

DATE: 31 January 19

SOFTWARE SPECIFICATION

for

KA630-A CONSOLE PROGRAM

REVISION 2.2

Written By: A. Randall Heap MLO5-5/E71

Issued By: A. Randall Heap

Reviewed By: Jesse Lipcon MLO5-5/E71

Jay Nichols MLO5-5/E71

Kathy Morse ZK01-1/D42

C O M P A N Y C O N F I D E N T I A L

COPYRIGHT (c) 1983 BY
DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.

DATE: 31 January 1983

SOFTWARE SPECIFICATION

for

KA630-A CONSOLE PROGRAM

REVISION 2.2

Written By: A. Randall Heap MLO5-5/E71

Issued By: A. Randall Heap

Reviewed By: Jesse Lipcon MLO5-5/E71

Jay Nichols MLO5-5/E71

Kathy Morse ZK01-1/D42

C O M P A N Y C O N F I D E N T I A L

COPYRIGHT (c) 1983 BY
DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.

PREFACE

This is a Software Product Specification. It is the definitive description in measurable terms of the goals, capabilities, and external characteristics of the KA630 console program. It is the commitment of WHAT IS TO BE BUILT as agreed by the Product Team.

No changes are to be made to the product as described here, unless approved and documented as an amendment to this Specification.

Associated Documents:

1. The KA630 Processor Specification which describes the processor board of which this software product (in ROM) is a component.
2. The KA630 System Development Plan, which describes the development project to design, build, test, evaluate, and deliver this product.
3. The KA630 ROM Diagnostic Specification which describes the ROM based diagnostics which are part of this program.
4. DEC Standard 32, the VAX Architecture Standard, which includes the definition of the VAX console system (13-Sep-82 - Rev 3).
5. The VIDEO SYSTEM STANDARDS Reference Manual which defines the video terminal architecture for DEC terminals.
6. DEC Standard 138, "Registry of Control Functions for Character Imaging Devices".

Change History:

Date: 25-July-1983

Revision 1.0 Original KA630 phase zero version.

Date: 22-February-1984

Revision 1.15 Revisions for phase 1
VMB details and figures defined according to "MicroVAX CPU Technical Description", second draft, October 1983. This revision brings the specification up to phase 1 status. All subsequent revisions will be tracked by change bars in the left margin.

Revision 1.16 Revisions for phase 1 - continued
Spelling corrections, deleted vestigial text fragments

remaining from previous versions, a correction to the discussion of national replacement characters was made powerup mode was simplified, QVSS location was described, boot flag bits 0 and 7 were dropped.

Revision 1.17 Revisions for phase 1 - continued

Power-up modes redefined, section on diagnostic enlarged, number of console languages enlarged from 5 to 10, documentation of all TOY registers strictly internal to the console program was removed, console flag bit moved to CPMBX TOY register, CPFLG TOY register eliminated, power-up figures revised.

Revision 1.18 Revisions for phase 1 - continued

Revision of appendix a, LED and console output codes halt message texts converted to abbreviation so as to avoid requirements for translation.

Date: 22-Jun-1984

Revision 1.19 Revisions for delayed phase 1 - continued

Discussion of the power-up sequence and the overall organization of the beginning of the Software Capabilities section was revised. Wording around dis boot was modified.

Cleanup for a pre-phase-1 release.

Date: 18-Jul-1984

Revision 1.20 Converted processor name from KDQ32 to KA630, added Portuguese to the list of supported languages.

Date: 2-Aug-1984

Revision 1.21 Changed spec name from KA630 to KA630-A. Deleted vestigial documentation of "interrupt page" base console memory.

Date: 10-Sept-84

Revision 2.0 Text of user-friendly console messages revised and reduced. Post Phase 1 copy.

Digital Equipment Corporation - Software Specification

Date: 2-Nov-84

Revision 2.1 Rearrange test sequence, add QxSS information, change screen displays.

Date: 31 January 1985

Revision 2.2 Correct PROM boot definition. Correct language default alg. displays.

CONTENTS

1	PRODUCT SUMMARY	1
2	ENVIRONMENT	1
2.1	Users	1
2.2	Hardware	2
2.3	Software	2
2.4	Services	2
3	SOFTWARE CAPABILITIES	2
3.1	Power-up	3
3.1.1	Power-up Mode	3
3.1.2	Power Stabilization And ROM Checksum Checks	4
3.1.3	Console Program Initialization	4
3.1.4	Battery Backup Check	6
3.1.5	Test Interprocessor Communication Register	6
3.1.6	Determining The Console Terminal Type	6
3.1.6.1	QxSS Determination	6
3.1.6.2	Console Terminal Determination	7
3.1.7	Console Message Language Check	7
3.2	Entry/Dispatch	9
3.3	Diagnostics	10
3.4	Restart	11
3.5	Bootstrap	12
3.5.1	Primary Bootstrap Program (VMB)	13
3.5.1.1	Bootstrap Devices	16
3.5.1.2	Bootstrap Command Fl	16
3.5.1.3	Booting From Disk	17
3.5.1.4	Booting From Tape	19
3.5.1.5	Booting From PROM	20
3.5.1.6	Booting From DEQNA	20
3.5.1.7	Booting An Auxiliary Processor	21
3.5.2	Secondary Bootstrap Program	22
3.6	Console I/O Mode (System Halted)	25
3.6.1	Console Control Characters	25
3.6.2	Console Command Syntax	27
3.6.3	Console Command Keywords	27
3.6.4	References To Processor Registers And Memory	28
3.6.5	Console Commands	28
3.6.5.1	BOOT	28
3.6.5.2	CONTINUE	29
3.6.5.3	DEPOSIT	29
3.6.5.4	EXAMINE	31
3.6.5.5	FIND	32
3.6.5.6	INITIALIZE	32
3.6.5.7	HALT	32
3.6.5.8	REPEAT	33
3.6.5.9	START	33
3.6.5.10	TEST	33
3.6.5.11	UNJAM	33
3.6.5.12	Binary Load And Unload Command	34
3.6.5.13	Comment	35
3.6.6	Console Errors And Error Messages	35
3.6.7	Halts And Halt Messages	36

3.7	Program I/O Mode (System Running)	37
4	PUBLICATIONS	38
5	PACKAGING	38
6	INSTALLABILITY	38
7	EASE OF USE	38
8	PERFORMANCE	40
9	RELIABILITY	40
10	MAINTAINABILITY	41
11	MAINTENANCE	41
12	COMPATIBILITY	41
12.1	Product Compatibility	42
12.2	Standards Conformance	42
12.3	Internationalization	42
13	EVOLVABILITY	43
14	COSTS	43
15	TIMELINESS	43
16	CONSTRAINTS AND TRADEOFFS	44

APPENDIX A INTERPRETATION OF KA630 LED AND CONSOLE DISPLAY CODES

APPENDIX B SUMMARY OF I/O REGISTERS USED BY CONSOLE PROGRAM

B.1	BOOT AND DIAGNOSTIC CONFIGURATION REGISTER (BDR)	B-1
B.2	TIME OF YEAR (TOY) CLOCK REGISTERS	B-2
B.2.1	Console Program Mailbox (CPMBX)	B-2
B.3	SYSTEM IDENTIFICATION EXTENSION REGISTER (SIDEX)	B-3
B.4	CONSOLE TERMINAL REGISTERS	B-4
B.5	INTERPROCESSOR COMMUNICATION REGISTER (IPCR)	B-4

APPENDIX C QXSS AND APT

APPENDIX D OUTSTANDING ISSUES

FIGURES

1	Console Memory Map	5
2	User Language Prompt	8
3	QxSS Keyboard prompts	9
4	RPB Format	12
5	Bootblock Format	18
6	Extended RPB	23
7	Secondary Bootstrap Argument list	24
8	Secondary Bootstrap Memory Map	24
9	Console Display on Successful Power-Up	39
10	Console Display on Power-Up with Hard Errors	40

TABLES

1	Power-up Modes	3
2	Console Entry Decision Table	11
3	VMB Register Usage	11
4	VMB Bootstrap Command Flags	16
5	Console Error Messages	31
6	KA630 Halt Messages	31
A-1	KA630 LED Interpretation	A-1

1 PRODUCT SUMMARY

This document describes the KA630 console program. This program in conjunction with KA630 hardware implements the KA630 console system which gains control whenever KA630 "halts". For the KA630, "halting" means only that control is transferred to this program, not that the processor stops executing instructions.

When the console program is running, it provides services expected of a VAX-11 console system. In particular, the following services are available:

- o Automatic restart or bootstrap following processor halts or initial power-up.
- o An interactive command language allowing the user to examine and alter the state of the processor.
- o Diagnostic tests executed on power up that check out the CPU, the memory system and the Q22-bus map.
- o Support of video or hardcopy terminals as the console terminals as well as support of QVSS or QDSS based bitmapped terminals.

2 ENVIRONMENT

2.1 Users

Engineering, Manufacturing, Field Service and customers will use this program to test, configure and boot their system. This program is needed by these users both to provide an easy means to bootstrap a KA630 system and to detect and isolate system malfunctions.

Of these users, all but customers are assumed to be computer "sophisticates". While this will often be true of customers as well, no such assumption is made. Target customers include both sophisticated users who can use and understand all the features provided by the console program as well as unsophisticated users who know next to nothing about computers.

Users are not assumed to speak English. The console program may be directed to output all console messages in either English, French, German, Spanish, Italian, Norwegian, Dutch, Swedish, Finnish, Danish or Portuguese. If when the system powers up there is no language specified, it prompts the user for the language to use. The user language is recorded in battery backed up RAM so the preferred language is retained when the system is shut off.

2.2 Hardware

The console program is located in ROM on the KA630 processor board. The ROM address range is located in the KA630's local I/O space. The program uses the KA630 processor LEDs and console terminal output to communicate diagnostic progress and error reports to the user.

A console terminal is not required for operation but halts should not be enabled on a system without a console terminal. A QxSS (either QVSS or QDSS) based display can be substituted for the console terminal.

In order for the console program to operate, the processor must be functioning to the point of executing instructions from the console program ROM.

The KA630 decodes the ROM addresses so that the same ROM appears more than once in the address space. The console program is written in position independent code so that it may be executed from any address range and the KA630 uses this feature to selectively enable and disable the external halt circuitry. If the console program is executing in the range of 20040000 to 2004FFFF (hex), external halt conditions are ignored. If the console program is executing in the range of 20050000 to 2005FFFF (hex) external halt conditions are honored and the console program may be "halted" (whereupon it will immediately reenter at its beginning to process the halt).

The console program normally executes from the first address range while the diagnostics software normally executes from the second address range.

For more information on the processor hardware, consult the KA630 Processor Specification. See also appendix B for a summary of the hardware registers used by the console program.

