elevated IPLs, use the following statement in your driver:

JSB G^INI$BRK

This may be used where you want to set XDELTA breakpoints in your driver (e.g., at the beginning of each driver routine, to isolate which routine is causing a system crash).
DEBUGGING

DELTA Debugger

To use the DELTA debugger, assemble and link a program in the following fashion:

1. $ MACRO prog-name + SYS$LIBRARY:LIB/LIB
2. $ LINK/DEBUG prog-name, SYS$SYSTEM:SYS.STB/SELECT
3. $ DEFINE LIB$DEBUG DELTA
4. $ RUN prog-name

Steps:

1. Assemble the program allowing system macros to be defined (SYS$LIBRARY:LIB/LIB)
2. Link the program with a debugger and resolve any system symbols (SYSSYSTEM:SYS.STB)
3. Define the debugger used to be DELTA
4. Activate the program, mapping in DELTA
DEBUGGING

CHMK PROGRAM

It is often convenient to observe data structures changing dynamically. One way to gain access to kernel mode data structures is to run the CHMK program (see Example 7-2). This program allows any privileged process (with CMKRNL privilege) to change mode to kernel, and enter XDELTA commands (e.g., to look at system data structures). Extreme caution should be exercised so that data structures are not modified, since such modification could lead to a system crash.

Perform the following steps to use the CHMK program:

1. Assemble and link CHMK
2. Run the CHMK program
3. Enter a breakpoint in the program and tell it to proceed

Corresponding Commands:

1. $ MACRO CHMK and $ LINK CHMK
2. $ RUN CHMK.EXE
3. 215;B;P

Note on Step 3, no prompt from DELTA is given.

After you receive the "stopped at breakpoint" message, you are in kernel mode, and may proceed to examine system data structures. To leave the program, type ';P', followed by 'EXIT' (if you just type EXIT, you will be logged off, since kernel mode exit implies process deletion).

GO:        .WORD 0                         ; null entry mask
$CMKRNL_S ROUTIN = 10$               ; enter kernel mode
RET
10$:       .WORD 0                         ; null entry mask
NOP
NOP
MOVZBL #$SS$_NORMAL,R0          ; where BPT instruction
RET
.END GO

Example 7-2 The CHMK program, which must be run with DELTA
## Debugging DELTA/XDELT A

### Table 7-2 DELTA/XDELT A Functions and Commands

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display contents of given address</td>
<td>address/</td>
<td>GA88/00060034</td>
</tr>
<tr>
<td>Display contents as instruction</td>
<td>address!</td>
<td></td>
</tr>
<tr>
<td>Replace contents of given address</td>
<td>addr/contents new</td>
<td>GA88/00060034 GA88 GA88/00060034 'A' (replace as ASCII)</td>
</tr>
<tr>
<td>Display contents of previous location</td>
<td>&lt;ESC&gt;</td>
<td>80000A88/80000BE4 &lt;ESC&gt; 80000A84/00000000</td>
</tr>
<tr>
<td>Display contents of next location</td>
<td>addr/contents &lt;LF&gt;</td>
<td>80000004/8FC0FFC 80000005/50E9002C</td>
</tr>
<tr>
<td>Display range of locations</td>
<td>addr,addr'/contents</td>
<td>G4,GC/8FC0FFC 80000008/50E9002C 8000000C/0000400</td>
</tr>
<tr>
<td>Display indirect</td>
<td>&lt;TAB&gt;</td>
<td>80000A88/80000BE4 &lt;TAB&gt; 80000BE4/8000078</td>
</tr>
<tr>
<td>or</td>
<td>/</td>
<td>80000A88/80000BE4/8000078</td>
</tr>
<tr>
<td>Single step</td>
<td>S</td>
<td>1 brk at 8000B17D</td>
</tr>
<tr>
<td>command</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Step Over Subroutine</td>
<td>O</td>
<td>8000B17E/9A0FBB05</td>
</tr>
<tr>
<td>Set Breakpoint</td>
<td>addr;N;B &lt;ret&gt;</td>
<td>800055F6;2;B (N is a number 2-8)</td>
</tr>
<tr>
<td>Display Breakpoint</td>
<td>;B</td>
<td>1 8000B17D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 800055F6</td>
</tr>
</tbody>
</table>
### Table 7-2 DELTA/XDELTA Functions and Commands (Cont)

