DECchip 21064 PALcode
System Design Guide

August 1993

This guide explains how to use the Privileged Architecture Library code (PALcode) to customize DECchip 21064 components to meet a variety of hardware and software application needs.

Order Number:	EC-N0543-72
Revision/Update Information:	This is a new document.
Operating System Version:	OSF/1 Version BL12 or higher
Software Version:	PALcode Version 1.2

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, August 1993

While Digital believes the information included in this publication is correct as of the date of publication, it is subject to change without notice.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Digital or an authorized sublicensor.

© Digital Equipment Corporation 1993.

All Rights Reserved.
Printed in U.S.A.

The following are trademarks of Digital Equipment Corporation: Alpha AXP, AXP, the AXP logo, DEC, DECchip, Digital, DECladebug, DECstation, the DIGITAL logo, OpenVMS, and VAX DOCUMENT.

OSF and OSF/1 are registered trademarks of the Open Software Foundation, Inc.
PostScript is a registered trademark of Adobe Systems Inc.

All other trademarks and registered trademarks are the property of their respective holders.

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

Preface ix

1 PALcode Fundamentals

1.1 PALcode Description 1–2
1.1.1 The Role of PALcode in the Alpha AXP Architecture 1–3
1.1.2 PALcode Functions 1–4
1.2 The PALcode Environment: PALmode 1–6
1.3 Special Functions Required for PALcode 1–6

2 DECchip 21064 PALcode Concepts

2.1 Invoking PALcode 2–2
2.1.1 PALcode Entry Points 2–4
2.1.2 Privileged and Unprivileged CALL_PAL Functions 2–6
2.1.3 DECchip 21064 Implementation of Opcodes Reserved to PALcode 2–6
2.2 DECchip 21064 Instruction Issue Rules 2–8
2.3 DECchip 21064 PALmode Restrictions 2–8
2.4 Internal Processor Register Access 2–10
2.5 Determining CALL_PAL Entry Points 2–12
2.5.1 Physical Addressing of CALL_PAL Functions 2–14

3 PALcode Product Design and Development Concepts

3.1 What is the PALcode Product? 3–2
3.2 User Categories and Support Considerations 3–2
3.2.1 User Categories and Descriptions 3–3
3.3 PALcode Memory Map 3–5
3.4 PALcode Parameter Block 3–7
3.5 PALcode Development Concepts 3–10
3.6 Using the PALcode Violation Checker 3–11
3.7 Files for Each User Category 3–12

3.7.1 Comparison of User Categories 3–12
3.7.2 File Creation Process for Composite Users 3–13
3.7.3 File Creation Process for Custom Users 3–14
3.7.4 Composite User Product Files 3–15
3.7.5 Custom User Product Files 3–17
3.8 Complete PALcode Files Listing 3–19
3.9 Sample PALcode Implementation 3–25

4 PALcode Design Process

4.1 Customization Decisions 4–2
4.1.1 Design Decision Examples 4–3
4.2 PALcode Modification Process 4–5
4.2.1 PVC Label Format 4–5
4.2.2 PALcode Assembly Rules 4–6
4.2.3 Modifying PALcode 4–7
4.3 Addressing New CALL_PAL Functions 4–10

5 How the Example Design Fits into the EB64 System

5.1 EB64 System 5–2
5.2 How the Example PALcode Design Fits into the EB64 System 5–2
5.2.1 General Features of the EB64 System 5–2
5.3 Steps in the Bootstrap Process 5–3
5.4 Building the Bootstrap Image 5–5
5.4.1 Building a Bootable Image 5–5
5.4.2 Contents of Memory During Bootstrap Process 5–6
5.4.3 The Sysgen Utility 5–7
5.5 Memory Map Relationship for PALcode 5–9

A DECchip 21064 PALcode ECOs

A.1 Cache Flush A–2
A.2 Console Service A–4
A.3 Read Machine Check Error Summary A–5
A.4 Write Interprocessor Interrupt request A–6
A.5 Write Machine Check Error Summary A–7
A.6 Performance Monitoring Function A–8
A.7 Swap PALcode Image A–9

B Example of PALcode Interaction with System Software

B.1 PALcode Interrupt Example B–2

C Technical Support, Ordering, and Related Documentation

C.1 Technical Support C–2
C.2 Ordering Information and Part Numbers C–2
C.3 Related Documentation C–3

Glossary

Index

Figures

1–1 PALcode Role in Alpha AXP Architecture 1–3
1–2 PALcode Functions 1–5
2–1 PALcode Structure in Memory 2–5
2–2 PALcode Entry Points into Memory 2–13
2–3 Privileged CALL_PAL Hardware Entry Point Addresses . . . 2–14
2–4 Unprivileged CALL_PAL Hardware Entry Point Address . . . 2–15
3–1 PALcode Memory Map Format 3–6
3–2 Parameter Block Data Structure 3–9
3–3 Tools and Applications to Build PALcode 3–10
3–4 File Creation Process for Composite Users 3–13
3–5 File Creation Process for Custom Users 3–14
4–1 Customization Issues 4–2
4–2 Platform-Dependent and Platform-Independent Code 4–7
4–3 Modify PALcode Flow 4–9
4–4 Sample New CALL_PAL Instruction Addressing Scheme . . . 4–11
5–1 Steps in the Bootstrap Process 5–4
5–2 Sample PALcode Implementation Memory Format 5–7
5–3 Memory Layout of Bootstrap Image 5–8
5–4 Example of Relationship Between the PALcode Memory Map and the EB64 System Software Memory Map 5–9
B–1 PALcode Procedure Example for an Interrupt B–3

Tables

1–1	PALmode Environment	1–6
2–1	Invoking PALcode Conditions	2–3
2–2	DECchip 21064 Implementation of Opcodes Reserved to PALcode	2–6
2–3	Internal Processor Register Access	2–10
2–4	Privileged CALL_PAL Hardware Entry Point Addresses	2–14
2–5	Unprivileged CALL_PAL Hardware Entry Point Addresses	2–15
3–1	User Category Summary	3–2
3–2	PALcode User Categories	3–4
3–3	PALcode Memory Map Description	3–5
3–4	PALcode Parameter Block Fields	3–8
3–5	File Names and Descriptions	3–11
3–6	Description of User Categories	3–12
3–7	Composite User Product Files	3–15
3–8	Custom User Product Files	3–17
3–9	PALcode Header Files	3–19
3–10	PALcode Reset Files	3–20
3–11	PALcode Memory Management Files	3–20
3–12	PALcode Exception and Interrupt Files	3–21
3–13	PALcode Privileged CALL_PAL Files	3–21
3–14	PALcode Unprivileged CALL_PAL Files	3–22
3–15	PALcode Platform-Dependent Files	3–23
3–16	PALcode Object and Executable Files	3–24
3–17	Other Types of Files	3–24
3–18	Files Changed for PALcode Sample Implementation	3–25
4–1	Example of Modifications - Backup Cache	4–3
4–2	Example of Modifications - Memory Management	4–3
4–3	Example of Modifications - Interrupts	4–3
4–4	Example of Modifications - Stack Frame	4–4
4–5	Example of Modifications - Non-OSF/1 **PPI** CALL_PALs	4–4
4–6	Modify PALcode Procedure	4–8
4–7	Relative Address Process for CALL_PAL Functions	4–12
5–1	Sample Memory Map During an OSF/1 System Bootstrap	5–6
5–2	Memory Layout of Bootstrap Image	5–8

A–1	PALcode Variants Assignments	A–11
A–2	Parameter List for a1 Through a3	A–12
A–3	State of DECchip 21064 upon Completion of the swppal	A–13
B–1	PALcode Procedure Example for an Interrupt	B–2
C–1	Related Documentation	C–3

Preface

This preface provides information about the purpose, audience, and structure of the *DECchip 21064 PALcode System Design Guide*. It also describes the document conventions contained in this document.

PALcode Product Definition

PALcode stands for **Privileged Architecture Library code**. The DECchip 21064 PALcode product is one design and implementation of the Privileged Architecture Library functions. System designers can use this PALcode as a sample, or customize it to create their own code to support their DECchip 21064-based system designs. The DECchip 21064 PALcode product provides a standard programming interface that:

- Is tailored to the OSF/1 operating system
- Can be adapted for embedded system designs
- Can be adapted for similar operating systems

Product Purpose

The Privileged Architecture Library code (PALcode) is software that enables Alpha AXP system designers and implementors to support their DECchip 21064-based system design. PALcode provides a common programming interface for the operating system across all Alpha AXP architecture implementations. PALcode implements low-level hardware support functions such as power-up initialization, memory management control, interrupt and exception dispatching, and other functions that are impractical to implement in hardware and cannot be handled by operating system software. PALcode also supports several software functions such as privileged or **atomic** operation instructions, context swapping, or instruction emulation that does not require hardware support. PALcode software is modular; it has many user-defined parameters for easy modification of the code.

Document Purpose

This document explains the basic concepts and structure of PALcode. It explains basic PALcode product concepts necessary for system designers and implementors to customize PALcode for a DECchip 21064-based system design either within the OSF/1 operating system environment or in other operating systems based on a similar interface. It also provides PALcode installation procedures and detailed reference information. This document includes a model system design description based on the **DECchip 21064 Evaluation Board (EB64)** and integrated information from the EB64, and Software Design Tool publications.

Intended Audience and Prerequisites

This document provides information for system designers and system implementors who need to modify PALcode for their DECchip 21064-based design.

Before you use this document, you need to:

- Be familiar with the application programming interface for the OSF/1 operating system.
- Be familiar with your system's hardware configuration and component characteristics (such as registers, backup caches, I/O controller, and so on).
- Read the following documentation:
  - *Alpha Architecture Reference Manual*
  - *DECchip 21064-AA Microprocessor Hardware Reference Manual*
- Be familiar with the other Alpha AXP software design tools and their documentation. (See Appendix C.)

Document Organization

The information in this document is organized as follows:

- Chapter 1 describes the basic PALcode fundamentals that are applicable to all hardware implementations.
- Chapter 2 describes PALcode concepts that relate to the DECchip 21064 PALcode.
- Chapter 3 describes the PALcode product, the file structure, and the user categories.

- Chapter 4 describes important programming considerations that you need to review before you customize PALcode for a specific system implementation.
- Chapter 5 describes how the example design fits into the EB64 system and how to build the bootstrap image.
- Appendix A contains descriptions of recent DECchip 21064 PALcode engineering change orders (ECOs).
- Appendix B contains an example and an explanation of how PALcode interacts with the system software during an interrupt.
- Appendix C lists technical support services and related documentation.
- The Glossary defines terms that may be new and are used in this document.

