BootBug

"the ideal tool for debugging boot-time code..."

Brigent, Inc.
684 Costigan Circle
Milpitas, California 95035-3366
(408) 956-0322 (voice & fax)
AppleLink: ScottC

Warranty

Brigent Incorporated (Brigent) warrants BootBug hardware against defects in material and workmanship for 90 Days from the original purchase date.

Brigent shall, at its own expense and option, either repair or replace the defective product during the warranty period, provided that the original purchaser has notified Brigent and, upon inspection by Brigent, Brigent has found the product defective.

Purchaser's sole and exclusive remedy hereunder shall be limited to the repair or replacement of the product.

Any misuse, abuse, modification, or tampering with serial numbers shall void this warranty.

The express warranties set forth herein are in lieu of all other warranties, expressed or implied, including without limitation, any warranties of merchantability or fitness for a particular purpose*.

In no event will Brigent be liable to the purchaser for damages, including any lost profits, lost savings or other incidental or consequential damages arising out of the use or inability to use such product even if Brigent has been advised of the possibility of such damages, or for any claim for any other party. In any event, the liability of Brigent shall not exceed the purchase price of the product*.

* May not apply in some states.

Table of Contents

Introduction

What is BootBug?

BootBug is the ideal debugging tool for those who develop Nubus Cards, SCSI Drivers, Accelerators, or any other boot time code. BootBug is a Nubus card that contains a full featured MacsBug style debugger in its configuration ROM and a serial port to talk to an external terminal.

How Does BootBug Work?

The BootBug card is installed in a specific Nubus slot of your Macintosh computer. When the system starts up, BootBug's software will be the first software loaded into the system. The software takes over as the systems debugger and allows you to debug any software that loads afterwards.

About the Manual

This manual assumes you are familiar with low level debugging on the Macintosh using debuggers such as MacsBug, TMON, or The Debugger. If you are not familiar with low level debugging, Chapters 1 through 7 of <u>MacsBug Reference and Debugging Guide</u> cover the topic very well.

The manual contains the following chapters:

1. Installation
gives you step by step instructions for installing BootBug. If you are familiar with the MacsBug command set, this chapter may be all you need.

2. How BootBug Works
describes the boot process, when BootBug loads, and when your software loads.

3. BootBug Debugging Tips
gives you a few pointers (or a handle) about debugging SCSI drivers, primary inits, and card drivers.

4. BootBug Command Reference
is a full description of each of BootBug's commands.

5. BootBug Communication Software
gives descriptions of the free communication software supplied with BootBug.

Notations

The following notations are used to describe BootBug commands:

`Bold courier`	signifies text that is typed by the user.
`Plain courier`	signifies text that is displayed by BootBug.
Italic text	signifies parameters that must be replaced with data.
[ ]	bracketed items are optional.
|	vertical bar signifies an either/or choice.

Chapter 1 Installation

To use BootBug, you will need the following equipment:

- NuBus equipped Macintosh.
- A second Macintosh or other computer equipped with a serial port that can be used as a 9600 baud terminal.
- BootBug card.
- BootBug serial cable or other serial cable.
- BootBug communication software or your own communication software.

Using a Macintosh as the Terminal

Step 1 **Installing the BootBug Card**

The BootBug card needs to be installed in the Macintosh that runs the software you want to debug. BOOTBUG MUST BE IN A SPECIFIC SLOT TO WORK PROPERLY. The slot varies depending on the model of Macintosh you have.

Model	Slot#	Slot Position from Front
Mac II	9	Left Most Slot
Mac IIx	9	Left Most Slot
Mac IIcx	9	Left Most Slot
Mac IIci	E	Right Most Slot
Mac IIfx	E	Right Most Slot
Mac IIsi	C	In NuBus Extender
Quadra700	D	Left Most Slot
Quadra 900	9	Upper Most Slot
Quadra 950	9	Upper Most Slot

Be sure that the computer is off and that you have grounded yourself to the metal power supply case.

If you are not familiar with installing Nubus cards, please refer to the Setting Up booklet that came with your Macintosh.

Step 2 **Connecting the Serial Cable**

Plug the 9 pin D connector end of the cable into the BootBug connector at the back of the card. Plug the 8 pin DIN connector into the modem port of the Macintosh you plan to use as a terminal.

Step 3 **Installing the Terminal Software**

Simply insert the BootBug floppy into the Macintosh you plan to use as a terminal, drag the "Terminal" application and the "Terminal Folder" onto your hard disk, and launch the program. Besure that the application and the folder are in the same folder.

The program is already set to 9600 baud 8 bits of data, 1 stop bit, no parity, and XON/XOFF handshaking using the modem port. If you want to use your own communication software, be sure to configure it to these parameters.