2.3 Software

None - the console program runs standalone, it uses no other software products.

2.4 Services

None - the console program requires no support services.

3 SOFTWARE CAPABILITIES

The console program is divided into six major components plus the diagnostics. These components and their responsibilities are documented

in the sections that follow. Diagnostics are documented in "KA630 R Diagnostic Specification".

Note that throughout this document, the console program is often referred to simply as the "console".

The console program receives control whenever the processor halts whether because of an external halt signal, the execution of a halt instruction, a serious system error or because of power-up. On any halt condition, the processor switches to physical addressing, saves the PC, PSL and the ISP internally, encodes and saves the cause of the halt in a halt code and then branches to the start of the console program ROM. Upon entry, the console program always outputs the hex value "E" to the console LED's to indicate that at least one instruction has been executed. It then waits for power to stabilize by looping until BDR<15 is set.

When these common preliminaries are completed, the console program checks to see if the halt is a power-up halt or a halt for some other reason. If it is a power-up halt, it begins the power-up sequence described in section 3.1. If it is not a power-up halt, control passes to the entry and dispatch code described in section 3.2.

3.1 Power-up

Upon power-up, the console automatically performs a variety of one time operations. These operations initialize the processor and are needed only on power-up.

3.1.1 Power-up Mode -

On power-up, BDR<10:9> is interpreted as a power-up mode field. The interpretation of the power-up mode field is shown in table 1. As discussed in the sections that follow, several power-up operations are dependent on the power-up mode.

Table 1: Power-up Modes

Mode	Language Prompt	Diagnostics
0	Prompt for language only if TOY battery backup failed.	Run full diagnostics.
1	Prompt for language on every power-up.	Run full diagnostics.

Table 1 (cont.)

Mode	Language Prompt	Diagnostics
2	Set language to English.	Run console terminal loop-back tests.
3	Set language to English.	Run abbreviated diagnostics.

3.1.2 Power Stabilization And ROM Checksum Checks -

Following the application of power, the processor first performs hardware initialization and then control is transferred to the console program ROM code. The common console startup functions were described previously in section 3. Following their completion, on power-up control passes to the code described here.

Beginning the power-up processing, the console code outputs the hex value of "D" to the LED's indicating that the power stabilization wait is over. It then calculates a checksum of the console program ROM and checks it against the valid checksum stored in the ROM itself. If the computed checksum differs from the stored checksum, the code "hangs" in a loop. If the checksum is the same, the power-up code proceeds to the next step.

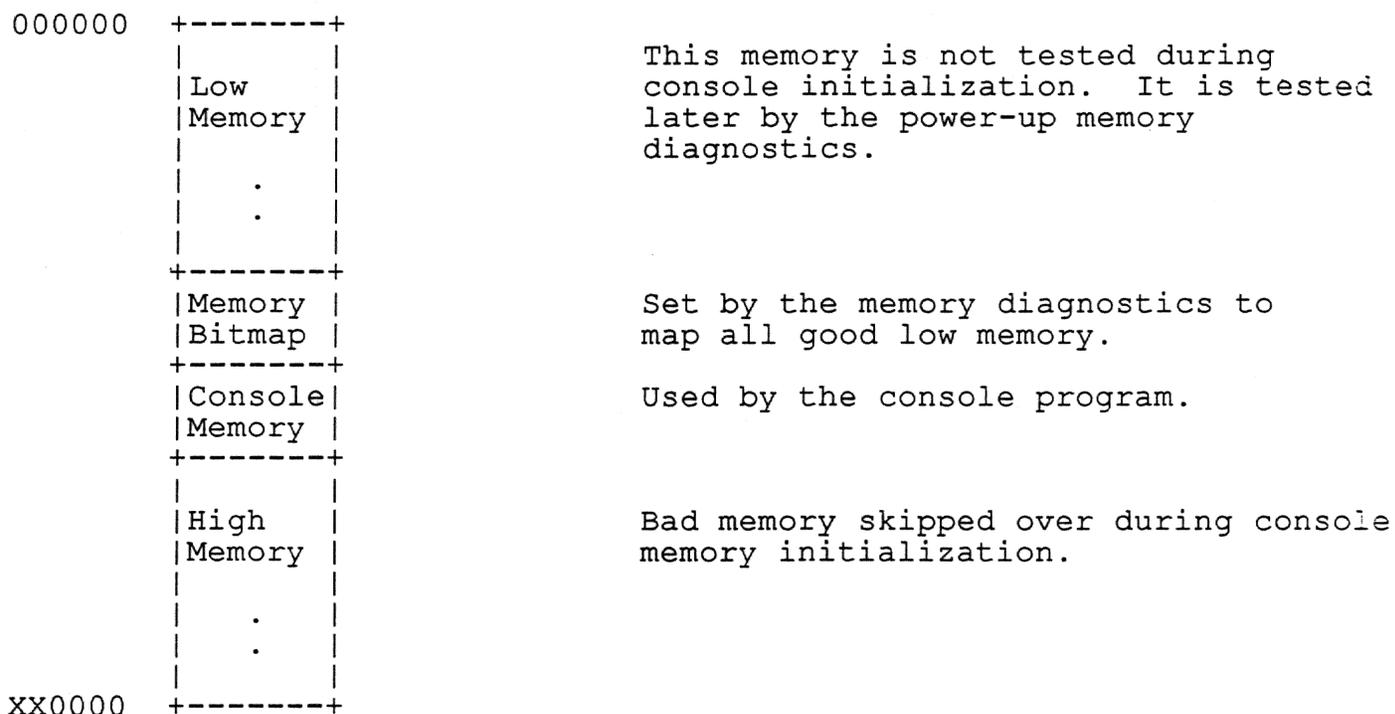
3.1.3 Console Program Initialization -

The next step of the power-up initialization is location and initialization of the memory needed for the console program itself. The hex value of "C" is output to the LED's at the beginning of this step.

During this step, the console ROM code searches top down through available memory for a small contiguous block that will be used by the console program for writeable storage. This block consists of two pages for the console's direct use and additional pages for use to store a bitmap of available memory. The amount of memory allocated to the bitmap varies according to the amount of memory available.

Following initialization, memory appears as shown in figure 1.

Figure 1: Console Memory Map



The console program memory is used by the console program for its stack and other data structures.

The bitmap is filled in later with a map of valid memory pages by the power-up memory diagnostics and this bitmap is passed to the bootstrap as a map of valid memory. Beginning from the base of the bitmap, the first bit corresponds to the first page of low memory, the second bit to the second page and so on. If the bit is set, the page is good, if the bit is clear, the page failed the memory test. The bitmap does not map itself nor any other memory that follows the bitmap. Since system software is expected to use only pages marked as good in the bitmap, system software is not expected to modify the bitmap or the console program memory. However, both the console program memory pages and the bitmap are checksummed by the console program to guard against accidental modification by the system software.

If the console cannot locate enough memory for its own use and the bitmap, it "hangs".

Following initialization of its memory, the console program clears CPMBX<1:0> (halt action), CPMBX<2> (bootstrap in progress flag) and CPMBX<3> (restart in progress flag).

3.1.4 Battery Backup Check -

The next thing done by the power-up operations is to check the TOY clock to see if the battery backup has failed. If this has happened, time of year has been lost along with the contents of all the TOY clock RAM. If the battery backup has failed, the console stops the TOY clock, zeroes the time and all TOY RAM and then initializes the TOY clock control and status registers. The operating system must check TOY clock control and status register B and determine if the clock is stopped to know if the TOY clock contains a valid time.

No change is made to the LED's during this operation.

3.1.5 Test Interprocessor Communication Register -

Next, the interprocessor communication register is tested. The hexadecimal value "B" is output to the LED's during this test.

This test is performed at this point as in addition to testing the register itself, this tests whether or not the Q22-bus is arbitrating properly. If it is not arbitrating, the processor will hang here. Hanging here is preferable to hanging in the next step where we check for QxSS video hardware on the bus. Were we to hang in the next step without first checking the Q22-bus, the resulting LED display would be ambiguous as to which failed, the Q22-bus or the QxSS.

3.1.6 Determining The Console Terminal Type -

3.1.6.1 QxSS Determination -

Next, if the processor is an arbiter processor, the console program checks for the presence or absence of a QxSS video display as the console device. Either a QVSS or a QDSS will be detected. If the processor is an auxiliary processor, this test is skipped. The hexadecimal value "A" is output to the LED's during this test.

The QxSS is determined by testing for the presence of its CSR address at tbs. (Both QVSS and QDSS use the same CSR address.) If there is no response, no QxSS is assumed present and the console program moves to the console terminal determination code which follows. If a QxSS is detected, it is initialized and a short diagnostic is executed. If the initialization and diagnostics succeed, the console will use the QxSS as its console terminal (however, see appendix C) and skips the next step and moves directly to the step described in section 3.1.7. If either the initialization or the diagnostic fails, the system hangs.

3.1.6.2 Console Terminal Determination -

If no QxSS is detected, it is assumed that normal a console terminal is connected to the console port or that NO terminal is connected. The console program then next attempts to determined the type of terminal connected. The value of "9" is output to the LED's during this test.

This determination is done by sending the terminal a device attribute request escape sequence. If the device responds with a recognizable response, the terminal is classified as a video terminal versus a hard copy terminal based on that response. This information is used when in console I/O mode to govern how command line editing is performed. The terminal must respond in 1 second to the device query. If no response or the response is not recognized, the test is repeated two more times or until the response is understood. If the device never responds or the response isn't recognized, the terminal is classified as a hard copy terminal.

Terminal response will be recognized in either eight bit or seven bit mode.

In addition to determining the presence or absence of a recognizable CP console, this information is also used to determine if the terminal supports the DEC multinational character set (MCS). The console program assumes that all new terminals, VT2xx and beyond support MCS. If the terminal does not support MCS, CPMBX<7:4> is set to 2, forcing English as the console display language.

3.1.7 Console Message Language Check -

The console next outputs a hex value of "8" to the LED's and then determines the appropriate language to use for all console messages. The algorithm used to determine the language appears below. The console language is stored in CPMBX<7:4> (see appendix B).

1. If power-up mode (see table 1) is 2 or 3, set the console language to English and exit.
2. If power-up mode is 0 or if the value of CPMBX<7:4> is zero solicit the language from the user. If the user doesn't respond within 30 seconds, set the language to English (3) and exit.

Remember that when the terminal is queried, if it isn't recognized as one that supports MCS, CPMBX<7:4> is set to 2, forcing English as the console language. English messages use the 7 bit subset of MCS. Also, if a loss of power for the TOY clock is detected, the contents of the TOY RAM is zeroed meaning that step 2 of the algorithm above will cause the user to be prompted.