Document Conventions

The following conventions are used in this document:

Convention	Description
boldface type	Boldface type in text indicates the first instance of terms defined either in the text, in the glossary, or both. It also indicates commands.
italic type	Italic type emphasizes important information, indicates variables, and indicates complete titles of manuals.
Note	Provides general information about a topic.
Warning	Provides information to prevent damage to equipment or software.

1
PALcode Fundamentals

This chapter describes the basic PALcode fundamentals that you need to know before you modify any of the PALcode routines. This chapter includes the following topics:

- PALcode Description
- The PALcode Environment: PALmode
- Special Functions Required for PALcode

1.1 PALcode Description

PALcode routines implement some necessary low-level hardware support functions that are too complex, too costly, or otherwise impractical to implement directly in the microprocessor chip's hardware, and that cannot be handled by normal operating system software, including:

- Power-up initialization, such as routines to initialize devices to a known state
- Memory management control, such as routines to fill the translation buffer
- Interrupt handling, such as code to determine what handler is requested and to collect relevant data
- Exception dispatching, such as an arithmetic exception

In some architectures, microcode handles these hardware functions, but the Alpha AXP architecture is careful not to mandate the use of microcode for reasonable chip implementations. Therefore, PALcode offers a preferable, more flexible, way to handle these hardware functions in a special environment situated between the chip and the operating system.

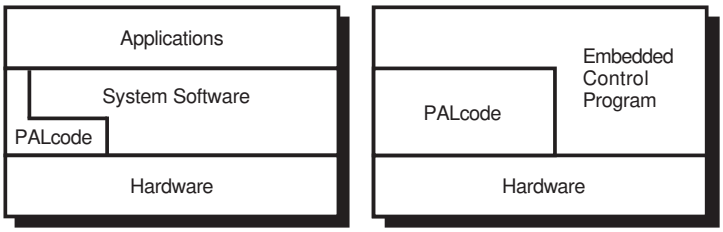
1.1.1 The Role of PALcode in the Alpha AXP Architecture

Layered applications have access to the PALcode and to the system software (Figure 1–1). This allows the layered applications to have direct access to the low-level hardware functions through the PALcode, or communicate through the system software. If the applications communicate through the system software, then the system software can either have direct access to the low-level hardware functions or pass control to the PALcode.

Embedded control programs have direct access to some of the low-level hardware functions or can pass control to the PALcode.

Figure 1–1 shows system software and an embedded control program.

Figure 1–1 PALcode Role in Alpha AXP Architecture



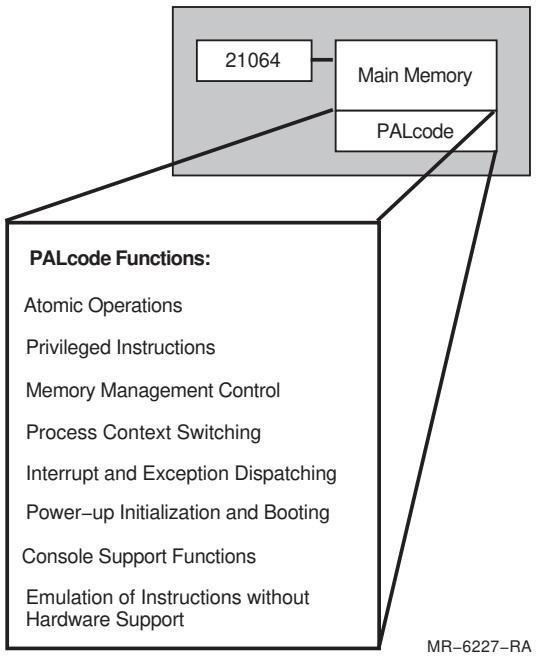
1.1.2 PALcode Functions

PALcode supports various functions that are too complex, too costly, or impractical for either the chip hardware or the operating system to handle, including:

- **Privileged** instructions
- Atomic operations, such as context swapping, returns from exceptions or interrupts that require long instruction sequences and complete access to all the underlying computer hardware
- Complex instructions, such as translation buffer fill routines or functions that were previously handled by microcode in other architectures
- Instruction emulation without hardware support

Figure 1–2 shows how PALcode provides many functions and can reside in main memory.

Figure 1–2 PALcode Functions



1.2 The PALcode Environment: PALmode

PALcode runs in a special, privileged environment called **PALmode** that is different from the normal operating environment.

Table 1–1 describes the PALmode environment.

Table 1–1 PALmode Environment

PALmode does this . . .	So PALcode can do this . . .
Disables Istream memory mapping	Implement memory management functions such as translation buffer fill.
Disables interrupts	Provide multiple instruction sequences in the form of atomic operations.
Enables the use of special reserved opcodes	Provide implementation-specific hardware functions that allow access to low-level system hardware.

1.3 Special Functions Required for PALcode

PALcode uses the Alpha AXP instruction set for most of its operations. A small number of additional functions are needed to implement the PALcode. These functions provide:

- Access to memory in a privileged fashion, such as accessing data by physical address. For example, on the DECchip 21064 this is accomplished with a HW_LD or HW_ST instruction.
- A set of instructions to access hardware control registers. For example, on the DECchip 21064 this is accomplished with a HW_MTPR or HW_MFPR instruction.
- A hardware mechanism to transition the machine from the PALcode environment to the non-PALcode environment. This mechanism loads the PC, enables interrupts, enables memory mapping, and disables PALcode privileges. For example, on the DECchip 21064 this is accomplished with the HW_REI instruction.

These instructions produce an illegal instruction trap if executed outside the PALcode environment.

2

DECchip 21064 PALcode Concepts

This chapter describes PALcode concepts that relate to the DECchip 21064, and it includes the following topics:

- Invoking PALcode
- DECchip 21064 Instruction Issue Rules
- DECchip 21064 PALmode Restrictions
- Internal Processor Register Access
- Determining CALL_PAL Entry Points

2.1 Invoking PALcode

PALcode can be invoked by the following events:

- Reset
- System hardware exceptions (**MCHK**, ARITH)
- Memory management exceptions
- Interrupts
- **CALL_PAL** instructions

PALcode is invoked at specific **entry points**, under certain well-defined conditions. PALcode can be thought of as a series of callable routines, with each routine indexed by an offset from a base address. The base address of the PALcode is programmable, is stored in the PAL_BASE **internal processor register**, and is normally set by the system reset code.

When an event occurs that needs to invoke PALcode, the DECchip 21064 does the following:

1. Drains the pipeline
2. Loads the current PC into the EXC_ADDR internal process register
3. Dispatches to the appropriate PALcode routine

Specifically, PALcode is invoked under the following conditions:

Table 2–1 Invoking PALcode Conditions

If the DECchip 21064 detects . . .	Then PALcode . . .	Comments
Reset	Resets the hardware and initializes as required with both the standard code and the system designer's PALcode.	PALcode is entered when the reset line of the DECchip 21064 is asserted.
Machine check (MCHK) error	Performs a certain level of error analysis, accesses PALmode visible registers to save state information as required by system software, and dispatches to system software.	This is the result of an uncorrected hardware error.
Arithmetic exception	Accesses PALmode visible registers to save state as required by the system software and then dispatches to the system software.	An example is arithmetic traps such as divide by zero or arithmetic overflow.
Interrupts	Performs a certain level of analysis, accesses PALmode visible registers to save state information as required by system software, and dispatches to system software.	This includes correctable hardware errors or device driver interrupts.
Dstream memory management fault	Accesses PALmode visible registers to save state as required by the system software and then dispatches to the system software.	–
Istream translation buffer miss	Refers to page tables to perform the translation buffer **(TB)** fill.	–
Istream access violation errors	Accesses PALmode visible registers to save state as required by the system software and then dispatches to the system software.	–
Dstream translation buffer miss	Refers to page tables to perform the TB fill.	–
PALmode data translation buffer miss	Refers to page tables to perform the TB fill, and returns to PALmode.	Data translation buffer miss detected while in PALmode.

(continued on next page)

Table 2–1 (Cont.) Invoking PALcode Conditions

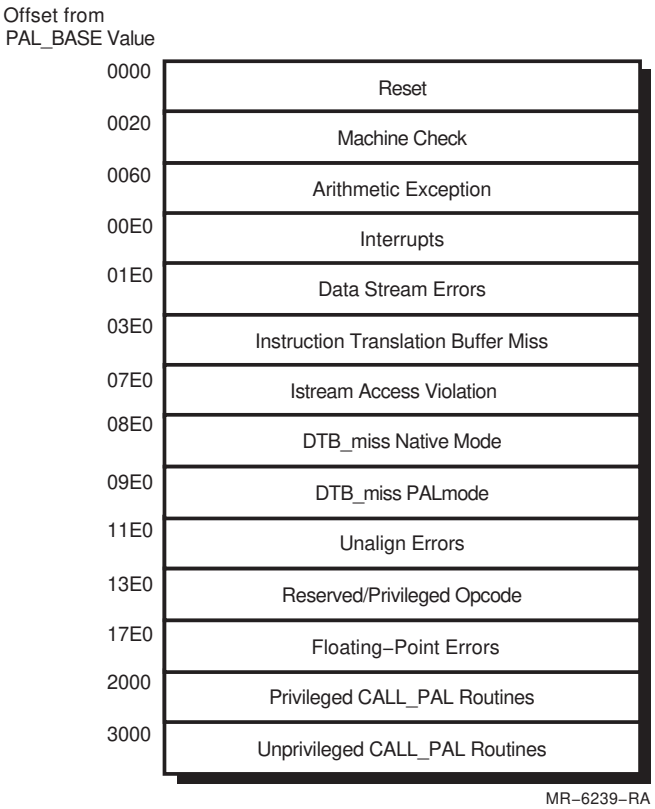
If the DECchip 21064 detects . . .	Then PALcode . . .	Comments
Dstream unalign error	Accesses PALmode visible registers to save state as required by the system software and then dispatches to the system software.	This is the result from detecting a data alignment error.
OPCDEC error	Accesses PALmode visible registers to save state as required by the system software and then dispatches to the system software.	This can be a result of attempting to execute a reserved or privileged opcode (PALcode) outside of PALmode.
Floating-point (FEN) error	Accesses PALmode visible registers to save state as required by the system software and then dispatches to the system software.	An example is the request for a floating-point operation while floating-point operations are disabled.
Privileged CALL_PAL instruction	Executes the specified CALL_PAL function.	A CALL_PAL request only from kernel mode.
Unprivileged CALL_PAL instruction	Executes the specified CALL_PAL function.	A CALL_PAL request from any user.

For more information about PALcode entry points, see the PAL Entry Points section in the *DECchip 21064-AA Microprocessor Hardware Reference Manual*.

2.1.1 PALcode Entry Points

Figure 2–1 shows the structure for PALcode entry points. For an explanation of the terms, see Table 2–1.