Step 4 **Booting BootBug!**

Now that everything is installed and the communication software is running on the Macintosh being used as the terminal, it is time to turn on the Macintosh that contains BootBug. After turning on the Mac, you will hear the normal startup chime. Then, the BootBug message will be displayed on the terminal software.

The message will tell you that you have some number of seconds left to press a key in order to invoke BootBug. If you do not press a key, BootBug will let the Mac finish booting. In both cases, BootBug will be installed in memory.

After you have invoked BootBug, you can begin setting breakpoints, setting A-Trap breaks, single stepping, and so on. Refer to the BootBug Command Reference chapter for a complete description of the commands and the Using BootBug chapter for ideas on how to debug your software.

Using Another Computer as the Terminal

If you wish to use a PC, Apple II , VDT, or other type of device as the terminal, you will have to supply the serial cable and any terminal software yourself.

The serial port on the BootBug card has the same configuration as a PC type serial port.

Pin	Signal
1	Carrier Detect
2	Receive Data
3	Transmit Data
4	Data Terminal Ready
5	Ground
6	Data Set Ready
7	Request to Send
8	Clear to Send
9	Ring Indicator

For example, to connect BootBug to a PC serial port, you will need two 9 to 25 pin serial cables (male both ends), a Null Modem adapter, and a gender changer.

The communication software you use must be set to 9600 baud, 8 bits of data, 1 stop bit, and no parity. XON/XOFF handshaking helps, but is not required.

After you have configured the cable and software, follow steps 1 and 4 above.

Chapter 2 How BootBug Works

The Boot Process

BootBug loads before any other software because the Macintosh ROMs load and execute the Primary Inits of NuBus cards very early in the boot process. Since BootBug is installed in the first slot that the ROMs look at, BootBug loads before all other Primary Inits and drivers.

Understanding the sequence of the boot process is important to understanding how your software is loaded. Below is the summarized sequence of the boot process:

- First, the system performs several self-tests, sounds the startup chime, checks memory, and does basic intialization of the IO chips.
- Next, the system starts to initialize low memory globals, the Memory Manager, the Trap Dispatcher, and the interrupt dispatch tables. Although initialized, interrupts have not been enabled yet.
- A system zone is created. Resource Manager, Time Manager, and ShutDown Manager are initialized.
- The system initializes the Slot Manager. This is the point at which **BootBug** and the rest of the Primary Inits are loaded and executed.
- The 60hz tick is initialized, and interrupts are enabled.
- The driver queue is initialized.
- The SCSI Manager is initialized, and a SCSIReset is issued.
- The Unit Table is initialized. The .Sony, .Sound, and the .SERD drivers are installed.
- If you are running a Mac II or Mac IIx, the ADB is initialized followed by the opening of the video driver for the startup screen. For Mac IIci machines and above, the order is reversed.
- The slot interrupts are enabled. The "Beep Screen" is drawn on the startup screen.

- The system starts to look for startup devices. It will search the SCSI chain, floppies, and the Nubus cards to find a device from which to boot the OS. During this process, the SCSI drivers will be loaded and opened.
- The system will then load and execute the boot blocks. This will open the system file and read in the boot resources.

For more information on the boot process, see the Start Manager chapter of the Inside Macintosh Volume V.

What BootBug Depends On

BootBug uses as little of the system software as possible so that it will not interfere with your software. However, there are a few basic system features BootBug needs to operate.

The most important is the SysError handler and its low memory globals, MacJmp and $BFF. MacJmp and $BFF tell the SysError handler to pass any exceptions it intercepts to BootBug.

BootBug also needs the low memory globals CPUFlag and SysZone during installation and the global MMU32Bit all the time.

If Virtual Memory is running, BootBug will use the DebugUtil trap to switch to supervisor mode and get the VM paging status.

BootBug is installed in the system heap. A second pointer is allocated in the system heap for BootBug's globals.

Chapter 3 BootBug Debugging Tips

Debugger Trap

The simplest way to debug your code is to place a _Debugger or _DebugStr trap. The _Debugger trap will simply invoke BootBug whenever it is encountered. The following assembly language code fragment demonstrates its use:

```
;...
_Debugger               ; Invoke BootBug Here
;...                    ; the rest of your program here
```

The _DebugStr trap will invoke BootBug, display a message, and optionally execute BootBug commands in the string. Your program puts the address of the pascal string on the stack and then executes the _DebugStr trap. A semi-colon in the string separates the message from the BootBug commands. The address of the string is removed from the stack by BootBug. Below is an assembly language code fragment using the _DebugStr:

```
        STRING PASCAL

        ;...
        pea     myStr           ; Put address of string on stack
        _DebugStr               ; Invoke BootBug which will then
                                ; display the 'Hello World' msg
                                ; and execute the dm A0 command.
        ;...

myStr   dc.b    'Hello World;DM A0'
```

Breaking on a Primary Init

Primary Inits are loaded from the declaration ROM by a call to the _SlotManager with D0 set to 5. This trap will load the Primary Init into memory and store its address at the address in A0. The address of the first instruction of the Primary Init will be @A0+8.