The format of the user query issued in step 2 is shown in Figure 2.

Figure 2: User Language Prompt

- | | |
|------------|---------------|
| 1) Danish | 7) Dutch |
| 2) German | 8) Norwegian |
| 3) English | 9) Portuguese |
| 4) Spanish | 10) Finish |
| 5) French | 11) Swedish |
| 6) Italian | |

(1-11):

The text will be displayed using the DEC Multinational Character set extension.

The user must respond by entering a value of 1 through 11 following the question mark prompt, any other input will be rejected and the prompt line will be issued again. Normal console input line editing features (see section 3.6.1) will be available during input.

If the console program has determined that a QxSS video display system is being used as the console, an additional step is required. The QxS display system uses the DEC LK201 keyboard which comes in 16 national variants. The keyboard variant cannot be determined by querying the keyboard itself, it must be determined either from the language selected or by means of an additional menu selection. If French, German or English is specified as the language, the keyboard variant is ambiguous and the appropriate menu from figure 3 is displayed and the user is prompted to specify the which national keyboard variant is in use. If the user does not respond in 30 seconds, the last selection is assumed.

Figure 3: QxSS Keyboard prompts

-----French-----

- 1) Canada
- 2) France/Belgium
- 3) Switzerland

(1..3):

-----German-----

- 1) Germany/Austria
- 2) Switzerland

(1..2):

-----English-----

- 1) United Kingdom
- 2) United States/Canada

(1..2):

3.2 Entry/Dispatch

Following the determination of the console language on power-up, or directly on entry from any other halt condition, the console dispatcher to the appropriate code to service the halt.

To determine what actions to take, the console program examines the halt code, the halt enable bit (BDR<14>), the console program mailbox register (CPMBX) and then acts in accordance with decision table shown in table 2.

Table 2: Console Entry Decision Table

BDR(14)	Power-up	CPMBX(1:0)	Actions
F	T	X	Diagnostics, bootstrap, halt.
F	F	0	Restart, bootstrap, halt
F	F	1	Restart, halt.
F	F	2	Bootstrap, halt.
F	F	3	Halt
T	T	X	Diagnostics, halt.
T	F	0	Halt.
T	F	1	Restart, halt.
T	F	2	Bootstrap, halt.
T	F	3	Halt.

A power-up entry is one where the processor halt code is 3. "X" means "don't care" and the meanings of the other CPMBX codes are defined in appendix B, section B.2.1.

Multiple actions in table 1 mean that the first action is taken and if it fails and only if it fails, the next action is taken. Diagnostics are an exception. If diagnostics fail the console will "hang" without attempting to bootstrap the processor. If they succeed, then the next action is taken.

Note that because the KA630 does not support battery backup, it examines the halt code and does not attempt to perform restart operation following power-up.

3.3 Diagnostics

On power-up, the Entry/Dispatch code dispatches first to the diagnostic to check out the processor and memory before proceeding. Before doing this, the console outputs the message "Performing normal system tests." to the console terminal. As each test is run, a code is displayed on the processor LED's and output to the console terminal as well, causing a "countdown" to be displayed on both.

The first diagnostic LED code is 8 so executing the diagnostic continues the LED countdown. The diagnostic codes are documented in table A-1 along with all other codes. See figures 9 and 10 in section for console display formats.

At the conclusion of all tests, the message "Tests completed." is output to the console terminal.

If a diagnostic test detects a fatal error, an error message is displayed on the console, a summary message is displayed indicating that the continued operation is not possible and then the system "hangs" leaving the test code in the LED's. If halts are disabled, the only way

to clear the hung system is to turn the system off and then on again. If halts are enabled, the hung system may be cleared by manually halting it, causing it to enter console command mode.

Diagnostic software is fully documented in "KA630 ROM Diagnostic Specification".

3.4 Restart

The console can restart a halted operating system. To do so, the console searches system memory for the Restart Parameter Block (RPB), a data structure constructed for this purpose by the operating system. If a valid RPB is found, the console restarts the operating system at an address specified in the RPB.

The console keeps a "restart in progress" flag in the CPMBX register which it uses to avoid repeated attempts to restart a failing operating system. An additional "restart in progress" flag may be maintained by software in the RPB.

The console uses the following algorithm to restart the operating system:

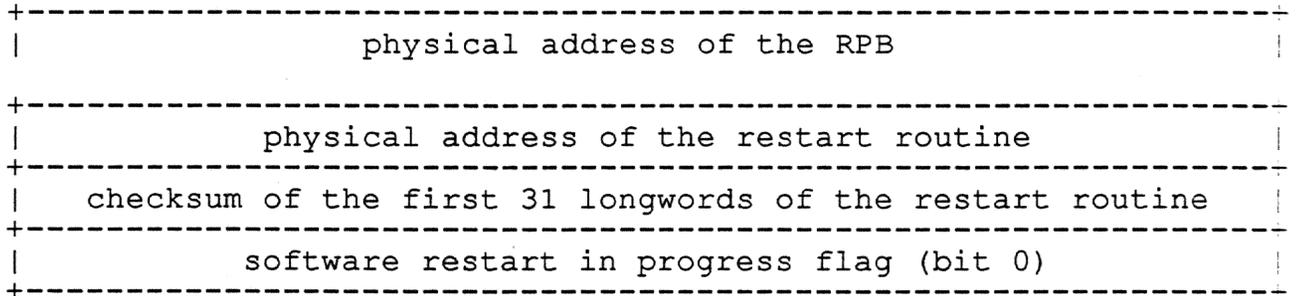
1. Check to see if the "restart in progress" flag in CPMBX is set. If so, restart fails.
2. Print the message "Restarting the operating system." on the console terminal.
3. Set the restart in progress flag.
4. Look for a Restart Parameter Block (RPB), left in memory by the operating system. If none is found, restart fails.
5. Read the software restart in progress flag from bit<0> of the fourth longword of the RPB. If it is set, restart fails.
6. Load SP with the address of the RPB plus 512.
7. Load AP with the halt code.
8. Display "0" in the console LED's.
9. Start the processor at the restart address, which is read from the second longword in the RPB.

If restart fails, the console prints "Attempt to restart operating system failed." on the console terminal. If the restart is successful, the operating system must clear the restart in progress flag in CPMBX (see appendix B for information on the CPMBX register).

The Restart Parameter Block is a page aligned control block, created by the operating system. Its format is shown in figure 4.

Figure 4: RPB Format

PB



Restart Parameter Block
(RPB)

The console uses the following algorithm to find a Restart Parameter Block:

1. Search for a page of memory that contains its address in the first longword. If none is found, the search for an RPB has failed.
2. Read the second longword in the page (the physical address of the restart routine). If it is not a valid physical address or if it is zero, return to step 1. The check for zero is necessary to ensure that a page of zeros does not pass the test for a valid RPB.
3. Calculate the 32 bit twos-complement sum (ignoring overflows of the first 31 longwords of the restart routine. If the sum does not match the third longword of the RPB, return to step 1.
4. A valid RPB has been found.

The same algorithm is used for both arbiter and auxiliary processors.

3.5 Bootstrap

The console can load and start (bootstrap) an operating system. To do so, it searches for a 64 kilobyte segment of correctly functioning system memory, sets SP equal to the base address of the segment plus 512 and then copies the primary bootstrap, called VMB, from the console program ROM to the segment starting at the location specified by SP. The console program then branches to the first location in VMB which then loads and starts the operating system.

To prevent a situation from occurring in which the console repeatedly tries and fails to bootstrap the operating system, it maintains "bootstrap in progress" flag in CPMBX. If a system bootstrap would occur automatically but the flag is already set, the console assumes that an attempt has already been made and has failed, so it does not try again.

The console uses the following algorithm to bootstrap the operating system:

1. If this bootstrap is the result of a console BOOT command, skip to step 4.
2. Check to see if the "bootstrap in progress" flag is set. If so, the bootstrap fails.
3. Print the message "Starting the operating system." on console terminal.
4. Set the bootstrap in progress flag.
5. Locate a page-aligned 64 kilobyte segment of good memory. If such a segment cannot be found, the bootstrap fails.
6. Initialize the Q22-bus I/O map as described below.
7. Load the general registers for the primary bootstrap program (VMB) as shown in table 3.
8. Copy VMB from the console ROM to 512 bytes past the base of the good segment.
9. Invoke VMB. If VMB fails, the bootstrap fails.

If bootstrap fails, the console prints "Attempt to start operating system failed." on the console terminal.

If the bootstrap is successful, the operating system must clear the bootstrap and restart in progress flags in the CPMBX register and clear the LED display by depositing a value of 0 in the BDR register (see appendix B).

Table 3: VMB Register Usage

Rn	Description
R0	ASCII device name (from BOOT command) or zero.
R1	Contents of KA630 Boot and Diagnostic Register
R2	Memory bitmap size in bytes.
R3	Address of memory bitmap.
R4	Unused.
R5	Software boot control flags (from BOOT command only).
R10	halt PC value.
R11	halt PSL value.
AP	halt code.
SP	512 bytes past base of 64 Kb of good memory.

During step 6 of the bootstrap process the Q22-bus I/O map is initialized using the algorithm shown below. The main function of this initialization is to preset the arbiter processor I/O map so that all unoccupied pages of the Q22-bus are mapped to the corresponding pages in the first 4 Mb of local memory. (This is a MicroVAX I compatibility feature). Note that this is not done for auxiliary processors, any auxiliary Q22-bus I/O mapping must be coordinated with all other processors so all auxiliary processor I/O map registers are marked invalid.

1. Turn on IPCR<8>, the halt flag.
2. Disable the I/O map by clearing IPCR<5>.
3. If the arbiter processor, do the following for each I/O map register:
 - a. Set the map register address bits to map the Q22-bus page to the corresponding local memory page.
 - b. If the corresponding Q22-bus page is unoccupied, turn on the valid bit.
 - c. If the page is occupied, turn off the valid bit.

If an auxiliary processor, turn off the valid bit in all map registers.

4. Enable the I/O map by setting IPCR<5>.
5. If an auxiliary processor, loop until the IPCR<8> bit is cleared.

Steps 1 and 5 are present to perform a secondary function while the Q22-bus I/O map is initialized, namely to synchronize an auxiliary processor with its bootstrap host. See section 3.5.1.7 for additional information.

3.5.1 Primary Bootstrap Program (VMB) -

VMB is the KA630 primary bootstrap. It is executed as the first part of a two part system bootstrap operation. VMB contains the code that:

1. initializes the system control block (SCB),
2. initializes an extended restart parameter block (RPB),
3. initializes a PFN (page frame number) bit map and the relevant extended RPB fields,
4. selects a bootstrap device, and
5. performs a Files-11 ODS2, boot block, ROM or down-line load of the secondary bootstrap.