Figure 2–1 PALcode Structure in Memory



2.1.2 Privileged and Unprivileged CALL_PAL Functions

PALcode allows you to create a library of special functions, all of which are designated as either *privileged* or *unprivileged*. The privileged designation indicates that the function is reserved only for use in kernel mode. The unprivileged designation indicates that an instruction can be executed by any user.

The DECchip 21064 has the ability to recognize and provide hardware entry points for 64 privileged and 64 unprivileged CALL_PAL instructions with regions of 64 bytes each. This allows you to create up to 128 functions that are directly callable by the DECchip 21064. Additional CALL_PAL functions can be created, these are implemented through your OPCDEC handler.

2.1.3 DECchip 21064 Implementation of Opcodes Reserved to PALcode

PALcode is written in standard machine code with some implementation-specific extensions. Table 2–2 describes how the DECchip 21064 implements five special functions.

Table 2–2 DECchip 21064 Implementation of Opcodes Reserved to PALcode

Function	21064 Mnemonic	Opcode (Hex)	Purpose and Example

Move from processor register	HW_MFPR	19	Loads **GPR**s from **IPR**s or PALtemp registers. Example:
			`hw_mfpr r9,pt0 ; Read PALtemp 0 into r9`
			`stq r9,0(r15) ; Store into memory`

Physical load	HW_LD	1B	Loads GPRs from memory. In the following example, a physical address is used instead of a virtual address. Example:
			`hw_ld r9,0 (r15) ; Read memory into r9`
			`addq r9,r10,r11 ; Add r9, r10 and place sum into r11`

Move to processor register	HW_MTPR	1D	Loads IPR values and PALtemp registers from GPRs. Example:
			`ldq r2, 0 (r3) ; Load from memory`
			`hw_mtpr r2,pt13 ; Move to PALtemp 13`

(continued on next page)

Table 2–2 (Cont.) DECchip 21064 Implementation of Opcodes Reserved to PALcode

Function	21064 Mnemonic	Opcode (Hex)	Purpose and Example

Return from exception or interrupt	HW_REI	1E	Used at the end of PALcode routines to transfer control out of PALcode. Example:
			`hw_mtpr r2, excAddr ; Write EXC_ADDR`
			`hw_mtpr r31,0 ; 1st cycle of delay`
			`hw_rei ; Return`

Physical store	HW_ST	1F	Stores GPRs into memory. In the following example, a physical address is used instead of a virtual address.
			`subq r5,r7,r2 ; r2 <- r5 - r7`
			`hw_st r2,0(r3) ; Store r2 into memory`

Note

Refer to the dc21064.h files for information about IPR naming conventions.

2.2 DECchip 21064 Instruction Issue Rules

PALcode is subjected to the same scheduling and dual-issue rules as code running in non-PALmode. These rules affect the performance of the system, and have no effect on the instruction functioning properly.

Here are some examples of the scheduling and dual-issue rules for the DECchip 21064:

- No LD instructions can be issued in the two cycles immediately following an STC.
- No floating-point operate instruction can be issued exactly five or exactly six cycles before a floating-point divide completes.

Note

A complete list of these rules is provided in the Instruction Issue Rules section in the *DECchip 21064-AA Microprocessor Hardware Reference Manual*. Carefully review the complete list of rules before you customize the PALcode routines.

2.3 DECchip 21064 PALmode Restrictions

The DECchip 21064 has certain rules governing the special PALmode environment in which PALcode routines operate.

Warning

Violations of these rules may result in unpredictable chip behavior.

Many of the PALmode restrictions involve waiting *n* cycles before using the results of a PALmode instruction. As a system designer, you can use the wait cycles efficiently for PALmode routines. Because the DECchip 21064 can issue up to two instructions per cycle for particular sequences of instructions, care must be taken in calculating the total number of wait cycles that the DECchip 21064 actually consumes.

Inserting *n* instructions between the two time-sensitive instructions is the typical method of waiting for *n* cycles. The DECchip 21064 can issue up to two instructions per cycle, which may require you to write code that requires $2 * n + 1$ instructions to wait *n* cycles.

Note that for the DECchip 21064, two copies of the identical instruction cannot be issued in the same cycle.

Here are examples of some of the rules:

- HW_MTPR instructions require at least 4 cycles to update the selected IPR.
- The first cycle (the first one or two instructions) at all PALcode entry points cannot execute a conditional branch instruction or any other instruction that uses the JSR stack hardware.

Note

A complete list of these rules is provided in the PALmode Restrictions section in the *DECchip 21064-AA Microprocessor Hardware Reference Manual*. Carefully review the complete list of rules before you customize the PALcode routines.

2.4 Internal Processor Register Access

While in PALmode, access is permitted to the **PAL_TEMP**, and the internal processor register sets. The following table defines the register naming conventions for the mnemonic, and provides the access type.

Table 2–3 Internal Processor Register Access

Mnemonic	Definition	Access Type
TB_TAG	Translation buffer tag register	Write
ITB_PTE	Instruction translation buffer page table entry register	Read/Write
ICCSR	Instruction cache control and status register	Read/Write
ITB_PTE_TEMP	Instruction translation buffer page table entry temporary register	Read
EXC_ADDR	Exceptions address register	Read/Write
SL_RCV	Serial line receive register	Read
ITBZAP	Instruction translation buffer ZAP register	Write
ITBASM	Instruction translation buffer ASM register	Write
ITBIS	Instruction translation buffer IS register	Write
PS	Processor status register	Read/Write
EXC_SUM	Exception summary register	Read/Write
PAL_BASE	PAL Base address register	Read/Write
HIRR	Hardware interrupt request register	Read
SIRR	Software interrupt request register	Read/Write
ASTRR	Asynchronous trap request register	Read/Write
HIER	Hardware interrupt enable register	Read/Write
SIER	Software interrupt enable register	Read/Write
ASTER	AST Interrupt enable register	Read/Write
SL_CLR	Clear serial line interrupt register	Write
SL_XMIT	Serial line transmit register	Write
DTB_CTL	Data translation buffer control register	Write

(continued on next page)

Table 2–3 (Cont.) Internal Processor Register Access

Mnemonic	Definition	Access Type
DTB_PTE	Data translation buffer page table entry register	Read/Write
DTB_PTE_TEMP	Data translation buffer page table entry temporary register	Read
MM_CSR	Memory management control and status register	Read
VA	Virtual address register	Read
DTBZAP	Data translation buffer ZAP register	Write
DTBASM	Data translation buffer ASM register	Write
DTBIS	Data translation buffer invalidate single register	Write
BIU_ADDR	Bus interface unit address register	Read
BIU_STAT	Bus interface unit status register	Read
DC_STAT	Data cache status register	Read
FILL_ADDR	Fill address register	Read
ABOX_CTL	Abox control register	Write
ALT_MODE	Alternate processor mode register	Write
CC	Cycle counter register	Write
CC_CTL	Cycle counter control register	Write
BIU_CTL	Bus interface unit control register	Write
FILL_SYMDROME	Fill syndrome register	Read
BC_TAG	Backup cache tag register	Read
FLUSH_IC	Flush instruction cache register	Write
FLUSH_IC_ASM	Flush instruction cache ASM register	Write
PAL_TEMP [31..0]	PAL_TEMP Internal processor registers	Read/Write

Note

For more information about register access, see the HW_MFPR Instructions and HW_MTPR Instructions sections in the *DECchip 21064-AA Microprocessor Hardware Reference Manual*.

2.5 Determining CALL_PAL Entry Points

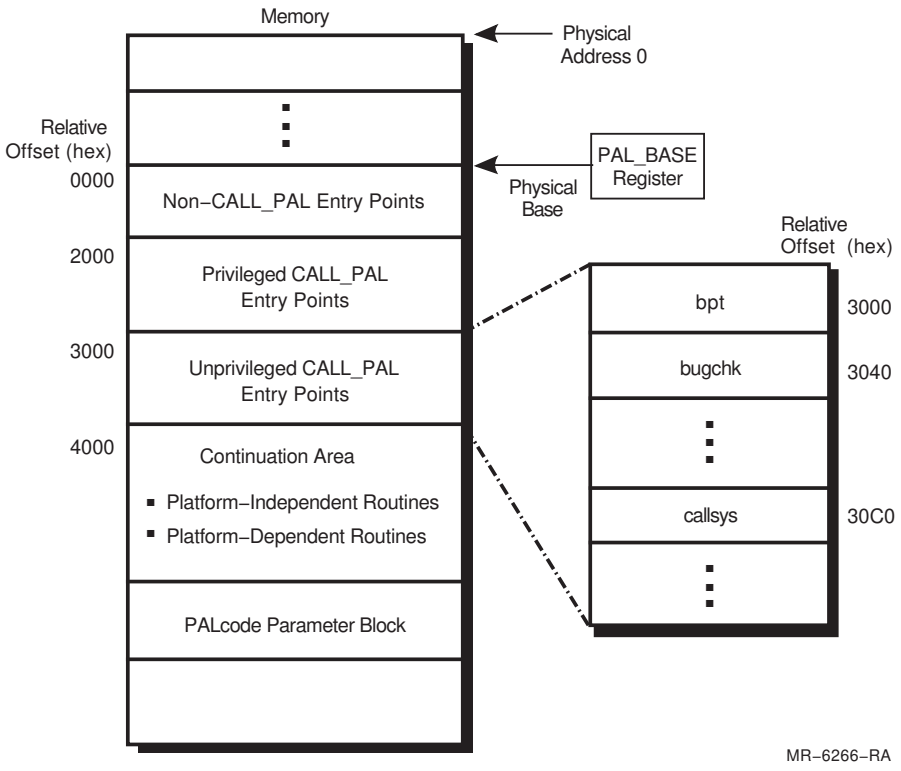
This section describes how a CALL_PAL entry point is determined.

As described in Section 2.1, all of the PALcode entry points are offsets relative to the PAL_BASE register value. For each CALL_PAL function, an additional offset is used to create an individual entry point address.

PALcode is invoked at specific entry points, which are offsets relative to the value of the physical base address stored in the PAL_BASE register (Figure 2–2). Privileged CALL_PAL functions have a relative offset starting at 2000 (hex), while unprivileged CALL_PAL functions start at 3000 (hex). To produce the actual entry point for the specific CALL_PAL function, an additional offset is created by shifting the lower 6 bits of the value in the CALL_PAL instruction's function field. These bits are shifted left 6 bits and then combined with the base address value (privileged or unprivileged) to create the offset of the actual entry point. The value of the entry point offset is then added to the physical base value, resulting in a physical address.

Figure 2–2 shows the PALcode entry points into memory.

Figure 2–2 PALcode Entry Points into Memory



For more information about the structure of PALcode in memory, see Section 2.1.1.

2.5.1 Physical Addressing of CALL_PAL Functions

This section describes how the entry point addresses are determined for privileged and unprivileged instructions.