Assuming your card is in slot C, you would type the following commands to break just before your Primary Init is loaded:

```
ATB  SlotManager  (D0.w=5)  AND  (@(A0+31).b=C)
G
```

When BootBug is re-invoked, the PC will point to the SlotManager trap that will load your Primary Init. Use the Step Over command to load the code, then set a breakpoint at the beginning of the code (@A0+8), and Go.

```
SO
br  @A0+8
G
```

The next time BootBug is invoked, the PC will point to the first instruction of your Primary Init.

Breaking on a SCSI Driver

SCSI drivers generally call _DrvrInstall very early in their install routine. Putting an A-Trap break on _DrvrInstall will cause a break every time a SCSI driver tries to install itself onto the Unit Table. It will also cause a break on the _DrvrInstall trap in the _Open routine. You can distinguish the two since the _Open routine will be in ROM while the SCSI driver will be in RAM.

Chapter 4 BootBug Command Reference

Command Expressions

Many of BootBug's commands accept numeric and boolean expressions as parameters. Below is a description of the different operands and operators that make up these expressions:

MathOp	+, - , *, /, MOD, @,^, !, .b, .w
BooleanOp	=,!=,>,<,>=,<=,AND,OR,NOT,XOR
Value	hexidecimal number, global name, register name, or A-Trap name or number.
Expression	Value [MathOp value]
Boolean	Expression BooleanOp Expression

Entering an expression on the command line causes BootBug to display the result.

Example Assume: D0 = $12345678, A0 = $2000, and the long at address $2000 = $1FF94.

```
@SysZone
 = $00002000    #8192         #8192
D0.b * 4 + 2
 = $000001E0    #482          #482
D0.w AND F0F0
 = $00005070    #20592        #20592
@A0
 = $0001FF94    #130964       #130964
@(A0+2).w
 = $FFFFFF94    #4294967188 #-108
(2+3)*6 > 30
 = $00000000    #0            #0
D0.w = 5678
 = $00000001    #0            #0
```

ATB - A-Trap Break

Description The A-Trap Break command invokes BootBug whenever the cpu encounters the specified A-Trap.

Syntax ATB [*trap* [*trap*]] [*n* | *boolean*] [";*command*"]

trap
is the trap name or number of the trap. Two traps specifies a range of traps. If no traps are specified then BootBug will break on all A-Traps.

n
is the number of times the trap will be encountered before invoking BootBug.

boolean
is a boolean expression that must be true before BootBug will be invoked.

command
is a string of BootBug commands separated by semi-colons to be executed when BootBug is invoked.

Examples `ATB Open`
will invoke BootBug whenever the _Open traps is encountered.

`ATB Blockmove 4 ";dm @A0"`
will invoke BootBug after the fourth occurrence of _Blockmove and then dump memory at A0.

`ATB SlotManager D0.b=23`
will invoke BootBug when a _SlotManager is encountered and the byte in D0 = $23.

Also See ATC, ATD, ATSS

ATC - A-Trap Clear

Description A-Trap Clear command clears A-Trap breaks.

Syntax ATC [*trap* [*trap*]]

trap
is the trap name or number of the A-Trap break to clear. Two traps specify a range of A-Trap breaks to clear. If no traps are specified, all A-Trap breaks are cleared.

Examples `ATC Open`
clears an A-Trap break on the _Open trap.

`ATC Open KillIO`
clears A-Trap breaks on traps _Open, _Close, _Read, _Write, _Control, _Status, and _KillIO.

Also See ATB, ATD, ATSS

ATD - A-Trap Display

Description A-Trap Display shows all of the current A-Trap breaks.

Syntax ATD

Also See ATB, ATD, ATSS

ATSS - A-Trap Step Spy

Description

A-Trap Step Spy will compute a checksum on a range of memory whenever the specified trap is encountered and invokes BootBug if the checksum has changed.

Syntax

ATSS [*trap* [*trap*]] [*n* | *boolean*], *addr1* [*addr2*]

trap
is the trap name or number of the trap. Two traps specify a range of traps. If no traps are specified, then BootBug will break on all A-Traps.

n
is the number of times the trap will be encountered before invoking BootBug.

boolean
is a boolean expression that must be true before BootBug will compute the checksum.

addr1, addr2
specify the address range to checksum. If only addr1 is specified, then a long word at addr1 will be checksummed.