<table>
<thead>
<tr>
<th>Function</th>
<th>Command</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Breakpoint</td>
<td>(0;N;B \text{&lt;ret&gt;})</td>
<td>(0;2;B)</td>
</tr>
<tr>
<td>Proceed from Breakpoint</td>
<td>(;P)</td>
<td>(;P)</td>
</tr>
<tr>
<td>Set Base Register</td>
<td>'value',N,X</td>
<td>(80000000,0,X)</td>
</tr>
<tr>
<td>Display Base Register</td>
<td>(Xn \text{&lt;ret&gt;}) or (Xn=)</td>
<td>(X0) or (X0=00000003)</td>
</tr>
<tr>
<td>Display General Register</td>
<td>Rn/(n\text{ is in Hex})</td>
<td>(R0/00000003)</td>
</tr>
<tr>
<td>Show Value</td>
<td>expression=</td>
<td>(1+2+3+4=0000000A) (+,-,\times,\div\text{(divide)})</td>
</tr>
<tr>
<td>Executing stored command strings</td>
<td>addr;E &lt;ret&gt;</td>
<td>(80000E58;E)</td>
</tr>
<tr>
<td>Change display mode</td>
<td>[B W L &quot; I]</td>
<td>byte width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>longword width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASCII display</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Instruction display</td>
</tr>
</tbody>
</table>
DEBUGGING

Debugging SDA

To activate SDA:

$ ANALYZE/qualifier

What you use as a qualifier will determine what you will be examining.

<table>
<thead>
<tr>
<th>To Examine</th>
<th>Qualifier</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current System</td>
<td>/SYSTEM</td>
<td>CMKRNRL privilege needed</td>
</tr>
<tr>
<td>System Dump file or</td>
<td>/CRASH_DUMP</td>
<td>Read access to file needed</td>
</tr>
<tr>
<td>Other Dump file</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SDA Functions

- examine locations by address or symbol
- displays process/system data
- formats and displays data known data structures
- assigns values to symbols as requested

Command Format

command [parameter] [/qualifier]
### Table 7-4 SDA Commands

<table>
<thead>
<tr>
<th>Information</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides help using SDA</td>
<td>HELP</td>
</tr>
<tr>
<td>Displays specific data/information</td>
<td>SHOW</td>
</tr>
<tr>
<td>Format and display data structures</td>
<td>FORMAT</td>
</tr>
<tr>
<td>Display contents of location(s)</td>
<td>EXAMINE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manipulation</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preserve second copy of dump file</td>
<td>COPY</td>
</tr>
<tr>
<td>Create and define symbols</td>
<td>DEFINE</td>
</tr>
<tr>
<td>Perform computations</td>
<td>EVALUATE</td>
</tr>
<tr>
<td>Set/Reset defaults</td>
<td>SET</td>
</tr>
<tr>
<td>Define other VMS symbols</td>
<td>READ</td>
</tr>
<tr>
<td>Repeat last command</td>
<td>REPEAT or</td>
</tr>
<tr>
<td></td>
<td>&lt;ESC&gt;</td>
</tr>
</tbody>
</table>
### DEBUGGING

**SDA> SHOW DEVICE TTC7**

#### DDB LIST

<table>
<thead>
<tr>
<th>Address</th>
<th>Controller</th>
<th>ACP</th>
<th>Driver</th>
<th>DPT</th>
<th>DPT size</th>
</tr>
</thead>
<tbody>
<tr>
<td>8009VA20</td>
<td>TTC</td>
<td>DZDRIVER</td>
<td>600788F0</td>
<td>1A46</td>
<td></td>
</tr>
</tbody>
</table>

**Controller:** TTC  

---

**Device Data Block (DDB):**
- DDT address: 80078DE8
- First UCB address: 8007E450
- CRB address: 8009DA80

**Channel Request Block (CRB):**
- UCB reference count: 8
- Channel allocation mask: 00
- Secondary CRB address: 00000000
- IDB address: 8009D9E0
- Controller init. routine: 8007A27C
- Unit init. routine: 8007A2FF
- Unit start routine: 8007A406
- Unit disconnect routine: 8007A35F

**Interrupt Dispatch Block (IDB):**
- CSR address: 80017A68
- Owner UCB address: 00000000
- Number of units: 8
- ADP address: 80077200

**Driver Dispatch Table (DDT):**
- Start I/O routine: 0000049B
- Unsol. interrupt routine: 8000981B
- Function Decision table: 00000020
- Cancel I/O routine: 0000010E
- Register done routine: 8000981B
- Diagnostic buffer size: 0000
- Error buffer size: 0000

**TTC7**

---

- UCB address: 8007E990
- Device status: 0110 onlinerbs
- Characteristics: 0C040007