Determining Privileged CALL_PAL Hardware Entry Point Addresses

Figure 2–3 shows the process and Table 2–4 describes the steps for generating the relative offset for a privileged CALL_PAL instruction.

Figure 2–3 Privileged CALL_PAL Hardware Entry Point Addresses

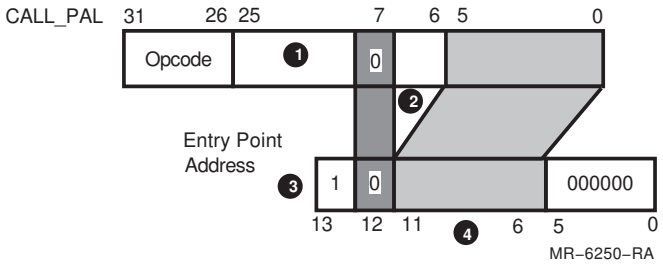


Table 2–4 Privileged CALL_PAL Hardware Entry Point Addresses

Step	Task
❶	A 26-bit function code field is read. *
❷	Bit 7 of the CALL_PAL function is clear, determining this function to be privileged.
❸	A privileged value of 2000 (hex) is provided as a relative base address.
❹	Bits <5:0> are shifted left 6 bits and then combined with the relative base address of 2000 (hex), resulting in a unique relative offset address.

*For a complete list of the function codes, see the Privileged OSF/1 PALcode Function Codes section in Appendix C of the *Alpha Architecture Reference Manual.*

Determining Unprivileged Hardware CALL_PAL Entry Point Addresses

Figure 2–4 shows the process and Table 2–5 describes the steps for generating the relative offset for an unprivileged CALL_PAL instruction.

Figure 2–4 Unprivileged CALL_PAL Hardware Entry Point Address

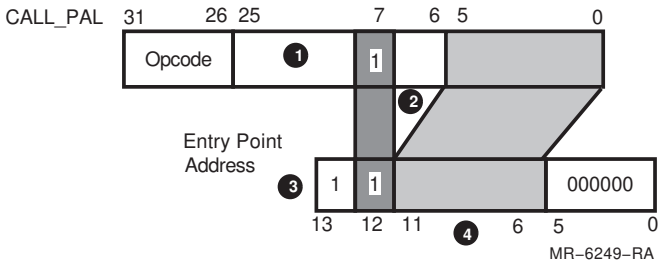


Table 2–5 Unprivileged CALL_PAL Hardware Entry Point Addresses

Step	Task
❶	A 26-bit function code field is read. *
❷	Bit 7 of the CALL_PAL function is set, determining this function to be unprivileged.
❸	An unprivileged value of 3000 (hex) is provided as a relative base address.
❹	Bits <5:0> are shifted left 6 bits and then combined with the relative base address of 3000 (hex), resulting in a unique relative offset address.

*For a complete list of the function codes, see the Unprivileged OSF/1 PALcode Function Codes section in Appendix C of the *Alpha Architecture Reference Manual.*

See Section 4.3 for information about addressing schemes for new CALL_PAL instructions.

3

PALcode Product Design and Development Concepts

This chapter describes the organization of the PALcode files and the user categories, and it includes the following topics:

- What is the PALcode Product?
- User Categories and Support Considerations
- PALcode Memory Map
- PALcode Parameter Block
- PALcode Development Concepts
- Using the PALcode Violation Checker
- Files For Each User Category
- Complete PALcode Files Listing
- Sample PALcode Implementation

3.1 What is the PALcode Product?

The PALcode product consists of the following:

- A library of routines, with each routine written as a separate **module**, or distinct section of code, so that you can change, supplement, or replace parts of the library with your own customized code as necessary.
- A model system design, based on the EB64 Evaluation and Development System, that serves as an example of a system implementation that you could achieve by using the PALcode routines.
- The *DECchip 21064 PALcode System Design Guide* (this document), which explains basic concepts and the tasks involved with customizing PALcode routines.

The PALcode product provides a common programming interface for DECchip 21064 implementations tailored to the OSF/1 operating system, and provides a basis for an interface to other similar operating systems or control programs.

3.2 User Categories and Support Considerations

The PALcode product is structured into a two-tiered design and support approach. This approach allows you the flexibility to perform as much or as little PALcode modification as you deem necessary for your system solution. The two user categories are defined in the following table:

Table 3–1 User Category Summary

Category	Amount of PALcode Change	Application Support	Design Constraints	Comments
Composite	Edits or adds modules targeted to platform	More support possible	Digital **OSF/1 PPI**-compliant on **EB64**	User modifies or adds code modules tailored to the platform configurations
Custom	Edits source code	Less support possible	None	Source code is provided to user; all original source can be altered

The following list summarizes some of the variables that distinguish PALcode users:

- Time—How much development time is available?
- Programming resources—Are programming resources, such as tools and people, available?
- Support—What are your PALcode support requirements?
- System implementation—What is your intended system design using the DECchip 21064?
- EB64 Evaluation and Development System—Do you plan to use the EB64 Evaluation and Development System?
- Operating system—What operating system do you intend to use with the DECchip 21064?
- Amount of change—What is the level of change you require to implement your system design?

3.2.1 User Categories and Descriptions

PALcode has been developed to address the needs of users who expect to make either many or few changes to PALcode routines. The PALcode product kit provides all of the files needed for both user categories. You can choose to implement your PALcode by either method. A complete list of PALcode files is provided later in this chapter. The following table briefly describes the two PALcode user categories:

Table 3–2 PALcode User Categories

User Categories	User Receives	Comments
Composite	• A binary object file of the fully-assembled, **platform-independent**, PALcode routines. These routines do not require customization for DECchip 21064 and OSF/1 operating system implementations. This file is supported by Digital. • **Platform-dependent** source and header files and the Makefile required to create the updated PALcode image to load into memory. This allows you to customize as much as necessary.	The **composite user** is implementing a system design similar to the EB64 Evaluation and Development System. This user implements a system software that is Digital OSF/1 PPI-compliant. This user may customize such things as the **Bcache** size and speed, or have a different I/O subsystem.
Custom	• All PALcode source files that you can customize as much as necessary. This includes header files and the Makefile required to create the updated PALcode image you will load into memory. • Maximum flexibility to use the DECchip 21064 with specially-designed boards or with an operating system not supported by Digital.	The **custom user** implements a system design that may or may not use the EB64 Evaluation and Development System or may use it very differently from the PALcode Sample Implementation provided in the product kit.

3.3 PALcode Memory Map

This section provides information about the PALcode **Memory Map** including the map structure and contents. See Section 5.5 for a description of the relationship between the PALcode Memory Map and a sample System Software Memory Map Layout.

PALcode Memory Map Description

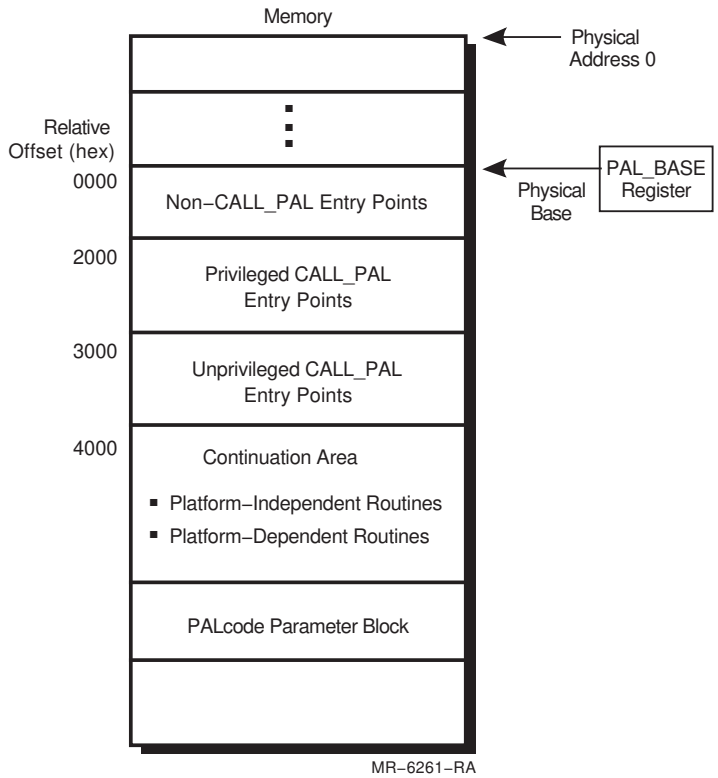
Table 3–3 describes the contents of the PALcode Memory Map (Figure 3–1).

Table 3–3 PALcode Memory Map Description

Map Item	Description
Relative offset	An offset value which when added to the PAL_Base register value forms the actual physical address.
Non-CALL_PAL entry points	The offset values for the entry points to PALcode routines. For more information, see Section 2.1.1.
Privileged CALL_PALs	CALL_PAL functions called from kernel mode.
Unprivileged CALL_PALs	CALL_PAL functions called from any user mode.
Continuation area	Memory space allocated for completion of long PALcode routines.
• Hardware platform-independent routines	Routines that will function on all Alpha AXP platforms.
• Hardware platform-dependent routines	Routines that are customized for a unique hardware implementation.
PALcode parameter block	Data structure for PALcode containing parameter values of system attributes. These attributes are accessed by system-independent code.

Figure 3–1 shows the PALcode Memory Map format.

Figure 3–1 PALcode Memory Map Format



3.4 PALcode Parameter Block

This section describes the PALcode parameter block.

Purpose

The PALcode parameter block is a structure in memory that contains data and addresses that are accessible to PALcode and users. The PALcode parameter block contains the following:

- Pointers to the PALcode **impure (scratch) area** and the **HWRPB**
- Initial system-dependent IPR settings for the BIU_CTL, ABOX_CTL, and the ICCSR
- Interrupt mask
- PALcode size and revision information
- Entry point location in the system software after a reset occurs
- Argument list value (argv) that is passed to the system software

Benefits

The PALcode parameter block can be thought of as a *single point of change* that allows:

- System designers to modify data that the PALcode routines use
- The system-independent portions of PALcode to have access to the system-dependent values

PALcode Parameter Block Fields

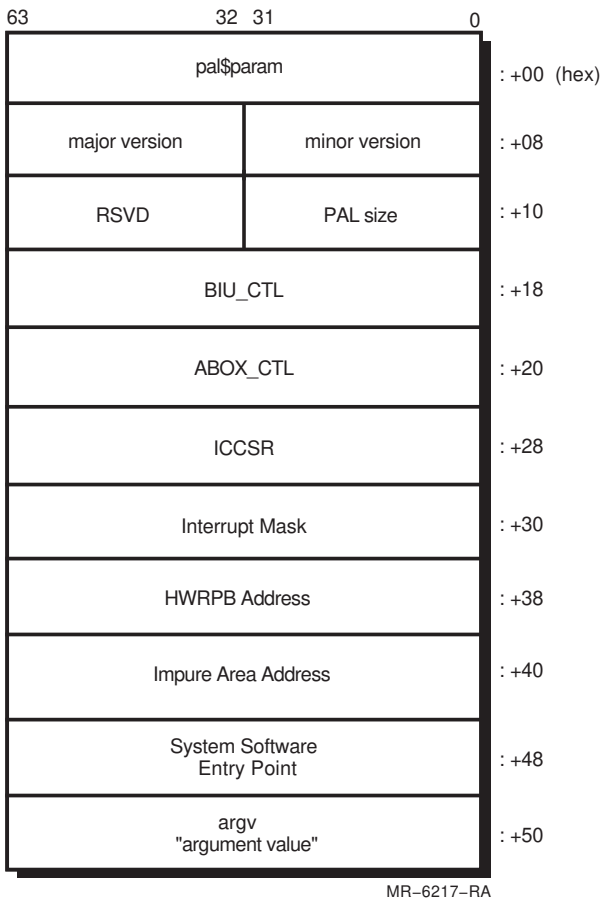
Table 3–4 describes the fields in the PALcode parameter block, and Figure 3–2 shows the structure.