Examples

`ATSS _BlockMove, 1b440`
will checksum the long word at $1B440 whenever the _BlockMove trap is encountered, invoking BootBug if it has changed.

`ATSS _SlotManager D0.b=23,1b440`
will checksum $1B440-$1B800 when both the _SlotManager trap is encountered and the byte in D0 = $23, invoking BootBug if it has changed.

Notes

The system will slow down depending on the range size and the frequency of the specified traps.

BAUD - Set or Display Baud Rate

Description

BAUD sets or displays the communication baud rate for BootBug.

Syntax

BAUD [n]

n
is the baud rate to set. Valid baud rates are 1200, 2400, 4800, 9600, 19200, 38400, and 57600.

Examples

`BAUD`
will display the current baud rate.

`BAUD 19200`
will change the communication baud rate to 19200.

Note

You will need to change the baud rate of your communication software after entering this command.

The baud rate is stored in the Parameter RAM for the BootBug Nubus card. When the card is removed and then re-installed, the baud rate will revert to its default 9600 baud.

BR - Breakpoint

Description Breakpoint will invoke BootBug whenever the cpu encounters the specified address.

Syntax BR addr [*n* | *boolean*] [";*command*"]

addr
is the memory address of the breakpoint.

n
is the number of times the breakpoint will be encountered before invoking BootBug.

boolean
is a boolean that must be true before BootBug will be invoked.

command
is a string of BootBug commands separated by semicolons to be executed when BootBug is invoked.

Examples

`BR PC+10`
will invoke BootBug when the cpu executes the instruction at $10 bytes past the current PC address.

`BR 1b400 4`
will invoke BootBug the fourth time the cpu executes the instruction at $1B400.

`BR SlotManager D0.b=23 ";dm A0"`
will invoke BootBug when both the cpu executes the first instruction of the SlotManager trap and D0 = $23, and then display memory at A0.

Note If the breakpoint address is in ROM, BootBug will have to single step each instruction until the breakpoint is reached. This will significantly slow down the system.

Also See BRC, BRD

BRC - Breakpoint Clear

Description Breakpoint Clear clears one or all breakpoints.

Syntax BR [addr]

addr
is the memory address of the breakpoint to be cleared. If addr is omitted, then all breakpoints are cleared.

Examples **BRC 1b400**
will clear the break point at $1B400.

BRC
will clear all breakpoints.

Also See BR, BRD

BRD - Breakpoint Display

Description Breakpoint Display will display all of the breakpoints.

Syntax BRD

Also See BR, BRC

CS - Checksum

Description

Checksum records the checksum of a range of memory or checks to see if the checksum has changed since the last Checksum command.

Syntax

CS [*addr* [*addr*]]

addr
specifies the address range to be checksummed. If only one addr is specified, then the long word at addr will be checksummed. If no addr is specified, then the checksum of the last checksum address ranges is recalculated and compared to the previous result.

Examples

```
CS 4000 5000
CS
Checksum is the same.
SB 4000 ff
CS
Checksum has changed.
```

DB - Display Byte

Description Display Byte shows the byte at a specified address in hexidecimal, unsigned decimal, and signed decimal.

Syntax DB [*addr*]

addr
is the address of the byte to be displayed. If no addr, then the byte at the dot address is displayed.

Examples `DB 4500`
will display the byte at address $4500 in hex, unsigned decimal, decimal, and ASCII.

Note If you press return after the DB command, BootBug will display the byte at the next address.

Also See DW, DL, DM

DL - Display Long

Description Display Long shows the long at a specified address in hexidecimal, unsigned decimal, and signed decimal.

Syntax DL[*addr*]

addr
is the address of the long to be displayed. If no addr, then the long at the dot address is displayed.

Examples `DL 4500`
will display the long at address $4500 in hex, unsigned decimal, decimal, and ASCII.

Note If you press return after the DL command, BootBug will display the long at the next address.

Also See DB, DW, DM

DM - Display Memory

Description Display Memory at the specified address.

Syntax DM [*addr* [*n*]]

addr
specifies the address of the memory to display. If no addr is specified, then the memory at the dot address is displayed.

n
specifies how many bytes to display. If omitted then 16 bytes will be displayed.

Examples `DM @SysZone`
will display 16 bytes at the address stored at SysZone.

Note If you press return after the DM command, BootBug will display memory at the next address.

Also See DB, DW, DL

DW - Display Word

Description
Display Word shows the word at a specified address in hexidecimal, unsigned decimal, and signed decimal.

Syntax
DW [*addr*]

addr
is the address of the word to be displayed. If no addr, then the word at the dot address is displayed.