The secondary bootstrap continues the bootstrap operation. For KA630 systems, primary bootstrap operations are defined by VMB, secondary bootstrap are defined by the operating system being booted.

VMB finds the bootstrap device in one of three ways:

1. If the bootstrap is the result of a console bootstrap command and a device name is specified in the command, that device is searched for the secondary bootstrap.
2. If not the result of a console bootstrap command or if no device name is specified, VMB searches first for a bootable disk, then for a TK50 (Maya) tape unit, then for MRV11 PROM, and finally for a DEQNA for a down-line bootstrap.
3. If the bootstrap is the result of a halt with CPMBX<1:0> equal to 2 - a request from the operating system to reboot the system - the bootstrap device used to previously bootstrap the operating system is used (as well as the same command flags).

When a VMB bootstrap attempt fails, it halts.

3.5.1.1 Bootstrap Devices -

The following bootstrap devices are supported:

1. RQDX, QDA and Aztec MSCP disk controllers. VMB can boot from any disk unit supported by a MSCP QD or Aztec disk controller. Units supported by RQDX are RX50, RD51, RD52 and RD53. Units supported by QDA are tbs. For purposes of the BOOTSTRAP command, units are designated DUA0, DUA1, and so on. The first controller must be configured at the Q22-bus address of 177215 (octal) and Q22-bus interrupt vector of 154 (octal). Additional controllers are located in floating CSR and vector space.
2. DEQNA Ethernet adapter. This controller connects to an Ethernet cable. For purposes of the BOOTSTRAP command, this device is designated XQA0. The controller must be configured at Q22-bus address 1774440 (octal) and Q22-bus vector of tbs.
3. MRV11 PROM. This is a Programmable Read Only Memory board. For purposes of the BOOTSTRAP command it is designated PRA0.
4. TMSCP tape controller (TK50). VMB can boot from any tape unit supported by the the TK50 TMSCP tape controller. For purposes of the BOOTSTRAP command it is designated tbs. The controller must be configured at Q22-bus address tbs and Q22-bus interrupt vector tbs. Additional controllers are not supported.

3.5.1.2 Bootstrap Command Fl - When a bootstrap is invoked via BOOTSTRAP command, the user can specify several boot command flags by bit encoding the flags in a flag word specified with the "/R5:" qualifier. These command flags are described in table 4.

Table 4: VMB Bootstrap Command Flags

Bit Number	Value (Hex)	Flag Name	Description
0	00000001	CONVERSATION	Conversational bootstrap.

Table 4 (cont.)

Bit Number	Value (Hex)	Flag Name	Description
3	00000008	BOOTBLOCK	Secondary bootstrap from boot block. When this bit is set, VMB reads logical block number 0 of the boot device and tests it for conformance with the boot block format. If in conformance, the block is executed to continue the bootstrap. No attempt to perform Files-11 bootstrap is made.
4	00000010	DIAGNOSTIC	Diagnostic bootstrap. When this bit is set the secondary bootstrap is called [SYS0.SYSMAINT]DIAGBOOT.EXE
6	00000040	HEADER	Image header. If this bit is not set, VMB transfers control to the first location of the secondary bootstrap. If this bit is set, it transfers control to the address specified by the file's image header.
8	00000100	SOLICIT	File name solicit. When this bit is set, VMB prompts the operator for the name of the secondary bootstrap file.
9	00000200	HALT	Halt before transfer. When this bit is set, VMB halts before transferring control to the secondary bootstrap.
<31:28>	X0000000	TOPSYS	X can be any value from 0 through 7 (hex). This flag changes the top level directory name for system disks with multiple operating systems. For example, if X=1, the top level directory name is [SYS1...].

3.5.1.3 Booting From Disk -

For VMB to boot using a MSCP disk controller, the first controller must be configured at 772150(octal) and subsequent controllers must be properly configured in their appropriate floating CSR and vector slots

Table 4 (cont.)

Bit Number	Value (Hex)	Flag Name	Description
3	00000008	BOOTBLOCK	Secondary bootstrap from boot block. When this bit is set, VMB reads logical block number 0 of the boot device and tests it for conformance with the boot block format. If in conformance, the block is executed to continue the bootstrap. No attempt to perform Files-11 bootstrap is made.
4	00000010	DIAGNOSTIC	Diagnostic bootstrap. When this bit is set the secondary bootstrap is called [SYS0.SYSMAINT]DIAGBOOT.EXE
6	00000040	HEADER	Image header. If this bit is not set, VMB transfers control to the first location of the secondary bootstrap. If this bit is set, it transfers control to the address specified by the file's image header.
8	00000100	SOLICIT	File name solicit. When this bit is set, VMB prompts the operator for the name of the secondary bootstrap file.
9	00000200	HALT	Halt before transfer. When this bit is set, VMB halts before transferring control to the secondary bootstrap.
<31:28>	X0000000	TOPSYS	X can be any value from 0 through 7 (hex). This flag changes the top level directory name for systems with multiple operating systems. For example, if X=1, the top level directory name is [SYS1...].

3.5.1.3 Booting From Disk -

For VMB to boot using a MSCP disk controller, the first controller must be configured at 772150(octal) and subsequent controllers must be properly configured in their appropriate floating CSR and vector slots

+ 20: |

sum of previous three longwords

+-----

BB+0: These two bytes can have any value.

BB+2: This value is the word offset from the start

the bootblock to the identification are described below.

- BB+3: This byte must be one.
- BB+4: This longword contains the logical block number (word swapped) of the secondary image.
- BB+(2*n)+0: This byte defines the expected instruction set. 18(hex) means VAX.
- BB+(2*n)+1: This byte defines the expected controller type. 0 means unknown.
- BB+(2*n)+2: This byte defines the file structure on the volume, it may be any value.
- BB+(2*n)+3: This byte must be the ones complement of the sum of the previous three bytes.
- BB+(2*n)+4: This byte must be zero.
- BB+(2*n)+5: This byte must be 1 or 81 (hex). This byte defines the version number of the format standard and the type of disk. The version is 1, the high bit is 0 for single sided, 1 for double sided.
- BB+(2*n)+6: These two bytes must be zero.
- BB+(2*n)+8: This entry is a longword containing the size in blocks of the secondary bootstrap image.
- BB+(2*n)+12: This entry is a longword containing a load offset (usually zero) from the default load address of the secondary bootstrap.
- BB+(2*n)+16: This entry is a longword containing the byte offset into the secondary bootstrap when execution is to begin.
- BB+(2*n)+20: This entry is a longword containing the sum of the previous three longwords.

3.5.1.4 Booting From Tape -

If no bootable disk is found, VMB attempts to bootstrap from a TK50 tape. The TK50 must be configured at <tbs> octal to be recognized by VMB.

If a TK50 is present, VMB determines if a tape is loaded and the unit is online. If so, VMB rewinds the tape and searches for the file TAPEBOOT.EXE. (The user may specify an alternative file name by setting the SOLICIT bit in the software command register [see table 4]). If this file is found, VMB loads and executes it. Normally this file would contain a program to load an operating system from tape onto a system disk.

Note that a TAPEBOOT.EXE program to load a disk from tape is not a part of this project.

If a user has both disks and tape, and a disk is bootable, to boot from tape the user must either take all bootable disks offline or explicitly boot the TK50 via the console boot command.

3.5.1.5 Booting From PROM -

If neither disk nor tape is bootable, VMB checks for a PROM bootstrap.

To locate a PROM bootstrap, VMB searches the Q22-bus address range from low to high addresses in 16 page chunks looking for readable memory. If the first six longwords of any such page contains a valid "signature block" (identical to the second part of the bootblock format, see figure 5), VMB copies the PROM image to main memory and then transfers control to it.

Note that while defined as a MRV11 PROM or equivalent bootstrap, VMB doesn't actually require that the signature block or the bootstrap code be in PROM, it could be in ROM, nonvolatile RAM or, as described in section 3.5.1.7, it could be loaded into another KA630's RAM and mapped to the Q22-bus.

3.5.1.6 Booting From DEQNA -

If no other bootstrap device is found, VMB attempts to bootstrap from the DEQNA ethernet controller. In this case, the secondary bootstrap is downline loaded from a host on the Ethernet using DECNET Low-level Maintenance Protocol (MOP) Version 3.0. The DEQNA module must be configured at Q22-bus address 774440 (octal).

The downline load process consists of the following steps:

1. VMB performs local testing of the DEQNA. If the tests fail, the bootstrap attempt fails and the three LED's on the DEQNA are set according to the problem detected. The LED settings and their interpretations are:
 - o 3 LED's on: DEQNA initialization failure.

- o 2 LED's on: internal loopback failure.
 - o 1 LED on: external loopback failure.
2. VMB transmits a Program Request MOP message over the Ethernet. The message destination is the Load Assistant Multicast address AB-00-00-01-00-00. The message source address is the DEQNA station address (from DEQNA PROM). The MOP program type is the operating system.
 3. VMB waits approximately 30 seconds to receive a response. If it does not receive a response it retransmits the request every thirty seconds for a total of 2 minutes. If a response is not received in two minutes, the bootstrap fails.
 4. VMB accepts MOP Load Messages and loads the data into memory terminating when the final message is received as indicated in the MOP message protocol. If the interval between load messages exceeds 30 seconds, VMB restarts the DEQNA bootstrap at step 2.

3.5.1.7 Booting An Auxiliary Processor -

VMB bootstraps an auxiliary processor via the ROM bootstrap protocol. Recall from section 3.5 the Q22-bus initialization algorithm:

1. Turn on IPCR<8>, the halt flag.
2. Disable the I/O map by clearing IPCR<5>.
3. Initialize the I/O map.
4. Enable the I/O map by setting IPCR<5>.
5. If an auxiliary processor, loop until the IPCR<8> bit is cleared.

(Note that whenever the console program is entered, it turns off IPCR<8>.) Steps 1 and 5 ensure that an auxiliary will loop until some other processor clears IPCR<8>. When another processor, the bootstrap host, clears IPCR<8>, the auxiliary proceeds with the bootstrap. This synchronization gives the arbiter processor control over the bootstrapping of all auxiliary processors.

An auxiliary processor cannot directly bootstrap itself from any of the normal bootstrap devices so VMB on an auxiliary checks only for the ROM bootstrap described in section previously. The ROM bootstrap may be either a block of nonvolatile memory on the Q22-bus bus or the bootstrap host can construct an equivalent bootstrap in RAM. In either case the

auxiliary will not proceed with the bootstrap until the bootstrap host clears IPCR<8>. The bootstrap host in turn should not clear the auxiliary's IPCR<8> unless IPCR<5> is clear.