Table 3–4 PALcode Parameter Block Fields

Field	Description
pal$param	Contains the offset to the parameter block in memory
major version	PALcode major version number
minor version	PALcode minor version number
RSVD	Reserved for future use
PAL size	Size of PALcode text and data segments in bytes
BIU_CTL	Initial BIU_CTL value
ABOX_CTL	Initial ABOX_CTL value
ICCSR	Initial ICCSR value
Interrupt Mask	Interrupt mask value (in hex)
HWRPB Address	Physical address of HWRPB *
Impure Area Address	Physical address of PAL scratch area
System Software Entry Point	Physical address of start of system software*
argv "argument value"	Pointer to system software argument list*

*For more information, see Section 5.4.

Figure 3–2 Parameter Block Data Structure



3.5 PALcode Development Concepts

Figure 3–3 shows the tools and applications used to build PALcode into a bootable image. Table 3–5 describes the elements in the figure.

Figure 3–3 Tools and Applications to Build PALcode

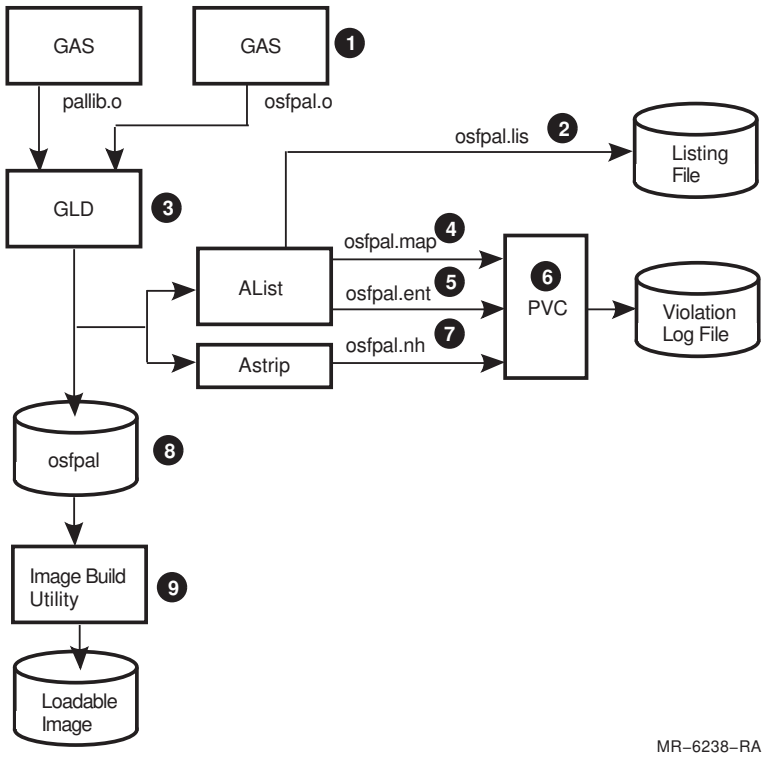


Table 3–5 File Names and Descriptions

Name	Description
❶ **GAS**	GNU-based assembler that produces the object files pallib.o and osfpal.o
❷ osfpal.lis	Source listing file created by AList
❸ **GLD**	GNU Loader needed to link GAS output
❹ osfpal.map	Provides branch and jump information to **PVC**
❺ osfpal.ent	Provides entry points of code to PVC
❻ PVC	PALcode Verification Checker
❼ osfpal.nh	Image file with header removed by Astrip
❽ osfpal	Image file of osfpal
❾ Image Build Utility	Build utility that combines PALcode and system software into a bootable image

3.6 Using the PALcode Violation Checker

The **PALcode Violation Checker** (PVC) is a tool you can use to check PALcode for timing and other coding violations. PVC searches for and identifies PALmode restrictions and violations as described in the *DECchip 21064-AA Microprocessor Hardware Reference Manual*. It is a useful way to find and correct critical PALcode coding errors, before you build the new PALcode image and load it onto your system. Chapter 4 in this book briefly summarizes how to use PVC labels and error codes in your customized PALcode routines. See the *DECchip 21064 Evaluation Board Debug Monitor User's Guide* for more detailed information about error codes, and see the *DECchip 21064 Software Design Tools User's Guide* for more information about PVC features and benefits, how to create and use PVC labels, and for a list of PVC error codes and descriptions.

3.7 Files for Each User Category

The PALcode product consists of routines that you can customize. This section considers PALcode from a *files* perspective, and describes the similarities and differences between the two PALcode categories.

3.7.1 Comparison of User Categories

Although PALcode has two different types of users, who each use the product files differently, the process that both users follow to change the files and produce the PALcode image that will boot on your operating system is much the same.

Table 3–6 describes the similarities and differences between the user types and their product files, the tools they might use, and the implementation tasks they perform to obtain the PALcode executable image OSFPAL that they load onto their system.

Table 3–6 Description of User Categories

Implementation Task	Composite User	Custom User
Change header files	Usually changes sys.h file only	Changes *.h files as necessary
Change source files	Changes sys-*.s files only	Changes *.s files as necessary
Change and run Makefile to pre-process updated PALcode image	Changes only if necessary	Probably will change significantly
Use GAS tool to assemble source files and create object file library	Creates osfpal.o file	Creates pallib.o and osfpal.o
Use GLD tool to link object files together and create executable file	Links all pallib.o and osfpal.o files together and creates updated osfpal executable file	Links pallib.o and osfpal.o files and creates updated osfpal executable file

Both users would then perform the remaining tasks described in the Modify PALcode Procedure described in Chapter 4.

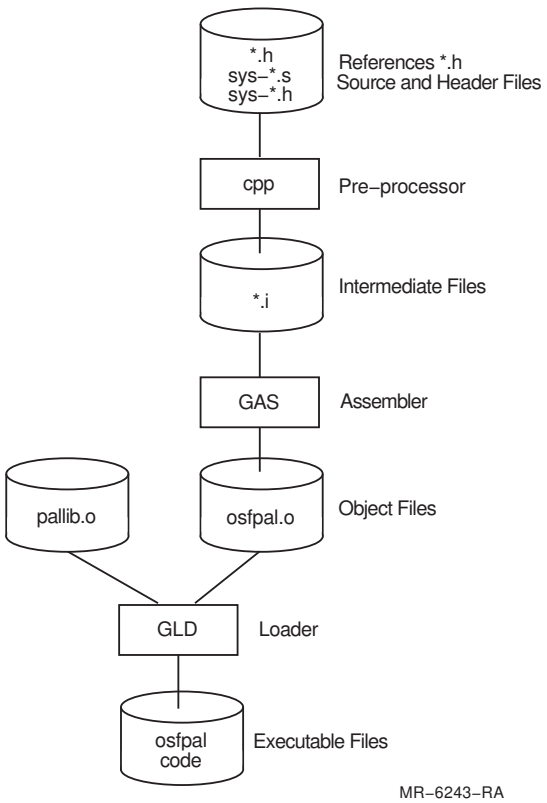
3.7.2 File Creation Process for Composite Users

Composite users need to do the following:

1. Assemble and link sys-*.s (source) files and sys-*.h (header) files to create osfpal.o (object) files.
2. Link pallib.o with osfpal.o to create osfpal executable files.
3. Run the osfpal executable files.

Figure 3–4 shows the osfpal file creation process for composite users.

Figure 3–4 File Creation Process for Composite Users



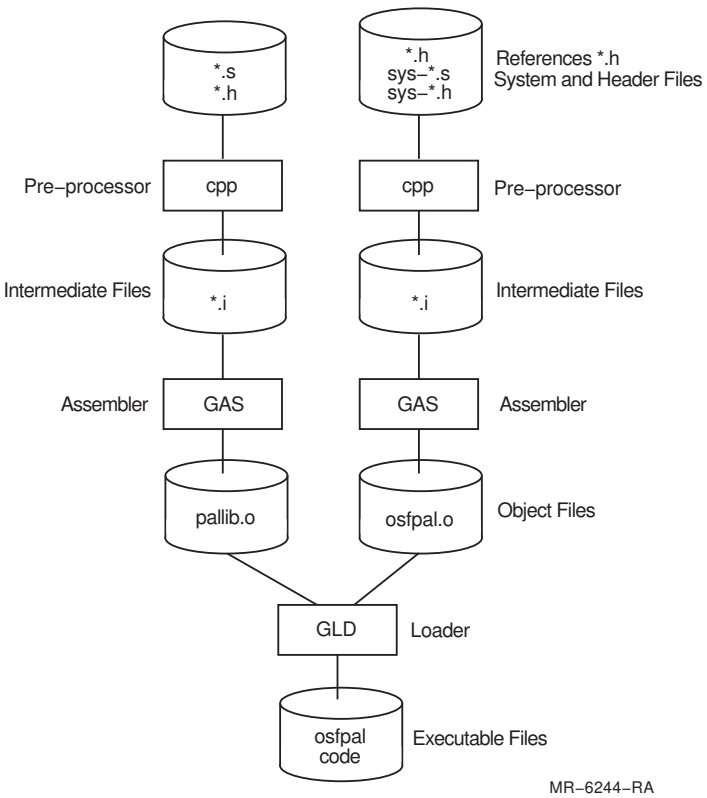
3.7.3 File Creation Process for Custom Users

Custom users need to do the following:

1. Assemble and link *.s (source) files and *.h (header) files to create pallib.o (object) files.
2. Assemble and link sys-*.s (source) files and sys-*.h (header) files to create osfpal.o (object) files.
3. Link pallib.o with osfpal.o to create osfpal executable files.
4. Run the osfpal executable files.