Examples
`DW 4500`
will display the word at address $4500 in hex, unsigned decimal, decimal, and ASCII.

Note
If you press return after the DW command, BootBug will display the word at the next address.

Also See
DB, DL, DM

DRVR - Display Driver Information

Description Display Driver Information displays the reference number, name, flags, head, storage, window, delay, driver address, and DCE address for each driver in the Unit Table.

Syntax DRVR

F - Find

Description Find will search an address range until it finds the specified value of the string.

Syntax F *addr nbytes value* | "*string* "

addr
is the starting address of the memory to be searched.

nbytes
is the number of bytes in the memory range (search from addr to addr+nBytes).

value
is the byte word or long to search for.

string
is the quoted string to search for.

Examples `F E00 1000 40809AE6`
will cause BootBug to search from $E00 to $1E00 looking $40809AE6.

Note If you press return after the F command is complete ,BootBug will continue searching from the address after the last find.

G - Go

Description Go exits BootBug and resumes program execution.

Syntax G [*addr*]

addr
specifies the address to resume execution at. If addr is not given, then execution resumes at the PC address.

Examples

```
G
```

will continue execution at the PC.

```
G  1b440
```

will resume execution at address $1B440.

Also See GT

GT - Go Until

Description	Go Until exits BootBug, resumes program execution at the PC, and will invoke BootBug when the specified address is reached. This is shorthand for setting a breakpoint (BR), exiting BootBug (G), and then clearing the breakpoint (BRC).
Syntax	GT *addr* [";command"] *addr* is the address where execution will stop and BootBug will be invoked. *command* is a string of BootBug commands separated by semicolons to be executed when BootBug is invoked.
Examples	`GT PC+10` will resume execution until it reaches the address PC+10.
Also See	G

HOW - How BootBug Was Invoked

Description How will restate the reason BootBug was last invoked.

Syntax HOW

Examples

```
HOW
User break at 1B440.
```

ID - Instruction Disassembly

Description	Instruction Disassembly displays the disassembly of one instruction.
Syntax	ID [*addr*] *addr* is the address of the instruction to disassemble. If not specified, the PC address will be used.

Examples	`ID`
	will disassemble one instruction at the PC.
	
	`ID 1b440`
	will disassemble one instruction at the address $1B440.

Note	If you press return after the ID command has been completed BootBug, will disassemble the next instruction.
Also See	IL, IP

IL - Instruction List

Description Instruction List displays the disassembly of ten instructions.

Syntax IL [*addr*]

addr
is the address of the first instruction to disassemble. If not specified, the PC address will be used.

Examples **IP**
will disassemble ten instructions starting at the PC address.

IP 1b440
will disassemble ten instructions starting at address $1B440.

Note If you press return after the IL command has been completed, BootBug will disassemble the next ten instructions.

Also See ID, IP

IP - Disassemble Around

Description	Disassemble Around will display the disassembly of ten instructions around an address.
Syntax	IP [*addr*] *addr* is the address of the instruction to disassemble around. If not specified, the PC address will be used.
Examples	**IP** will disassemble ten instructions around the PC address. **IP 1b440** will disassemble ten instructions around address $1B440.
Note	If you press return after the IP command has been completed, BootBug will disassemble the next ten instructions.
Also See	ID, IL

R - Display Register

Description	Register displays the general cpu registers and disassembles the instruction at the PC. This command is automatically executed each time BootBug is invoked.
Syntax	R
Also See	TD

RB - Reboot

Description	Reboot command will unmount the startup volume and then restart the system.
Syntax	RB
Also See	RS

Register Name

Description Register allows you to display or set the value of specific cpu registers.

Syntax *registerName* [= *expression*]

registerName
is one of the cpu's registers: D0-D7, A0-D7, SR, PC, ISP, MSP, USP, VBR, CACR.

expression
is the value that will be assigned to the register.

Examples

```
D0 = 1234
```

assigns 1234 to D0.

```
vbr
```

will display the contents of the VBR register.

Also See R, TD

RS - Restart

Description	Restart unmounts all volumes and then restarts the system.
Syntax	RS
Also See	RB

S - Step

Description Step executes the instruction at the PC and then invokes BootBug with the PC pointing to the next instruction. If the instruction at the PC is an A-Trap, the next instruction will be the first instruction of the A-Trap routine.

Syntax S

Note Control - S is the equivalent of S and then Return.

Also See SO

SC6 - A6 Stack Crawl

Description

A6 Stack Crawl displays the A6 frame addresses and possible return addresses from the stack.

Syntax

SC6 [*addr* [*nbytes*]]

addr
is the address of the first A6 stack frame. If not specified, then A6 will be used.

nbytes
addr+nbytes specifies the upper limit of the stack. If not specified, then @CurStackBase is used.