3.5.2 Secondary Bootstrap Program -

The secondary bootstrap program is invoked as the second part of system bootstrap. Following successful execution of the primary bootstrap, the secondary bootstrap has either been loaded into memory or located in ROM. It is the responsibility of the secondary bootstrap to complete the bootstrap of the processor.

VMB calls the secondary bootstrap with the processor in the following state:

- o The processor is running in kernel mode on the interrupt stack at IPL 31(hex).
- o R11 contains the base address of the extended RPB (figure 6) built by VMB.
- o AP contains the address of the secondary bootstrap argument list (figure 7).
- o SP contains the address of the top of the stack plus 4, which is also the address of the beginning of the secondary bootstrap (figure 8 for a map of memory).
- o SCBB (internal processor register) contains the address of the SCB built by VMB.

Note that the first four longwords of the VMB built extended RPB would not be recognized as a valid RPB by the console restart algorithm. It is up to the secondary bootstrap or the operating system itself to complete the RPB if automatic restart is desired.

Figure 6: Extended RPB

Offset (hex):	
00:	address of the extended RPB
04:	0
08:	0
0C:	0
10:	PC at restart/halt
14:	PSL at restart/halt
18:	halt code
1C:	VMB input register R0
20:	VMB input register R1
24:	VMB input register R2
28:	VMB input register R3
2C:	VMB input register R4
30:	VMB input register R5
34:	two longwords reserved
3C:	disk block address of secondary bootstrap
40:	size of secondary bootstrap file in blocks
44:	descriptor of PFN bitmap (two longwords)
48:	number of good physical pages
4C:	reserved
50:	physical CSR address of boot device
54:	four longwords reserved
68:	secondary bootstrap file name (40 characters)
90:	eight longwords reserved
B0:	system control block base address

Figure 7: Secondary Bootstrap Argument list

(AP)+00:	12
(AP)+04:	reserved
(AP)+08:	reserved
(AP)+12:	lowest valid PFN
(AP)+16:	highest valid PFN
(AP)+24:	PFN map size in bytes
(AP)+28:	address of PFN bitmap
(AP)+32:	reserved
(AP)+36:	reserved
(AP)+40:	processor ID (8????)
(AP)+44:	reserved
(AP)+48:	reserved

Figure 8: Secondary Bootstrap Memory Map

R11:	Extended RPB built by VMB	
+ 200(hex):	VMB	
+ ????(hex):	2 page SCB used by VMB	: SCBE
+ ????(hex):	8 page PFN bitmap	
+ ????(hex):	4 page stack for Secondary Bootstrap	
+ ????(hex):	Secondary Bootstrap :	: SP
+10000(hex):		

3.6 Console I/O Mode (System Halted)

When the KA630 is "halted", the operator controls the system through the console terminal using the console command language. The console terminal is in "Console I/O Mode". The console prompts the operator for input with the string ">>>".

3.6.1 Console Control Characters -

In console I/O mode, several characters have special meanings.

- o carriage return - ends a command line. No action is taken on command until after it is terminated by a carriage return. A null line terminated by a carriage return is treated as valid, null command. No action is taken, and the console re-prompts for input. Carriage return is echoed as carriage return, line feed.
- o rubout - when the operator types rubout, the console deletes the character that the operator previously typed. What appears on the console terminal depends on whether the terminal is video terminal or a hardcopy terminal.

For hard copy terminals, when a rubout is typed, the console echoes with a backslash (\), followed by the character being deleted. If the operator types additional rubouts, the additional characters deleted are echoed. When the operator types a non-rubout character, the console echoes another backslash, followed by the character typed. The result is to echo the characters deleted, surrounding them with backslashes. For example:

The operator types: EXAMI;E<rubout><rubout>NE<CR>

The console echoes: EXAMI;E\E;\NE<CR>

The console sees the command line: EXAMINE<CR>

For video terminals, when rubout is typed the previous character is erased from the screen and the cursor is restored to its previous position.

The console does not delete characters past the beginning of command line. If the operator types more rubouts than there are characters on the line, the extra rubouts are ignored. If a rubout is typed on a blank line, it is ignored.

- o control-U - the console echoes "^U<CR>", and deletes the entire line. If control-U is typed on an empty line, it is echoed and otherwise ignored. The console prompts for another

command.

- o control-S - stops output to the console terminal until control-Q is typed. Control-S and control-Q are not echoed. Control-C, control-O, and BREAK also clear control-S.
- o control-Q - resumes output to the console terminal. Additional control-Q's are ignored. Control-S and control-Q are not echoed.
- o control-O - causes the console to throw away transmissions to the console terminal until the next control-O is entered. Control-O is echoed as "^O"<CR> when it disables output, but is not echoed when it reenables output. Output is reenabled if the console prints an error message, or if it prompts for a command from the terminal. Displaying a REPEAT command does not reenable output. When output is reenabled for reading a command, the console prompt is displayed. Output is also enabled by entering program I/O mode, by BREAK and by control-C. Control-O clears control-S.
- o control-R - causes the console to echo <CR><LF> followed by the current command line. This function can be used to improve the readability of a command line that has been heavily edited.
- o control-C - causes the console to echo "^C" and to abort processing of a command. Control-C has no effect as part of a binary load data stream. Control-C clears control-S, and reenables output stopped by control-O. When control-C is typed as part of a command line, the console deletes the line as it does with control-U.
- o BREAK - If the console is in console I/O mode, BREAK is equivalent to control-C, but is echoed as "^P". If the console is in program I/O mode and halt is disabled, BREAK is ignored. If the console is in program I/O mode and halt is not disabled, BREAK causes the processor to halt and enter console I/O mode.

Control characters are typed by pressing the character key while simultaneously holding down the control key.

If an unrecognized control character is typed (a control character here means a character with an ASCII code less than 32 decimal [C0] or between 128 and 159 decimal [C1]) it is echoed as up arrow followed by the character with ASCII code 64 greater. For example, BEL (ASCII code 7) is echoed as "^G", since capital G is ASCII code 7+64=71. When a control character is deleted with rubout, it is echoed the same way. After echoing the control character, the console processes it like a normal character. Unless the control character is part of a comment, the command will be invalid, and the console will respond with an error message. Note that control codes from 128 to 159, the C1 control codes, cannot be entered by any present Digital terminal. The fact that both

the character with code 7 and the character with code 135 will both be considered as "^G" is not expected to have any practical consequences.

3.6.2 Console Command Syntax -

The console accepts commands of lengths up to 80 characters. Longer commands are responded to with an error message. The count does not include rubouts, rubbed out characters, or the terminating carriage return.

Commands may be abbreviated. Abbreviations are formed by dropping characters from the end of a keyword. All commands are recognized from their first character.

Multiple adjacent spaces and tabs are treated as a single space by the console. Leading and trailing spaces and tabs are ignored.

Command qualifiers can appear after the command keyword, or after an symbol or number in the command.

All numbers (addresses, data, counts) are in hexadecimal. (Note though, that symbolic register names include decimal digits.) Hex digits are 0 through 9, and A through F. The console does not distinguish between upper and lower case either in numbers or in commands. Both are accepted.

3.6.3 Console Command Keywords -

Processor control commands:

- o INITIALIZE
- o START <address>
- o CONTINUE
- o HALT
- o BOOT <device>
- o UNJAM

Data transfer commands:

- o EXAMINE <address>
- o DEPOSIT <address> <data>

- o X <address> <count>

Console control commands:

- o FIND
- o REPEAT <command>
- o TEST
- o ! <comment>

3.6.4 References To Processor Registers And Memory -

The KA630 console is implemented by macrocode executing from ROM. For this reason, the actual processor registers may not be modified by the command interpreter. When the console is entered, the console saves the processor registers in console memory and all command references to the registers are directed to the corresponding scratch page locations, not to the registers themselves. When the console reenters program mode, the saved registers are restored and any changes become operative only then. References to processor memory are handled normally except where noted below.

The console uses memory at the top of the available system memory for its own use (see figure 1). References to the console memory pages by EXAMINE and DEPOSIT commands must be qualified by the "/U" qualifier (Access is primarily to simplify debugging of the console program.) The binary load and unload command may not reference the console memory pages.

3.6.5 Console Commands -

3.6.5.1 BOOT -

BOOT [<qualifier list>] [<device>]

The device specification is of the format 'ddcu', where 'dd' is a two letter device mnemonic, 'c' is an optional one digit controller number and 'u' is a one digit unit number.

The console initializes the processor and starts VMB running. (See the section on System Bootstrapping.) VMB boots the operating system from the specified device. The default bootstrap device is determined as described in the section on system bootstrapping.

Qualifiers:

- o /R5:<data> - After initializing the processor and before starting VMB, R5 is loaded with the specified data. This allows a console user to pass a parameter to VMB. (To remain compatible with previous processors, /<data> will also be recognized to have the same result.)

3.6.5.2 CONTINUE -

CONTINUE

The processor begins instruction execution at the address currently contained in the program counter. Processor initialization is not performed. The console enters program I/O mode.

If upon entry the console program is unable to preserve the machine state, either because it was a power-up or because it was unable to locate a scratch page on the interrupt stack, the CONTINUE command will be rejected.

3.6.5.3 DEPOSIT -

DEPOSIT [<qualifier list>] <address> <data>

Deposits the data into the address specified. If no address space or data size qualifiers are specified, the defaults are the last address space and data size used in a DEPOSIT or EXAMINE command. After processor initialization, the default address space is physical memory, the default data size is long, and the default address is zero.

If the specified data is too large to fit in the data size to be deposited, the console ignores the command and issues an error response. If the specified data is smaller than the data size to be deposited, it is extended on the left with zeros.

The address may also be one of the following symbolic addresses:

- o PSL - the processor status longword. No address space qualifier is legal. When PSL is examined, the address space is identified as "M" (machine dependent).
- o PC - the program counter (general register 15). The address space is set to /G.
- o SP - the stack pointer (general register 14). The address space is /G.

- o Rn - general register 'n'. The register number is in decimal. The address space is /G. For example:

D R5 1234 is equivalent to D/G 5 1234

D R10 6FF00 is equivalent to D/G A 6FF00
- o '+' - the location immediately following the last location referenced in an examine or deposit. For references to physical or virtual memory spaces, the location referenced is the last address, plus the size of the last reference (1 for byte, 2 for word, 4 for long). For other address spaces, the address is the last address referenced, plus one.
- o '-' - the location immediately preceding the last location referenced in an examine or deposit. For references to physical or virtual memory spaces, the location referenced is the last address minus the size of this reference (1 for byte, 2 for word, 4 for long). For other address spaces, the address is the last addressed referenced minus one.
- o '*' - the location last referenced in an examine or deposit.
- o '@' - the location addressed by the last location referenced in an examine or deposit.