Figure 3–5 shows the osfpal file creation process for custom users.

Figure 3–5 File Creation Process for Custom Users



3.7.4 Composite User Product Files

Table 3–7 describes the PALcode standard product files for the composite user.

Table 3–7 Composite User Product Files

File Name	File Description	Can It Be Changed?	File Purpose
*.h	Platform-independent header or definition files	Yes (but not recommended)	Defines common headers and definitions shared across all PALcode routines. These should not need to be changed for systems running the OSF/1 operating system on the DECchip 21064.
sys.h	Platform-dependent header or definition file	Yes	Defines headers and definitions for PALcode routines that are specific to your system implementation. Some examples of platform-dependent variables include specifying the Bcache speed and size, the interrupt mask used to enable/disable external interrupts, and the location of the HWRPB data structure.
pallib.o	An object file of fully assembled, platform-independent, source files	No	Defines the routine modules that are standard for the OSF/1 operating system on the DECchip 21064.
sys-*.s	Platform-dependent source files such as those specifically for the EB64 Evaluation and Development System	Yes	Defines the PALcode routines that you need to change to reflect your specific system implementation if you are not using the EB64 Evaluation and Development System or if you are implementing it differently than the PALcode Sample Implementation.
Makefile	Command file that processes PALcode source files and creates updated PALcode image	Yes	Builds updated PALcode image and performs PVC pre-processing during the Modify PALcode Procedure.

(continued on next page)

Table 3–7 (Cont.) Composite User Product Files

File Name	File Description	Can It Be Changed?	File Purpose
Makeconf	Command file that is used in conjunction with the Makefile	Yes	Specifies input and output directories.
pal_sdg.ps	PostScript output file of *DECchip 21064 PALcode System Design Guide*	No	Explains how to use PALcode routines and sample implementation. Can be printed on a printer that prints PostScript files.
pal_sdg.txt	ASCII text output file of *DECchip 21064 PALcode System Design Guide*	No	Explains how to use PALcode routines and sample implementation. Can be displayed and scrolled on most standard terminals.

3.7.5 Custom User Product Files

Table 3–8 describes the PALcode standard product files for the custom user.

Table 3–8 Custom User Product Files

File Name	File Description	Can It Be Changed?	File Purpose
*.h	Header or definition file	Yes	Defines common headers and definitions shared across all PALcode routines.
*.s	Platform-independent source files such as those for the OSF/1 operating system on the DECchip 21064	Yes (but not recommended)	Defines the generic PALcode routines for each operating system on the particular platform. These routine modules should not need to be changed unless they are being used with an operating system that does not run on the DECchip 21064 and is not supported by Digital.
sys-*.s	Platform-dependent source files such as those specifically for the EB64 Evaluation and Development System	Yes	Defines the PALcode routines that you need to change to reflect your specific system implementation if you are not using the EB64 Evaluation and Development System or if you are implementing it very differently than the PALcode Sample Implementation.
Makefile	Command file that processes PALcode source files and creates updated PALcode image	Yes	Builds updated PALcode image and performs PVC preprocessing during the Modify PALcode Procedure.
pal_sdg.ps	PostScript output file of *DECchip 21064 PALcode System Design Guide*	No	Explains how to use PALcode routines and sample implementation. Can be printed on a printer that prints PostScript files.

(continued on next page)

Table 3–8 (Cont.) Custom User Product Files

File Name	File Description	Can It Be Changed?	File Purpose
pal_sdg.txt	ASCII text output file of *DECchip 21064 PALcode System Design Guide*	No	Explains how to use PALcode routines and sample implementation. Can be displayed and scrolled on most standard terminals.

3.8 Complete PALcode Files Listing

Each routine has been partitioned as a separate module and is provided to you in a source file form. In addition, object files and an executable file are provided. You also receive relevant header files and a Makefile that performs various building and processing tasks similar to a command file. The following tables list all PALcode files grouped into the following categories:

- Header
- Source
  - Reset
  - Memory management
  - Exception and interrupt
  - Privileged CALL_PALs
  - Unprivileged CALL_PALs
  - Platform-dependent
- Object and executable
- Other

Header Files

The following table describes the header files.

Table 3–9 PALcode Header Files

File Name	File Type	Routine Name	File Description
ev4.h	Header	Referenced by many routines	This file contains PALcode headers, values, and other information for the DECchip 21064 implementation.
macros.h	Header	Referenced by many routines	This file contains PALcode headers, values, and other information for PALcode routines.
osf.h	Header	Referenced by many routines	This file contains PALcode headers, values, and other information for the OSF/1 operating system implementation.

(continued on next page)

Table 3–9 (Cont.) PALcode Header Files

File Name	File Type	Routine Name	File Description
sys.h	Header	Called by many routines	This file contains PALcode headers, values, and other information for EB64 Evaluation and Development System implementation.

Source Files

The following tables describe the sources files.

Table 3–10 PALcode Reset Files

File Name	File Type	Routine Name	File Description
reset.s	Source	pal$reset	This file implements reset.

Table 3–11 PALcode Memory Management Files

File Name	File Type	Routine Name	File Description
dtb-fault.s	Source	pal$dtb_ fault	This file implements the Dstream TB fault handler.
itb-acv.s	Source	pal$itb_acv	This file implements the Istream access violation handler.
itb-miss.s	Source	pal$itb_miss	This file implements the Istream translation buffer miss handler.
mm-fault.s	Source	pal$mm_ fault	This file implements the common memory management fault handlers.
ndtb-miss.s	Source	pal$ndtb_ miss	This file implements the native mode Dstream translation buffer miss handler.
pdtb-miss.s	Source	pal$pdtb_ miss	This file implements the PALmode Dstream translation buffer miss handler.
unalign.s	Source	pal$unalign	This file implements the unalign data-access fault handler.

Table 3–12 PALcode Exception and Interrupt Files

File Name	File Type	Routine Name	File Description
arith.s	Source	pal$arith	This file implements an arithmetic exception handler.
fen.s	Source	pal$fen	This file implements the floating-point fault handler.
interrupt.s	Source	pal$interrupt	This file implements the interrupt handler.
mchk.s	Source	pal$mchk	This file implements the machine check handler.
opcdec.s	Source	pal$opcdec	This file implements the reserved or privileged instruction fault handler.

Table 3–13 PALcode Privileged CALL_PAL Files

File Name	File Type	Routine Name	File Description
cflush.s	Source	sys$cflush	This file implements a cflush.
cserve.s	Source	sys$cserve	This file implements a cserve.
draina.s	Source	pal$draina	This file implements a draina.
halt.s	Source	pal$halt	This file implements a halt.
rdmces.s	Source	pal$rdmces	This file implements a read machine check error summary.
rdps.s	Source	pal$rdps	This file implements a read processor status.
rdusp.s	Source	pal$rdusp	This file implements a read user stack pointer.
rdval.s	Source	pal$rdval	This file implements a read system value.
retsys.s	Source	pal$retsys	This file implements a return from system call.
rti.s	Source	pal$rti	This file implements a return from fault, trap, or interrupt.
swpctx.s	Source	pal$swpctx	This file implements a swap process context.

(continued on next page)

Table 3–13 (Cont.) PALcode Privileged CALL_PAL Files

File Name	File Type	Routine Name	File Description
swpipl.s	Source	pal$swpipl	This file implements a swap IPL.
tbi.s	Source	pal$tbi	This file implements a translation buffer invalidate.
whami.s	Source	pal$whami	This file implements a who-am-I.
wrent.s	Source	pal$wrent	This file implements a write system entry.
wrfen.s	Source	pal$wrfen	This file implements a write floating-point enable.
wrkgp.s	Source	pal$wrkgp	This file implements a write kernel global pointer.
wrmces.s	Source	pal$wrmces	This file implements a write machine check error summary.
wrusp.s	Source	pal$wrusp	This file implements a write user stack pointer.
wrval.s	Source	pal$wrval	This file implements a write system value.
wrvptptr.s	Source	pal$wrvptptr	This file implements a write virtual page table pointer.

Table 3–14 PALcode Unprivileged CALL_PAL Files

File Name	File Type	Routine Name	File Description
bpt.s	Source	pal$bpt	This file implements the breakpoint trap.
bugchk.s	Source	pal$bugchk	This file implements the bugcheck trap.
callsys.s	Source	pal$callsys	This file implements the system call.
dbgstop.s	Source	pal$dbgstop	This file implements the special breakpoint trap for the kernel debugger.
gentrap.s	Source	pal$gentrap	This file implements the generate trap.
imb.s	Source	pal$imb	This file implements the Istream memory barrier.

(continued on next page)

Table 3–14 (Cont.) PALcode Unprivileged CALL_PAL Files

File Name	File Type	Routine Name	File Description
rdunique.s	Source	pal$rdunique	This file implements the read unique value.
wrunique.s	Source	pal$wrunique	This file implements the write unique value.

Table 3–15 PALcode Platform-Dependent Files

File Name	File Type	Routine Name	File Description
sys-cflush.s	Source	sys$cflush	This file implements the platform-dependent section of cflush.
sys-crd-interrupt.s	Source	sys$crd_interrupt	This file implements the platform-dependent section of CRD interrupt.
sys-cserve.s	Source	sys$cserve	This file implements the platform-dependent section of cserve.
sys-enter-console.s	Source	sys$enter_console	This file implements the system-enter-console.
sys-get-offset.s	Source	sys$get_offset	This file implements the system-get-offset.
sys-interrupt.s	Source	sys$interrupt	This file implements the platform-dependent section of interrupt.
sys-mchk.s	Source	sys$mchk	This file implements the platform-dependent section of machine check.
sys-reset.s	Source	sys$reset	This file implements the platform-dependent section of reset.
sys-scrub-memory.s	Source	sys$scrub_mem	This file implements the system scrub memory.

Object and Executable Files

The following table describes the object and executable files.

Table 3–16 PALcode Object and Executable Files

File Name	File Type	Routine Name	File Description
pallib.o	Object	Not applicable	This file contains assembled and linked *.s and *.h files.
osfpal.o	Object	Not applicable	This file contains assembled and linked sys-*.s and *.h files, including sys.h files.
osfpal	Executable	Not applicable	This file is the resultant OSF/1 PALcode image.

Other Files

The following table describes other PALcode files.

Table 3–17 Other Types of Files

File Name	File Type	File Description
Makefile	Command	This file is used during the Modify PALcode Procedure to build the updated PALcode image and perform PVC pre-processing.
Makeconf	Command	This file is used during the Modify PALcode Procedure by the Makefile command file.
main.s	Source	This file defines the layout of the text sub-segment.