Examples

```
sc6
```

will display all of the A6 frame and return addresses from A6 to @CurStackBase.

```
sc6  278d40  2000
```

will display all of the A6 stack frames and return addresses from address $278D40 to $27AD40.

Note

If A6 does not point to an A6 Stack frame, then nothing will be displayed.

SC is equivalent to SC6.

Also See

SC7

SC7 - A7 Stack Crawl

Description A7 Stack Crawl displays all of the possible return addresses on the stack.

Syntax SC7 [*addr* [*nbytes*]]

addr
is the base of the stack. If not specified, A7 will be used.

nbytes
addr+nbytes specifies the upper limit of the stack. If not specified, then @CurStackBase is used.

Examples SC7
will display all of the possible return addresses from A7 to @CurStackBase.

`SC7 278d40 2000`
will display all of the possible return addresses from address $278D40 to $27AD40.

Note A7 Stack Crawl will make its best guess about what is a return address, but is not always correct. The user should verify the addresses that are displayed.

Also See SC6

SB - Set Byte

Description Set Byte assigns values to one or more bytes starting at a specified address.

Syntax SB *addr* *expression* [*expression*]

addr
is the starting address of the bytes to be set.

expression
is one or more byte values to assign.

Examples `SB 1b440 40 41 42`
will set the byte at $1B440 to $40, $1B441 to $41, and $1B442 to $42.

Also See SW, SL

SL - Set Long

Description Set Long assigns values to one or more longs starting at a specified address.

Syntax SL *addr* *expression* [*expression*]

addr
is the starting address of the longs to be set.

expression
is one or more long values to assign.

Examples `SL 1b440 11111111 22222222`
will set the long at $1B440 to $11111111 and $1B444 to $22222222.

Also See SB, SW

SO - Step Over

Description	Step Over executes one instruction at the PC and then invokes BootBug. If the instruction at the PC is a JSR, BSR, or an A-Trap, Step Over will execute the entire subroutine. The PC will point to the instruction after the JSR, BSR, or A-Trap.
Syntax	SO
Note	T and Control - T are equivalent to SO. When stepping over an A-Trap instruction, all A-Trap breaks are temporarily disabled.
Also See	S

SS - Step Spy

Description Step Spy checksums a range of memory before the execution of each instruction. If the checksum changes, BootBug will be invoked.

Syntax SS *addr1* [*addr2*]

addr1, addr2
specifies the address range to checksum. If only addr1 is specified, then a long word at addr1 will be checksummed.

Examples **SS CurStackBase**
will checksum the long at CurStackBase before executing each instruction and will invoke BootBug if it changes.

SS 1b440 1b500
will checksum the address range $1B440 to $1B500 and will invoke BootBug if it changes.

Note Step Spy will significantly slow down the system. It is recommended that Step Spy be used only while debugging a small section of code.

If you press return after Step Spy has caused BootBug to be invoked, the Step Spy will checksum the same range again.

Also See ATSS

STARTUP - Set Startup Mode

Description Startup sets the BootBug startup mode. When BootBug is first loaded, it can automatically: a) be invoked, b) allow the user some number of seconds to press a key before being invoked or not, or c) just continue the boot process without invoking BootBug.

Syntax STARTUP [*mode*]

mode
is the startup mode. Valid modes are ALWAYS, NEVER, or a number which specifies the number of seconds to allow the user to press a key. If no mode is specified, Startup will display the current startup mode.

Examples

```
STARTUP  ALWAYS
```

will cause the BootBug to always be invoked after startup.

```
STARTUP  NEVER
```

will cause BootBug not to be automatically invoked after startup and to continue the boot process.

```
STARTUP  4
```

will cause BootBug to wait four seconds for the user to press a key. If a key is pressed, BootBug will be invoked. If no key is pressed, the boot process will continue.

Note The startup mode is stored in the Parameter RAM for the BootBug Nubus card. When the card is removed and then re-installed, the Startup mode will revert to its default of waiting 10 seconds.

SW - Set Word

Description Set Word assigns values to one or more words starting at a specified address.

Syntax SW *addr* *expression* [*expression*]

addr
is the starting address of the words to be set.

expression
is one or more word values to assign.

Examples `sw 1b440 1111 2222`
will set the word at $1B440 to $1111 and $1B442 to $2222.

Also See SB, SL

TD - Total Display of Register

Description Total Display will show the values of all of the cpu's registers.