Qualifiers:

- o /B - The data size is byte.
- o /W - The data size is word.
- o /L - The data size is longword.
- o /V - The address space is virtual memory. All access and protection checking occur. If the access would not be allowed to a program running with the current PSL, the console issues an error message. Virtual space DEPOSITS cause the PTE<M> bit to be set. If memory mapping is not enabled, virtual addresses are equal to physical addresses.
- o /P - The address space is physical memory.
- o /I - The address space is internal processor registers. These are the registers addressed by the MTPR and MFPR instructions.
- o /G - The address space is the general register set, R0 through PC.
- o /U - Access to the console scratch page is allowed. This qualifier also disables virtual address protection checks.

- o /N:<count> - The address is the first of a range. The console deposits to the first address, then to the specified number of succeeding addresses. Even if the address is the symbolic address "-", the succeeding addresses are at larger addresses. The symbolic address specifies only the starting address, not the direction of succession. For repeated references to preceding addresses, use "REPEAT DEPOSIT - <data>".

For example:

D/P/B/N:1FF 0 0	Clears the first 512 bytes of physical memory.
D/V/L/N:3 1234 5	Deposits 5 into 4 longwords starting at virtual address 1234.
D/N:8 R0 FFFFFFFF	Loads general registers R0 through R7 with -1.
D/N:200 - 0	Starting at previous address, clear 512 bytes.

If conflicting address space or data sizes are specified, the console ignores the command and issues an error response.

3.6.5.4 EXAMINE -

EXAMINE [<qualifier list>] [<address>]

Examines the contents of the specified address. If no address is specified, "+" is assumed. The address may also be one of the symbolic addresses described under DEPOSIT.

Qualifiers:

The same qualifiers may be used on EXAMINE as may be used on DEPOSIT.

RESPONSE: <tab><address space identifier> <address> <tab> <data>

The address space identifier can be:

- o P - physical memory. Note that when virtual memory is examined, the address space and address in the response are the translated physical address.
- o G - general register.
- o I - internal processor register.
- o M - machine dependent (used only for display of the PSL).

3.6.5.5 FIND -

FIND [<qualifier list>]

The console searches main memory starting at address zero for a page-aligned 64 kilobyte segment of good memory, or a restart parameter block (RPB). If the segment or block is found, its address plus 512 is left in SP. If the segment or block is not found, an error message is issued, and the contents of SP are UNPREDICTABLE. If no qualifier is specified, /RPB is assumed.

Qualifiers:

- o /MEMORY - search memory for a page aligned segment of good memory, 64 kilobytes in length. The search includes a read/write test of memory and leaves the contents of memory UNPREDICTABLE.
- o /RPB - search memory for a restart parameter block. See the section 3.4 for the search algorithm. The search leaves the contents of memory unchanged.

3.6.5.6 INITIALIZE -

INITIALIZE

A processor initialization is performed. The following registers are set (all values are hexadecimal):

PSL	041F0000
IPL	1F
ASTLVL	4
SISR	0
ICCS	0
RXCS	0
TXCS	80
MAPEN	0

All other registers are UNPREDICTABLE.

The previous console reference defaults (the defaults used to fill in unsupplied qualifiers for DEPOSIT and EXAMINE commands) are set to physical address, longword size and address 0.

3.6.5.7 HALT -

HALT

The HALT command has no affect, the processor is already halted when in console I/O mode.

RESPONSE: Already halted

3.6.5.8 REPEAT -

REPEAT <command>

The console repeatedly displays and executes the specified command. The repeating is stopped by the operator typing control-C. Any valid console command may be specified for the command with the exception of the REPEAT command.

RESPONSE: <dependent upon command specified>

3.6.5.9 START -

START [<address>]

The console starts instruction execution at the specified address. If no address is given, the current PC is used. If no qualifier is present, macroinstruction execution is started. If memory mapping is enabled, macroinstructions are executed from virtual memory. The START command is equivalent to a DEPOSIT to PC, followed by a CONTINUE. No INITIALIZE is performed.

3.6.5.10 TEST -

TEST [<test number>]

The console invokes a diagnostic test program denoted by <test number>. Valid test numbers are 3 through 7 and hexadecimal B.

See "KA630 ROM Diagnostic Specification" for information on diagnostic tests and their corresponding test number codes.

3.6.5.11 UNJAM -

An I/O bus reset is performed.

3.6.5.12 Binary Load And Unload Command -

X <address> <count> <CR> <checksum>

The X command is for use by automatic systems communicating with the console. It is not intended for use by operators. The console loads or unloads (that is, writes to memory, or reads from memory) the specified number of data bytes, starting at the specified address.

If bit 31 of the count is clear, data is to be received by the console and deposited into memory. If bit 31 of the count is set, data is to be read from memory and sent by the console. The remaining bits in the count are a positive number indicating the number of bytes to load or unload.

The console accepts the command upon receiving the carriage return. The next byte the console receives is the command checksum, which is not echoed. The command checksum is verified by adding all command characters, including the checksum, (but not including the terminating carriage return or rubouts or characters deleted by rubout), into an 8 bit register initially set to zero. If no errors occur, the result is zero. If the command checksum is correct, the console responds with the input prompt and either sends data to the requester or prepares to receive data. If the command checksum is in error, the console responds with an error message. The intent is to prevent inadvertent operator entry into a mode where the console is accepting characters from the keyboard as data, with no escape sequence possible.

If the command is a load (bit 31 of the count is clear), the console responds with the input prompt, then accepts the specified number of bytes of data for depositing to memory, and an additional byte of received data checksum. The data is verified by adding all data characters and the checksum character into an 8 bit register initially set to zero. If the final contents of the register is non-zero, the data or checksum are in error, and the console responds with an error message.

If the command is a binary unload (bit 31 of the count is set), the console responds with the input prompt, followed by the specified number of bytes of binary data. As each byte is sent it is added to a checksum register initially set to zero. At the end of the transmission, the 2's complement of the low byte of the register is sent.

If the data checksum is incorrect on a load, or if memory errors or line errors occur during the transmission of data, the entire transmission is completed, and then the console issues an error message. If an error occurs during loading, the contents of the memory being loaded are UNPREDICTABLE.

Echo is suppressed during the receiving of the data string and checksums.

It is possible to control the console through the use of the console control characters (control-C, control-S, control-O, etc.) during binary unload. It is not possible during a binary load, as all received characters are valid binary data.

Data being loaded with a binary load command must be received by the console at a rate of at least one byte per second. The command checksum that precedes the data must be received by the console within 10 seconds of the <CR> that terminates the command line. The data checksum must be received within 10 seconds of the last data byte. If any of these timing requirements are not met the console aborts the transmission by issuing an error message and prompting for input.

The entire command, including the checksum, may be sent to the console as a single burst of characters at the console's specified character rate. The console is able to receive at least 4K bytes of data in a single 'X' command.

3.6.5.13 Comment -

! <comment>

The comment command is ignored. It is used to annotate console I/O command sequences.

3.6.6 Console Errors And Error Messages -

The console can issue error messages in response to commands. These error messages are listed in table 5.

Table 5: Console Error Messages

Code	Message	Description
?15	CORRPTN	The console program database has been corrupted, the console program does a powerup initialization to rebuild its database.
?16	ILL REF	The requested reference would violate virtual memory protection, the address is not mapped, the reference is invalid in the specified address space, or the value is invalid in the specified destination.
?17	ILL CMD	The command string cannot be parsed.
?18	INV DGT	A number has an invalid digit.

Table 5 (cont.)

Code	Message	Description
?19	LTL	The command was too large for the console to buffer. The message is issued only after receipt of the terminating carriage return.
?1A	ILL ADR	The address specified falls outside the limits of the address space.
?1B	VAL TOO LRG	The value specified does not fit in the destination.
?1C	SW CONF	For example, two different data sizes are specified with an EXAMINE command.
?1D	UNK SW	The switch is unrecognized.
?1E	UNK SYM	The symbolic address in an EXAMINE or DEPOSIT is unrecognized.
?1F	CHKSM	The command or data checksum of an X command is incorrect. If the data checksum is incorrect, this message is issued, and is not abbreviated to "Illegal command".
?20	HLTED	The operator entered a HALT command.
?21	FND ERR	A FIND command failed either to find the RPB or 64 kb of good memory.
?22	TMOUT	During an X command, data failed to arrive in the time expected.
?23	MEM ERR	Parity error detected.

Some console commands may result in errors. For example, if a memory error occurs as the result of a console command, the console will respond with an error message.

3.6.7 Halts And Halt Messages -

Whenever the processor halts, the console prints the response "PC <PC>". For example:

```
?06 HLT INST
    PC = 800050D3
```

The number preceding the halt message is the halt code, and is passed to the operating system on a restart. The halt messages are shown in table 6.

Table 6: KA630 Halt Messages

Code	Message	Description
?02	EXT HLT	BREAK was typed on the console, QBINIT or QBHALT was asserted.
?04	ISP ERR	In attempting to push state onto the interrupt stack during an interrupt or exception, the processor discovered that the interrupt stack was mapped NO ACCESS or NOT VALID.
?05	DBL ERR	The processor attempted to report a machine check to the operating system, and a second machine check occurred.
?06	HLT INST	The processor executed a HALT instruction in kernel mode.
?07	SCB ERR3	The vector had bits <1:0> equal to 3.
?08	SCB ERR2	The vector had bits <1:0> equal to 2.
?0A	CHM FR ISTK	A change mode instruction was executed when PSL<IS> was set.
?0B	CHM TO ISTK	The exception vector for a change mode had bit <0> set.
?0C	SCB RD ERR	A hard memory error occurred while the processor was trying to read an exception or interrupt vector.
?10	MCHK AV	An access violation or an invalid translation occurred during machine check exception processing.
?11	KSP AV	An access violation or an invalid translation occurred during processing of of an invalid kernel stack pointer exception.

3.7 Program I/O Mode (System Running)

When the processor is not executing instructions from the console program ROM, it is in "program I/O mode" in which all terminal

interaction is handled by the operating system. In program I/O mode the console terminal behaves like any other operating system terminal. If halts are disabled, BREAK is ignored. If halts are enabled, BREAK causes the processor to "halt", that is the KA630 enters console I/O mode.

4 PUBLICATIONS

tbd

5 PACKAGING

NOT APPLICABLE - This product resides in ROM and is shipped as part of the processor board.

6 INSTALLABILITY

NOT APPLICABLE - This product resides in ROM and is shipped as part of the processor board.