<table>
<thead>
<tr>
<th>IRF address</th>
<th>Device class</th>
<th>IRF</th>
<th>Mode</th>
<th>Chain</th>
<th>Func</th>
<th>UCR</th>
<th>EFN</th>
<th>ASR</th>
<th>IDEB</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>800940C0</td>
<td>42</td>
<td>SFVN</td>
<td>0000000C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8009DA80</td>
<td>60</td>
<td>SVAFT</td>
<td>800838FC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>80</td>
<td>BOFF</td>
<td>0014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800024CB</td>
<td>18101380</td>
<td>BCNT</td>
<td>00C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800024CC8</td>
<td>0001</td>
<td>ERTCNT</td>
<td>1C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80074019</td>
<td>1</td>
<td>ERRCNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>1425</td>
<td>Owner UIC</td>
<td>0000:0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001A</td>
<td>00000000</td>
<td>PID</td>
<td>0002003c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**I/O request queue**

---

7-22
DEBUGGING

DRIVER TESTING HINTS

1. You will need to write one or more programs to test your driver. Often, it is convenient to write test programs in higher-level languages (such as FORTRAN). Your test programs must:

   • test all function-codes supported by your driver.
   • test all paths through your driver (including timeout, powerfail, and cancel I/O).
   • test all conditions relevant to your driver (e.g., error conditions, multiple program access, and high-speed access).

If possible, test your test programs on some other device (terminal) by changing the $ASSIGN system service to assign a channel to a terminal, rather than your device, in the DEVNAM argument. That way, you can debug only your driver, not both your driver and your test program. As always, it is good programming practice to document your test programs.

2. Be systematic in your approach. Develop FDT routines first. Test them by temporarily changing

   JMP G$EXE$QIODRVPKT

   to

   JMP G$EXE$FINISHIO

   In this way, you can develop sections of your driver at a time.

3. You may want to include a mask at the end of your Function Decision Table to transfer control to an error routine (to prevent FDT dispatching errors). The mask should specify all legal function codes.

4. If you plan on using the XDELTA stored command feature, leave room in your UCB for stored command strings.

5. Place a (commented) JSB G$INISBRK at the start of each driver routine, so that if you have to use XDELTA, you can quickly locate which routine is causing system crashes.

6. To test for powerfail recovery, turn off the power on the front cabinet by using the key.

7–23
DEBUGGING

7. Observe your device while testing for clues as to how far an I/O request got before the system crashed. Was the device powered-up?, did the device initiate some hardware activity?, did the system crash after the I/O completed?, etc.

If, while observing your device, your driver causes the device to behave incorrectly, or to loop (e.g., never stop punching tape), you may have to type control-P on the console (followed by HALT) to halt the processor. Or, if looping at too high an IPL for the console control-P interrupt to be granted, turn the power off using the key on the front cabinet. To force a dump file to be written, you can crash the system on the system console as follows:

type control-P
>>> HALT
>>> @CRASH

8. Specify a real CSR to SYSGEN when you CONNECT a unit unless you use the /NOADAPTER qualifier. If your hardware is not yet installed, specify an existing CSR (such as the lineprinter CSR, which will not be affected by a test to see if it is there). Also, a vector address that is not currently in use must be specified, even in the /NOADAPTER case.

9. When using XDELTA, set up a base register to point to the start address of your driver (plus the $$15115 DRIVER PSECT base address), so that you can reference instructions as offsets from the base register (corresponding to offsets in your driver listing file).

10. When calling a DIGITAL representative (field service, software support, etc.) for assistance in device driver or device hardware problems, it is always helpful to have a driver testing program available. Such a program can be used as a "first pass" diagnostic for your device driver or device hardware.
### Common Problems and Reasons When Writing Device Drivers

<table>
<thead>
<tr>
<th>Problem</th>
<th>Reason</th>
</tr>
</thead>
</table>
| Access violation | 1. **UCB$L...(R5)**  
Empty register  
2. Didn't save/restore registers before/after MOV C |
| NULL process "caught" in IOPOST routine | Didn't set buffered I/O bit IOPOST, tried to unlock "bad" pages |
| Process requesting I/O in MWAIT state (AST wait) | Didn't IOPORK before REQCOM |
| Random behavior not reproducible | 1. Using registers other than R3 R4 before/after IOPORK  
2. Leaving data on stack when calling system macros (causing fork process to wait) |
DEBUGGING

APPENDIX

SAMPLE STACK OUTPUTS

Figure 7-1 Reserved Operand
Figure 7-2  Access Violation
Figure 7-3 Pagefault Above IPL2
Figure 7-4  Machine Check in Kernel