Syntax TD

Examples TD

```
D0 = 00000001   A0 = 0001B440    USP = 00000000
D1 = FFFF0000   A1 = 00000000    MSP = 00000000
D2 = 001B4400   A2 = 00000000    ISP = 0027E4C0
D3 = 00000000   A3 = 00000000    VBR = 00000000
D4 = 00000000   A4 = 001B434C   CACR = 00003111
D5 = 00443E40   A5 = 00000000
D6 = 00000000   A6 = 00000000     PC = 4080274E
D7 = 00000007   A7 = 0027E4C0     SR = Sm7xnzvc
```

Also See R

WH - Where

Description Where displays information about the location of an address or A-Trap.

Syntax WH [*addr* | *trap*]

addr
is an address to display information about.

trap
is an A-Trap name or number to display information about. If no trap or addr is specified, then Where will display information about the PC address.

Examples

```
WH  Open
Open (A000) = 00098E42

WH  1b440
0001B440 = Main+004E
```

Chapter 5 Terminal Program

The terminal program included in the BootBug package is a publicdomain general purpose communication program. Its features include: file transfer, scripting, macros, and text capture. BootBug only requires a subset of these features which are documented below. However, if you are interested in using the other features of the terminal program, a complete manual is on the BootBug disk in the terminal folder.

Terminal Files and Folders

The application file, "Terminal", can be located anywhere on your disk. The associated data files, "Terminal Settings" and ".m" macro files, must either be in the same folder as the application or in a folder titled "Terminal Folder" located in the same folder as the application.

Terminal Communication Settings

Selecting the "Communication" item under the "Options" menu will display a dialog box which allows you to change the communication settings. The default settings are set to the correct values for BootBug operation. They are as follows:

BAUD	9600
Data Bits	8
Stop Bits	1
Parity	None
Handshaking	XON/XOFF

If you change the BootBug baud rate using the BAUD command (see Command Reference Chapter 4), you will also need to change the terminal applications baud rate.

Terminal Settings

Selecting the "Terminal" item under the "Options" menu will display a dialog box which allows you to change the TTY terminal emulation settings. The default settings are set to the correct values for BootBug operation. They are as follows:

Local Echo	Disabled
Remote Echo	Disabled
Display	Enabled
Capture	Enabled
AutoLF	Disabled
Startup Script	Disabled

These values should not be changed for operation with BootBug.

Saving BootBug Output to Disk

There are two ways to save BootBug output to disk. The first is to use the "Save capture buffer..." item under the "File" menu. This will present a standard file dialog box allowing you to save the contents of the capture buffer (the last 32k) to a file.

The second way is to select the "Text Capture" item under the "File" menu. This will present a standard file dialog box allowing you to save all further output to a file. After Text Capture is activated, the Text Capture menu item will appear in an outline font to indicate that it is active. To stop the Text Capture, select the Text Capture menu item again.

Holding down a modifier key (shift, option, command) while selecting Save Capture Buffer or Text Capture will append the data to the selected file.

Other Terminal Settings

The file creator type of the output files can be set by selecting the "Other..." item under the "Options" menu. This will display a dialog box that will allow you to change the creator type.

The same dialog box will allow you to change the modifier key used to create control characters. For example, selecting the command key will feel very comfortable to MacsBug users who are used to command-S for step and command-T for step over, etc.

Appendix A TechNotes

Below is a list of special circumstances where BootBug will behave differently.

Mode32	Mode32 from Connectix must reboot the system in order to install its 32 bit memory manager. This causes BootBug to be loaded twice, first under a 24 bit system and then under a 32 bit clean system. BootBug will not remember the breakpoints or A-Trap break points during this reboot. Also, be aware that if you are debugging a Primary Init, card driver, or SCSI driver, your software will also be loaded twice.
Rocket	Rocket from Radius, Inc. also reboots the system while loading, but in a slightly different way from Mode 32. The system first boots on the motherboard normally. BootBug will be loaded as well as Primary Inits and drivers. Once Rocket takes over, it will reboot the system using Rocket's cpu and memory. BootBug, the Primary Inits, and the drivers will be reloaded on Rocket. In contrast to Mode 32's reboot, the first instance of BootBug which was installed on the motherboard is still active. If you set breakpoints within the first instance of BootBug, they may be triggered by the Rocket software on the motherboard while the rest of the system is running on Rocket's cpu. In fact, if both BootBugs (the motherboard instance and the Rocket instance) are invoked at the same time, they will conflict with each other's access to the BootBug hardware.
MacsBug	If MacsBug is in the system folder it will replace BootBug as the system debugger. In general, the handoff works fairly well. The exception is when there are A-Trap breaks set when BootBug loads. When and if an A-Trap break is encountered, it will

invoke BootBug. Once in BootBug, execute an ATC and then a G.

If you do not want MacsBug to replace BootBug, remove it from the system folder.

Appendix B Trouble Shooting

This appendix contains information to help you trouble shoot problems you may have with BootBug. First, it is important to know about the two status LED's on the BootBug card. They indicate the current state of BootBug. The meanings are as follows:

RED is off and GREEN is on.