7 EASE OF USE

The users of the KA630 console program include many who are sophisticated in their use and understanding of computers. These users are Engineering, Manufacturing and Field Service personnel as well as many customers. These users are not intimidated by the console command language and its messages nor by the messages generated by the console diagnostics. The relative complexity of these components of the console system are perceived as worth the diagnostic power that they provide. It is for these users that these features are provided. Indeed these features have evolved over time to satisfy these users and therefore for this user group, the KA630 console program is regarded as "easy to use".

Another class of users however are those users who know little about computers and who may be intimidated by them. These users would like to simply flip a switch and know that the computer will turn on and they can proceed with their work. These users are not expected to use the console command language, nor are they expected to understand and interpret diagnostic messages.

If a KA630 is to be used by a user from the latter group, halts must be disabled, the appropriate bootstrap device configured, and the proper baud rate must be set. Establishing this configuration for the unsophisticated user may be done in manufacturing (the preferred

approach), by Field Service personnel, by a sophisticated user, or by a unsophisticated user following a well designed and documented installation procedure.

Once properly configured, assuming no hardware problems are detected during the diagnostics on power up, figure 9 shows what the user will see displayed.

Figure 9: Console Display on Successful Power-Up

```
KA630.xx  
Performing normal system tests.  
  7..6..5..4..3..  
Tests completed.  
Loading system software.  
  2..1..0  
<Operating system specific output>
```

The first line identifies the processor and the version number (xx) of the console program ROM. The next line explains that the system is performing NORMAL tests. The count-down sequence reassures the user that the system is progressing through its tests, that the system hasn't "gone away", and documents which tests are executed. When diagnostic are completed, the console notifies the user that the diagnostic successfully completed. The "Loading system software." message indicates the beginning of the bootstrap sequence. The execution of the bootstrap sequence causes the remaining digits of the countdown to be displayed. Because successful completion of a bootstrap occurs in the context of the operating system bootstrapped, a confirming message indicating that the system startup has completed can only be issued by the bootstrapped operating system.

Figure 10 shows the display generated if any hard errors are detected during any test (test 7 for example).

Figure 10: Console Display on Power-Up with Hard Errors

```
KA630.xx  
  
Performing normal system tests.  
  
7..  
  
?<subtest> <p1> <p2> <p3>  
  
Failure.  
Normal operation not possible.
```

In the case of fatal problems detected by the diagnostics, the countdown sequence is interrupted and a diagnostic message is displayed. The diagnostic message is composed of a question mark, a subtest code number and up to three parameters for use by diagnostic personnel. More than one such error message is possible but unlikely. The summary message which follows indicates that the test failed and that normal operation is not possible. The console program then hangs.

8 PERFORMANCE

All diagnostic checks will be performed in less than tbd. Diagnostic coverage of the processor itself will be tbd percent, coverage of the memory system will be tbd percent, and coverage of the Q22-bus map will be tbd.

The console responds to all commands within 1 second.

When the console power-up display is displayed, no more than 5 seconds shall pass without some output to the screen (additional periods may be displayed during the countdown if necessary).

9 RELIABILITY

Errors detected by this program are classified as user errors, hard errors, or catastrophic errors. User errors are errors arising from invalid input from the user. All user errors are diagnosed and the user is notified by a diagnostic message on the console terminal. All such errors are recoverable.

Hard errors are errors detected by the program which, while unrecoverable, do not prevent the continuation of the program. An example of a hard error is the detection of an unexpected non-existent memory error when in console I/O mode. The console program notifies the user of all hard errors with a diagnostic message on the console.

Catastrophic errors are errors of such a severe magnitude that the program cannot continue. When a catastrophic error is detected, the program attempts to display an error message on the console terminal. Following that, the processor goes into an infinite loop at IPL 31 (there is no halt state for MicroVAX). An example of a catastrophic error is when the console program is unable to locate any working memory.

It is possible to bypass all diagnostic tasks. This may be done by enabling halts and by manually halting the processor following power-up or reset. The diagnostics will be halted and the processor will enter console I/O mode. This option allows a field service engineer to bypass a failing test and enter console I/O mode where the console commands can be used to further diagnose the problem.

As part of each diagnostic subtest, a test code is displayed on the console and on the LEDs, making it possible to monitor the progress of the diagnostics. The two display mechanisms use unrelated logic providing a high probability that at least one will be operative. If a hard error is detected by a test, a diagnostic message is displayed on the console. If a catastrophic error occurs, it may not be possible to display a diagnostic message on the console but the most recent test code will be left in the LEDs.

10 MAINTAINABILITY

The console program is located in two socketed ROM's on the KA63 processor. If necessary, they can be replaced. This is the only form of maintenance possible.

The console program displays the version number of the console program ROM when the power up messages are displayed. This can be used to easily determine which version of the console program is present.

Upon completion of development, the source copy of the console program will be preserved for ECO purposes by the VMS group.

11 MAINTENANCE

The console program is located in two socketed ROM's on the KA63 processor. If necessary, they can be replaced. This is the only form of maintenance possible.

12 COMPATIBILITY

12.1 Product Compatibility

This program is a new product developed for the first time for the KA630. Even though other MicroVAX based processors might be able to utilize parts of this product, no effort is planned to facilitate such sharing beyond that which arises naturally from sound modular design.

12.2 Standards Conformance

Although not required, the command language accepted by the console is intended to be as compatible as possible with that defined in DEC Standard 32. However, the following significant console features are defined in the standard but omitted by the KA630 console program:

- o MICROSTEP command - not supported by the MicroVAX chip.
- o LOAD command - no console storage device is supported.
- o SET command - no set options are defined.
- o NEXT command - not supported by the MicroVAX chip.
- o @ command - no console storage device is supported.

The console program recognizes terminal device attributes based on device attribute responses registered in DEC Standard 138.

Escape sequence parsing (used to identify terminals and to cause all escape sequences to act as line terminators) is based on the algorithm defined in the "Video System Standards Reference Manual".

12.3 Internationalization

All console program message texts are either multilingual or language-independent. The message texts displayed on power-up (shown in figures 9 and 10) are multilingual. All other texts are language independent; they incorporate short language insensitive abbreviations rather than readable sentences. Depending on the user preference, the message texts are output in either English, French, German, Italian, Portuguese, Spanish, Dutch, Danish, Finnish, Norwegian or Swedish.

Numeric status displays on the processor LEDs facilitate the diagnosis of failing processors in a language independent manner.

The operation of the console is independent of the user's line voltage or frequency.

The console supports the Digital Multinational Character Set (MCS). This support extends to displaying foreign language messages with MCS accepting and echoing MCS characters and accepting a device attribute report (the console queries the terminal to determine if it is a CRT or not) using the C1 control characters of MCS. However, all console commands must be entered using the ANSI subset.

If the terminal does not support MCS, the console uses English message texts.

The console program uses four characters that are national replacement characters, the caret (^), the backslash (\) and the right and left square brackets ([,]). The caret is used by the console to denote control characters. The backslash is used to delimit text deletion when editing console input. The square brackets are used to denote directory specifications when the user directs the bootstrap to solicit a secondary bootstrap file name. No provision is made for terminal characters which replace any of these characters.

13 EVOLVABILITY

Because the console program is in ROM and cannot be easily changed there is little likelihood of significant evolution over its lifetime. An exception is support for additional bootstrap devices.

Support for new types of disk and tape units is provided not by the console program itself but by the console's support for MSCP and TMSC disks and tapes. Any new disk or tape supported by the MSCP and TMSC controllers will be supported by the console program.

Another possible need is support for a new Ethernet controller. Any new controller compatible with the old will be supported. Any new and incompatible controller could not be supported except by an ECO of the console ROM.

The console terminal recognition code relies on compliance with DE video terminal architecture standards to insure proper recognition of future terminals.

14 COSTS

NOT APPLICABLE

15 TIMELINESS

This product must be available for the first release of the KA630 processor which is gated by availability of this product.

In order to insure that this goal is met, every effort will be made to minimize the interdependence of the KA630 development project and others. For this reason, the KA630 development group will maintain complete responsibility for the entry/dispatch code and the diagnostic code. These sections are very processor implementation specific and will be done outside the group, schedule risk would be raised.

The command language interpreter is less implementation specific because it will be needed early in the development phase of the KA630, it will be done within the development group as well.

The restart code will be done by the KA630 development group as well as the bootstrap code will be written by the VMS development group. The VMS supplied code will be available on schedule is insured by the fact that the code is essentially the same as that in SEAHORSE, which will be completed previously.

16 CONSTRAINTS AND TRADEOFFS

The following list reflects the priority ordering of the goals for the product (1 is highest):

1. Time to market - must be available for the KA630 first release
2. Reliability - cost of field repairs is high, therefore, errors should ship.
3. Bootstraps supported - must support RQDX1 disks and DEQ (DECNET).
4. Diagnostic coverage - within available ROM space, diagnostic coverage should be maximized.
5. Memory consumption - ROM is available in 16, 32 and 64 K amounts. The console program should attempt to meet its goal within 32 Kb.
6. Support of QxSS based bitmapped terminals.
7. Console I/O language complexity - reduction of command verbosity and editing capabilities will be considered to meet space and schedule goals.

If necessary, to meet the highest priority goals lower priority goals will be compromised.

APPENDIX A

INTERPRETATION OF KA630 LED AND CONSOLE DISPLAY CODES

There are four red LEDs on the KA630 (echoed off the processor board as well). These four LEDs, interpreted as a four bit hexadecimal digit have the meanings described in table A-1. All LED codes are displayed during power-up. A problem is indicated only if the LED code is continuously displayed.

Each LED is illuminated if its control bit in the boot and diagnostic register (BDR) is one and is dark if its control bit is zero.

It is possible for privileged code to output to the LEDs when the processor is in program mode. Because this may cause confusion in the interpretation of the LEDs, this use of the LEDs in program mode is discouraged.

When executing the power-up sequence, the codes '9' through '0' are also output to the console terminal. See figures 9 and 10.

Table A-1: KA630 LED Interpretation

Code	Activity	Exit Criteria
F	Electrical power-up.	MicroVAX starts execution from console program ROM.
E	Wait for PWR-OK.	BDR<15> set.
D	Perform ROM checksum and TOY RAM tests. (1)	Test success.
C	Initialize console program memory.	Console memory and bitmap initialized, registers saved.
B	Run IPCR tests. (1)	Test success.
A	Test for and check out QxSS video console display if present. (1)	QxSS operational or not present.

INTERPRETATION OF KA630 LED AND CONSOLE DISPLAY CODES

Table A-1 (cont.)