3.9 Sample PALcode Implementation

The PALcode product kit for both user types includes both the PALcode routines and PALcode implementation discussed here. The PALcode Sample Implementation, used throughout this guide, has the following characteristics:

- It was developed on a host system (such as a DECstation 3000 or DECstation 4000) that supports the EB64 software design tools.
- It supports the standard DECchip 21064 and is Digital OSF/1 PPI-compliant.
- It demonstrates a customization of PALcode for the EB64.

The PALcode Sample Implementation is one of many possible examples of how a user might customize PALcode.

Table 3–18 describes the changes in the standard product files.

Table 3–18 Files Changed for PALcode Sample Implementation

File Name	What Changed	Code Example

sys.h	Bcache size set to accommodate the EB64 Evaluation and Development System	Example of Bcache setting:
		`#define BC$K_SIZE \`
		`0x2 << BIU$V_BC_SIZE`

	Interrupt mask set to accommodate the EB64 Evaluation and Development System	Example of interrupt mask setting:
		`#define INT$K_MASK \`
		`0x00000008080A0A0A`

	HWRPB physical address specified for the EB64 Evaluation and Development System	Example of HWRPB setting:
		`#define SYS$K_HWRPB_BASE \`
		`0x100000`

sys-reset.s	Initializes the real-time clock and interrupt controller for the EB64 Evaluation and Development System	See the sys-reset.s file.
sys-interrupt.s	Interrupt cause determined (such as clock, device) as required to be Digital OSF/1 PPI-compliant	See the sys-interrupt.s file.

(continued on next page)

Table 3–18 (Cont.) Files Changed for PALcode Sample Implementation

File Name	What Changed	Code Example
sys-enter-console.s	State and switch saved to **debug monitor**	See the sys-enter-console.s file.
sys-cserve.s	Functions implemented as required by debug monitor	See the sys-cserve.s file.

4

PALcode Design Process

This chapter provides design information and procedures for customizing PALcode for your specific system implementation. It includes the following topics:

- Customization Decisions
- PALcode Modification Process
- Addressing New CALL_PAL Functions

4.1 Customization Decisions

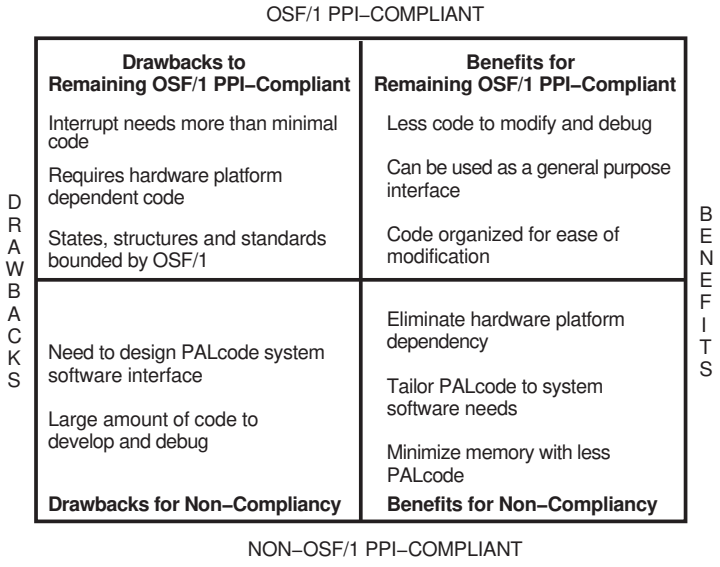
This section describes issues that you must consider before you customize PALcode. Consider the following:

- Initialization settings for specific DECchip 21064 internal processor registers (IPR) and updating PALcode data structures.
- Platform-specific customizations such as special machine check exceptions or interrupt handling.
- System software integration considerations.

Deciding on OSF/1 PPI-Compliance

As system designers, you also need to consider many issues before you modify PALcode. Figure 4–1 shows the benefits and drawbacks of being Digital OSF/1 PPI-compliant.

Figure 4–1 Customization Issues



MR–6237–RA

4.1.1 Design Decision Examples

The following examples show some of the modifications that you, as system designers, may need to make:

Table 4–1 Example of Modifications - Backup Cache

Example	Areas Affected
Alternative backup cache implementation	• IPRs impacted (BIU_CTL, ABOX_CTL,...) • Code segment altered (reset, cflush,...) • PALcode parameter block altered

Table 4–2 Example of Modifications - Memory Management

Example	Areas Affected
Memory management policy	
• One-to-one mapping • Alternate page table structure	• IPRs affected (TB_CTL, ITB_PTE,...) • Code segment altered (ITB_MISS, ITB_ACV, DTB_MISS,...)

Table 4–3 Example of Modifications - Interrupts

Example	Areas Affected
Alternative interrupt environment	
• Adding new interrupt devices on current interrupt design • Modifying interrupt pin assignments	• IPRs affected (HIER, SIRR, SIER,...) • Code segment altered (interrupt, cserve, sys-reset) • PALcode parameter block altered

Table 4–4 Example of Modifications - Stack Frame

Example	Areas Affected
Alternative interrupt/exception stack frame/dispatch • Enhancing amount of information passed to system software • Altering the system software entry	• IPRs affected - none • Code segment altered (all interrupts and exceptions, BPT, SYSCALL, WRENT, GENTRP...) • PALcode parameter block altered

Table 4–5 Example of Modifications - Non-OSF/1 PPI CALL_PALs

Example	Areas Affected
Alternative non-Digital OSF/1 PPI-compliant CALL_PALs • Creating a new operating system interface design	• IPRs affected (possibly all) • Code segment altered (all CALL_PALs)

4.2 PALcode Modification Process

This section provides information about the tools and process for modifying PALcode. Before you proceed, review Section 2.2 and Section 2.3 for rules and restrictions that may apply.

4.2.1 PVC Label Format

The PALcode Violation Checker requires a certain format for the labels it uses to check for timing violations and other errors. These labels are used in subroutines to control how PVC follows branch to and from subroutines, for computed **goto** instructions such as jump tables, or to ignore a specific branch entirely. The labels are also useful to disable a specific PVC error from an instruction in a PALcode routine.

PVC Error Message

The following is an example of a PVC error message with an error code of 24, at address 48D0. This PVC error indicates that you cannot perform a HW_REI during the 2 cycles immediately following a HW_MTPR ITBIS.

```
Error executing instruction HW_REI at address 48D0 on cycle 9!!
(PVC #24) You can't HW_REI during the 2 cycles following a HW_MTPR ITBIS.
```

Suppressing PVC Error Messages

For this example, assume that the system designers have determined this error to be harmless for their particular system implementation and have chosen to ignore this error. This error can be ignored by placing the following label at the offending instruction in the PALcode source file. This label instructs PVC to ignore this instruction, and no error is displayed the next time PVC runs.

```
PVC$Ignore_48D0$24:
```

PVC Label Format

As shown in the format of the following PVC label, PVC dictates that you start the label with the phrase PVC$. You then need to add a label name between the two dollar signs ($) to make the PVC label unique, and to indicate the address of the error you are attempting to mask. The second dollar sign must be followed by the error number that resulted in PVC finding the error. The last field is optional, and is used for branches. See the following note for more information about suppressing error messages in PVC.

```
PVC$<your_label_name>$<PVC_error_number>[.destination]:
```

Note

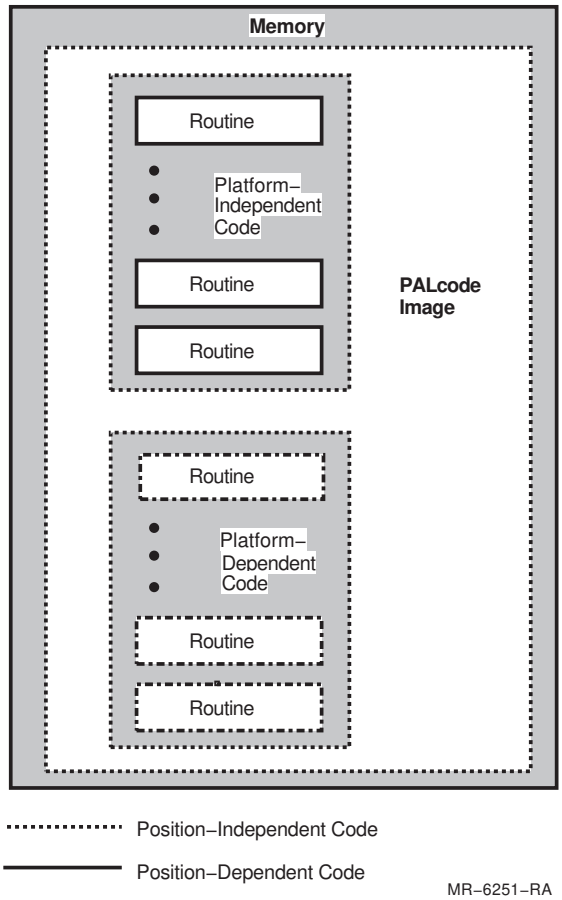
Before you customize PALcode routines, refer to the PVC section in the *DECchip 21064 Software Design Tools User's Guide* and to the *DECchip 21064 Evaluation Board Debug Monitor User's Guide* for more details about PVC labels and for the complete list of PVC error codes.

4.2.2 PALcode Assembly Rules

Follow these rules when you customize your PALcode routines:

- The Makefile assumes that the PALcode is assembled *in two halves* rather than in one monolithic assembly. Specifically, it assembles the platform-dependent half and links it to the already assembled platform-independent half.
- Routines in the platform-dependent half should not directly address the assembler's location counter, so that they remain position-independent code. Figure 4–2 shows the structure of the two halves, and it shows which routines are position-dependent code.
- Do not rely on the .align directive to align code to a page. It is more reliable to use zeros to align code within a page. See the *Alpha Architecture Reference Manual* for more details about page sizes and *The GNU Assembler* manual for more information about the *.align* directive.

Figure 4–2 Platform-Dependent and Platform-Independent Code



4.2.3 Modifying PALcode

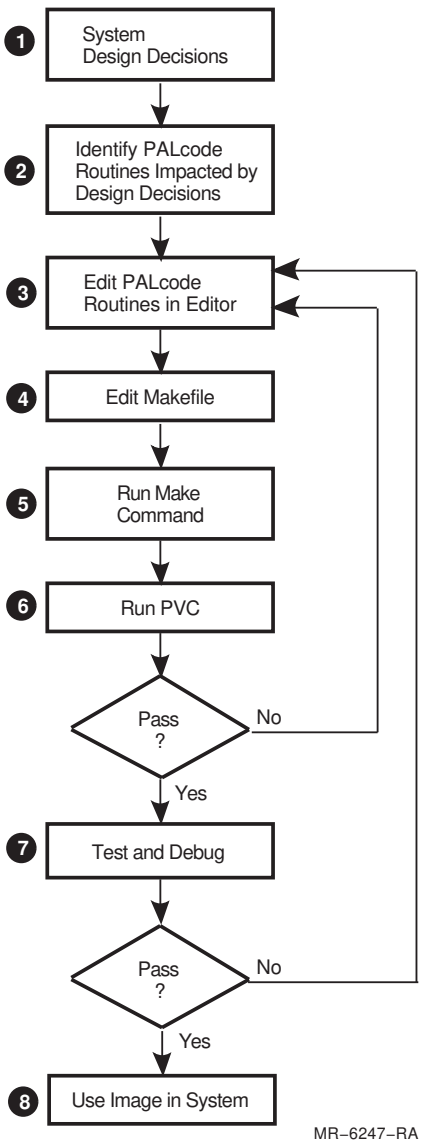
This section provides a detailed procedure that you can use to customize, assemble, and link PALcode. Before you modify the PALcode, review Section 2.2 and Section 2.3 for rules and restrictions that may apply.