BootBug is installed. The system is executing code.

RED is on and GREEN is off.

BootBug is installed. BootBug has been invoked by a breakpoint, A-Trap break, bus error, NMI, etc.

Both RED and GREEN are on.

This indicates that BootBug was not recognized by the system and was not installed. Note, however, when the BootBug is executing a Step Spy command it will be switching between the RED on, GREEN off state and the RED off, GREEN on state quickly. It may look as if both LED's are on. Of course, if you had a chance to set enter a Step Spy command, this means that BootBug was loaded.

Both RED and GREEN are off.

BootBug encountered a problem while trying to install itself. The most likely problem is low memory.

If You Do Not Get Any Messages from BootBug

Either BootBug is not getting installed, or the terminal is not getting the message.

First determine if BootBug is installed by looking at the LED's. If only one LED is on (RED or GREEN), then BootBug was installed.

If it was installed, make sure the terminal communications are okay by checking the serial cable, the settings of the terminal program, and the configuration of the serial port. Make sure any network software on the terminal computer is not interfering with the serial port. If you are using a Power Book , make sure the modem port is configured to external modem.

If none of this helps, make sure BootBug is set to its default configuration by turning off the Macintosh that contains BootBug, remove BootBug, turn the computer back on, turn it off again, and re-install BootBug. This will clear the Parameter RAM, force BootBug to return to its default of 9600 baud, and allow the user ten seconds to press a key.

If you determine that BootBug is not loaded by looking at the LED's, then there is probably a hardware problem with the BootBug card or one of the other cards in the system.

If you still are having trouble, contact us at :

Brigent, Inc.
684 Costigan Circle
Milpitas, California 95035-3366
(408) 956-0322
AppleLink: ScottC
Compuserve: 70672,1313

Appendix C BootBug Nubus Card

The BootBug Nubus card contains a configuration ROM, 16450 UART, and supporting logic. The 16450 UART is similar to the UART used in PC's. The card uses byte lane three. The address map for the card is:

Fss00000	Card Base
Fss00003	Transmit and Receive Holding Register
Fss00007	Interrupt Enable Register
Fss0000B	Interrupt Status Register
Fss0000F	Line Control Register
Fss00013	Modem Control Register
Fss00017	Line Status Register
Fss0001B	Modem Status Register
FssE0000	Base of ROM

The decloration ROM containts three sResources:

Board Contains the standard baord sResource data.

BootBug Contains an sBlock that holds the debugger code and data.

UART Contains the addresses of the 16450 UART. The minorBaseOS contains the slot base address for the first register of the UART and the minroLength contains the offset from one register to the next.

Index

BootBug

"the ideal tool for debugging boot-time code..."

Addendum to the Manual

Version 1.3

Brigent, Inc.
684 Costigan Circle
Milpitas, California 95035-3366
Phone: (408) 956-1234
Fax: (408) 956-0322
AppleLink: ScottC

Table of Contents

DM - Display Memory

Description Display Memory at the specified address.

Syntax DM [*addr* [*n* | *template* | *basic type*]]

addr
specifies the address of the memory to display. If no addr is specified, then the memory at the dot address is displayed.

n
specifies how many bytes to display. If omitted then 16 bytes will be displayed.

template
specifies named templates to use in formatting the display output.

basic type
specifies a named basic type to use in formatting the display output.

Examples `DM @SysZone`
will display 16 bytes at the address stored at SysZone.

`DM @SysZone Zone`
will display the system zone formatted as a zone data structure.

Note If you press return after the DM command, BootBug will display memory at the next address.

Also See DB, DL, DW, TMP

HC - Heap Check

Description	Heap Check will identify corruption in the heap zone header or any of the block headers in the current heap.
Syntax	HC
Also See	HD, HT, HX, HZ

HD - Heap Display

Description	Heap Display will display information about the blocks in the current heap.
Syntax	HD
Also See	HC, HT, HX, HZ

HT - Heap Totals

Description	Heap Totals will display information about the current heap.
Syntax	HT
Also See	HC, HD, HX, HZ

HX - Heap Exchange

Description Heap Exchange will select the current heap.

Syntax HX [*addr*]

addr
specifies the address of a heap zone. If the address is not specified, then the HX command will cycle through the available heaps.

Also See HC, HD, HT, HZ

HZ - Heap Zone

Description Heap Zone will list the starting and ending addresses of all known heaps.

Syntax HZ [*addr*]

addr
is the starting address of a heap containing embedded heaps.

Also See HC, HD, HT, HX

TMP - List Templates

Description TMP will display the names of the available templates.

Syntax TMP

Index