Code	Activity	Exit Criteria
9	Perform console port tests and terminal identification. (1)	Console terminal type determined.
8	Query console language (1), then enter console command mode.	Exits automatically on power-up, otherwise exits on console CONTINUE, START, BOOT or TEST command.
7	Run memory pattern tests. (2)	At least 64 kb of contiguous good memory found.
6	Run memory address tests. (1)	All address tests passed.
5	Run I/O map tests. (1)	Test success.
4	Run CPU tests. (1)	Test success.
3	Run Interrupt tests. (1)	Test success.
2	Search for bootstrap device. (3)	Valid bootstrap device located.
1	Load bootstrap. (3)	Bootstrap successfully loaded.
0	Program mode.	Not applicable.

(1) Performed only on power-up entry into console program.

(2) Performed on power-up entry and on operator requested bootstrap.

(3) Performed only during bootstrap.

APPENDIX B

SUMMARY OF I/O REGISTERS USED BY CONSOLE PROGRAM

B.1 BOOT AND DIAGNOSTIC CONFIGURATION REGISTER (BDR)

The BDR register is used to read the values of the KA630 configuration switches and to output to the processor LED display.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PWR		HLT		CPU			BDG			DSPL					
OK		ENB													

Boot and Diagnostic Register

Hex addr: 20080000 (hex)

- BDR<15> PWR OK Power OK - set when power is stable.
- BDR<14> HLT ENB Halt enable - A read only bit that enable recognition of the BHALT(L) bus signal as a halt condition and enables recognition of a BREA condition from the console terminal as a halt condition. If not set, neither condition will result in a processor halt. This bit is set by a switch external to the KA630.

If CPMBX<1:0> is zero, this bit also control the actions of the console program in response to any halt condition. If set the console program will enter console I/O mode following any halt. If not set the console program will attempt to restart the processor and if that fails, it will attempt to reboot the processor. If that fails as well, the console program will then enter console I/O mode. If CPMBX<1:0> is non-zero, the console program uses that field to determine its actions on processor halt.

SUMMARY OF I/O REGISTERS USED BY CONSOLE PROGRAM
 BOOT AND DIAGNOSTIC CONFIGURATION REGISTER (BDR)

BDR<12:11> CPU CPU designation

0	Arbiter CPU
1	Auxiliary CPU number 1
2	Auxiliary CPU number 2
3	Auxiliary CPU number 3

BDR<10:9> MODE Power-up mode control bits. These two bits are combined with the HLT ENB to generate a three bit power-up mode field. The interpretation of this three bit field is given in table 1.

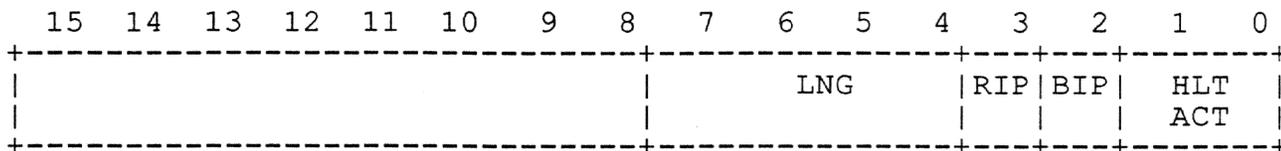
BDR<3:0> DSPL LED display bits - these write-only bits are used to load the LED status display. A bit value of one causes the associated LED to be illuminated, a bit value of zero causes the LED to be dark.

B.2 TIME OF YEAR (TOY) CLOCK REGISTERS

The TOY registers contain the time of year clock registers as well as 5 bytes of battery backed-up RAM. The console program references the clock registers only to check for battery backup failure. See the KA630-A Processor Specification for information on the TOY clock registers. The console program reserves all of the RAM registers. There are a total of 64 eight bit registers, word aligned. They may be accessed either as bytes or as words. When accessed as words, bits <15:8> are ignored on writes and return zeros on reads. The base address of the TOY registers is 200B8000 (hex).

B.2.1 Console Program Mailbox (CPMBX)

The console program and the operating system communicate with each other via this register.



Console Program Mailbox Register (CPMBX)

TOY register 14, Hex addr: 200B801C

CPMBX<7:4> LNG Console Message Text Language. This field

SUMMARY OF I/O REGISTERS USED BY CONSOLE PROGRAM
 TIME OF YEAR (TOY) CLOCK REGISTER

controls the output of message texts to the console terminal. If the field is set to zero the console program will prompt the user to set this field on power up.

- 0 Prompt for language.
- 1 German 6 Danish 11 Portuguese
- 2 English 7 Dutch
- 3 Spanish 8 Finnish
- 4 French 9 Norwegian
- 5 Italian 10 Swedish

CPMBX<3> RIP If set, a restart attempt is in progress. This flag must be cleared by the operating system if the restart succeeds.

CPMBX<2> BIP If set, a bootstrap attempt is in progress. This flag must be cleared by the operating system if the bootstrap succeeds.

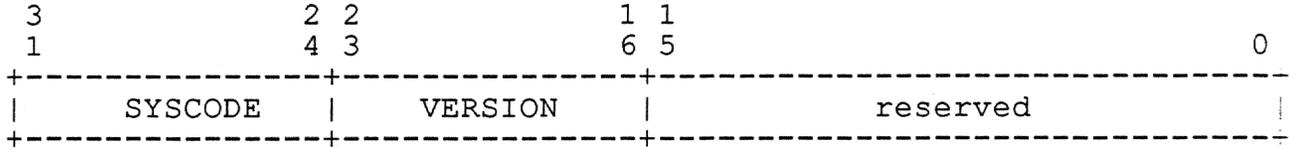
CPMBX<1:0> HLT ACT Processor halt action - this field is used to control the automatic restart/bootstrap procedure. This mailbox allows operating system software to override the BDR HLT ENB field. Both bits are cleared on power up and when the console program exits.

- 0 Use HLT ENB (BDR<14>) to determine action.
- 1 Restart, if that fails, halt.
- 2 Reboot, if that fails, halt.
- 3 Halt.

B.3 SYSTEM IDENTIFICATION EXTENSION REGISTER (SIDEX)

Any MicroVAX based system must implement a longword at physical location 20040004 (hex) to distinguish it from other MicroVAX based systems. The MicroVAX implements the system identification IPR but this register only identifies the processor implementation, not the system itself. This longword exists in the physical address range of the KA630-A console program ROM.

SUMMARY OF I/O REGISTERS USED BY CONSOLE PROGRAM
SYSTEM IDENTIFICATION EXTENSION REGISTER (SIDEX)



System Identification Extension (SIDEX)

- SIDEX<31:24> SYSCODE System code. This field is 1 for the KA630-A.
- SIDEX<23:16> VERSION Version number of console program ROM.
- SIDEX<15:0> Reserved.

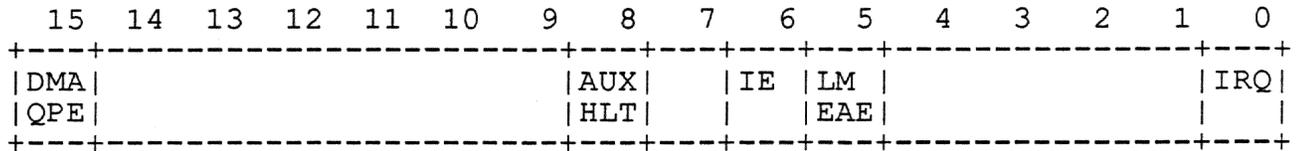
B.4 CONSOLE TERMINAL REGISTERS

The console program accesses the console terminal through four internal processor registers, the Console Receive Control and Status Register (RXCS), the Console Receive Data Buffer Register (RXDB), the Console Transmit Control and Status Register (TXCS), and the Console Transmit Data Buffer Register (TXDB). See the KA630-A Processor Specifications for their definition.

B.5 INTERPROCESSOR COMMUNICATION REGISTER (IPCR)

The IPCR register is used to support communication between an arbiter and auxiliary processors. It controls the visibility of auxiliary memory on the Q-22 bus, it allows an auxiliary processor to be halted by another processor, and it provides for an interprocessor interrupt capability. The location of the IPCR in the Q-22 bus address range depends on the CPU identification code of the processor.

SUMMARY OF I/O REGISTERS USED BY CONSOLE PROGRAM
 INTERPROCESSOR COMMUNICATION REGISTER (IPCR)



Interprocessor Communication Register

Hex address:	20001F40	Arbiter
	20001F42	Auxiliary #1
	20001F44	Auxiliary #2
	20001F46	Auxiliary #3

- ICR<15> DMA QPE DMA Q-22 address space parity error. This read only bit is set whenever a parity error is detected when an external CPU or device references KA630 local memory. (Not used by the console program).
- ICR<8> AUX HLT Auxiliary halt. A read/write bit which when set on an auxiliary processor, it causes the processor to halt. This bit has no effect if the auxiliary is in console I/O mode and is ignored all together by arbiter processors. This bit is always cleared upon entry into console I/O mode. It is also used by auxiliary processors as part of its bootstrap process (see section 3.5.1.7).
- ICR<6> IE Interrupt enable. If set, an interrupt will be generated if IPCR<0> is set. To the local processor this is a read/write bit. To external processors this is a read only bit.
- ICR<5> LM EAE Local memory external access enable. When this bit is set, local memory can be accessed via the Q-22 bus map. To the local processor this is a read/write bit. To external processors this is a read only bit.
- ICR<0> IRQ Interrupt request. When set, a processor interrupt is requested. This bit is enabled by ICR<6>. If ICR<6> is not set, writes to this bit are ignored and the bit is not set.

APPENDIX C

QXSS AND APT

When a KA630 system powers up for the first time it determines what console device is present. If the console device is a QxSS video display, it uses it as the console. However, during manufacturing systems are tested by APT which relies on the console program using the console port rather than the QxSS.

To continue to support APT testing, the KA630 console program allows APT to force the console program to revert to using the console port instead of the QxSS. To do this, APT must wait until the system has completed its power-up. APT then asserts break and the system halts. When the console program halts, it checks for the existence of a break condition on the console port. If break is being asserted, the console program resets itself to assume that the console port is connected to a hardcopy terminal and the console language is English. APT can now use the system normally.

Once this is done, there is no way short of powering the system off and turning it back on again to cause the console program to resume use of the QxSS as the console display device.

APPENDIX D
OUTSTANDING ISSUES

1. Section 1, page 1 - QVSS support needs to be further defined within this specification. The intent to support is defined but the exact definition of that support is TBS. This is important as QVSS is considerably more complicated to support than a console terminal.
2. Section 3.5.1.1, page 16 - More details on MAYA and QVSS descriptions are needed, planned controller names, unit designations, etc.
3. Section 4, page 38 - What should be in the publication section?
4. Section 8, page 40 - Need performance indications from diagnostics.
5. Section 10, page 41 - Need agreement from VMS that they will take the console program sources as part of VMS source kit.

[End of file]