Table 4–6 describes the steps and tasks you need to modify PALcode, and Figure 4–3 shows the procedure.

Table 4–6 Modify PALcode Procedure

Step	Task
❶	Determine the needs of your system. Some system decision examples are: larger cache, different interrupt strategy.
❷	Identify the PALcode routines that are impacted by the design decisions.
❸	Edit the PALcode routines you want to customize by modifying them in an editor of your choice on your host system.
❹	If you are adding or removing routines, edit the Makefile provided with your product kit.
❺	Run the Make command, which performs the following tasks: • Pre-processes the routine source files using **CPP**. • Assembles the pre-processed source files and creates two objects (independent and dependent) using GAS. • Links both objects together and creates the updated PALcode executable image called osfpal using GLD. This image is an a.out file. • Post-processes the updated PALcode image with AList and produces the following three files: – PVC entry point with a .ent file extension (such as osfpal.ent) – PVC map file with a .map file extension (such as osfpal.map) – Disassembled listing with a .lis file extension (such as osfpal.lis)
❻	Run PVC to check the osfpal image for timing violations and other errors. Remember that the **go** command checks all entry points in sequence and will take longer than the **do** command which checks a specific entry point. If you encounter timing violations or other PVC errors, correct the affected source file and repeat steps 3 through 6 in this procedure.
❼	Verify that your new PALcode performs as intended. If any modifications are necessary, return to the editing stage and repeat all steps.
❽	Create and load bootstrap image into memory. The operating system or application runs and operates normally if your PALcode customizations have processed successfully. If your operating system or application does not run or if it behaves abnormally, then debug your customized PALcode and repeat the steps necessary for this procedure.

Figure 4–3 Modify PALcode Flow



Note

Refer to Chapter 5 for information about the steps for building a bootable image on the sample implementation.

4.3 Addressing New CALL_PAL Functions

This section provides examples for determining the relative addresses of new CALL_PAL functions. You may need to consider changing these routines to accommodate your system implementation, or you may need to create new CALL_PAL functions for your unique system implementation. Other considerations are when:

- You are using the DECchip 21064 in an embedded system.
- You are using the DECchip 21064 with an operating system other than Digital OSF/1.

Note

Ensure that the entry point address for the newly created CALL_PAL function has been calculated correctly, based on the process in Section 2.5. Otherwise, the hardware will not dispatch to the correct memory location to execute the function properly.

Figure 4–4 shows two standard CALL_PAL functions and examples of two new CALL_PAL functions that you might create, and it shows their respective addressing schemes. Table 4–7 describes the terms used in Figure 4–4.

Figure 4–4 Sample New CALL_PAL Instruction Addressing Scheme

Type of CALL_PAL	CALL_PAL Function	Function Number	Function Description	Function Entry Points (Relative Address)
OSF/1 Unprivileged CALL_PAL Function	rdunique	9E (hex) 10011110 (binary)	Read Unique Value Returns Process Unique Value In v0	11 011110 000000 (binary) 3780 (hex)
Sample New Unprivileged CALL_PAL Function	newroutinename	82 (hex) 10000010 (binary)	This is your new unprivileged CALL_PAL	11 000010 000000 (binary) 3080 (hex)
OSF/1 Privileged CALL_PAL Function	rdusp	2E (hex) 00101110 (binary)	Read User Stack Pointer Returns The User Stack Pointer In v0	10 101110 000000 (binary) 2B80 (hex)
Sample New Privileged CALL_PAL Function	newroutinename	3A (hex) 00111010 (binary)	This is your new privileged CALL_PAL	10 111010 000000 (binary) 2E80 (hex)

MR–6225–RA

Table 4–7 Relative Address Process for CALL_PAL Functions

Column	Description
Type of CALL_PAL	The type of CALL_PAL function— *privileged* or *unprivileged*. All privileged functions have a relative address with the base starting at 2000 (hex) and all unprivileged functions have a relative address with the base starting at 3000 (hex). Using the example shown in Figure 4–4, the first row is an unprivileged CALL_PAL. *
CALL_PAL Function	The name of the CALL_PAL function, which is limited to 12 characters. As shown in the top row of Figure 4–4, the name of this function is rdunique.
Function Number	The hex and binary value of the CALL_PAL function. As shown in Figure 4–4, the rdunique function has a hex value of 9E in its function field.*
Function Description	A brief description of the function.
Function Entry Points (Relative Address)	This relative address value is the sum of the unprivileged or privileged address base value, and the function value shifted left 6 bits. In Figure 4–4, the last column has the most-significant digits set to a 3 for the rdunique function because it is an unprivileged CALL_PAL function. The next 6 bits are the bits of the function number that have been shifted left 6 bits. This value is also shown in its converted hex form as 3780.

*For a complete list of the functions, see the Unprivileged OSF/1 PALcode Functions Codes and Privileged OSF/1 PALcode Functions Codes sections in Appendix C of the *Alpha Architecture Reference Manual*.

See Section 2.5.1 for information about the physical addressing scheme for standard CALL_PAL functions supported by the DECchip 21064 microprocessor.

5

How the Example Design Fits into the EB64 System

This chapter provides information about how the standard PALcode is incorporated into the EB64 system. It includes the following topics:

- EB64 System
- How the Example Design Fits into the EB64 System
- Steps in the Bootstrap Process
- Building the Bootstrap Image
- Memory Map Relationship for PALcode

5.1 EB64 System

The EB64 (Evaluation Board) system allows you to develop code on a host system and then load the software into the evaluation board using either a serial line or the Ethernet port. This allows you to write and load code into the test system and perform software debug functions. This includes exercising embedded control products such as communication engines and video products, and layered software products for workstations and personal computer (PC) products.

5.2 How the Example PALcode Design Fits into the EB64 System

This section provides information about how the example PALcode design fits into the EB64 system. For this particular implementation, the PALcode was tailored for bootstrapping and running the OSF/1 operating system on the EB64 system. The EB64 determines the following:

- Contents of the system dependent routines (sys-*.s and sys.h files)
- Backup cache size and speed issues
- I/O issues

The EB64 does not affect the system independent PALcode routines.

5.2.1 General Features of the EB64 System

The EB64 system offers the following capabilities:

- File download through serial and Ethernet ports.
- Examine and deposit the EB64 system registers, 21064 IPRs, and I/O mapped registers.
- Examine and modify DRAM and I/O mapped memory.

- Disassembly of CPU instructions in memory.
- Transfer control to programs loaded into memory.
- Native debugging capabilities including breakpoints and single stepping.
- Full source level debugging capabilities using **DECladebug** running on a remote host communicating through an Ethernet connection.

Note

For more information about the EB64 system, refer to the *DECchip 21064 Evaluation Board Debug Monitor User's Guide*.

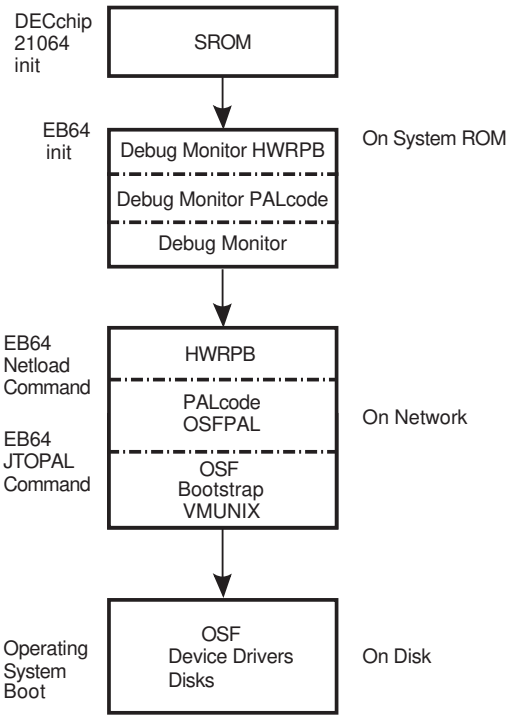
5.3 Steps in the Bootstrap Process

The following list provides an overview of the steps in the bootstrap process. Note that PALcode resides in both the debug monitor and the operating system boot images.

1. SROM code loads and activates the debug monitor.
2. Debug monitor image is loaded and activated.
3. Bootstrap image is loaded and activated.
4. OSF/1 operating system is loaded and activated.

Figure 5–1 shows the steps in the bootstrap process.

Figure 5–1 Steps in the Bootstrap Process



Refer to Appendix C for a list of other documentation that may be useful in this procedure.

5.4 Building the Bootstrap Image

This section describes the structure and contents of memory during an OSF/1 system boot process using the EB64 Debug Monitor.

5.4.1 Building a Bootable Image

The section describes how the **hardware restart parameter block (HWRPB)**, PALcode, and the OSF/1 kernel image operate together, and form a bootable image.

HWRPB

The HWRPB provides information shared by both the operating system and PALcode during bootstrap and restart situations. This includes:

- The initial state information for OSF/1 including the kernel stack pointer and the page table base register (PTBR).
- The page tables to map initial bootstrap address space.
- The address of the impure area.
- Code that executes a jump to the PALcode reset entry point.

PALcode

The PALcode needs to know:

- The system software transfer address. PALcode obtains the transfer address from the PALcode parameter block.
- The address of the impure area, for logging machine checks, and so on. PALcode obtains the impure area address from the PALcode parameter block.
- Operating system information contained in the HWRPB such as the initial kernel stack pointer, page table base register, and hardware privileged context block (HWPCB). PALcode obtains the address of the HWRPB from the PALcode parameter block.

OSF/1 Kernel

The OSF/1 kernel is provided with the following:

- Page tables for the OSF/1 host image. (**Sysgen**, which is a sample utility, is responsible for building the initial level 2 and level 3 page tables.)
- The initial page table information (such as PTBR) and the first free page frame number (PFN), which are passed in as arguments from the PALcode reset routine.

5.4.2 Contents of Memory During Bootstrap Process

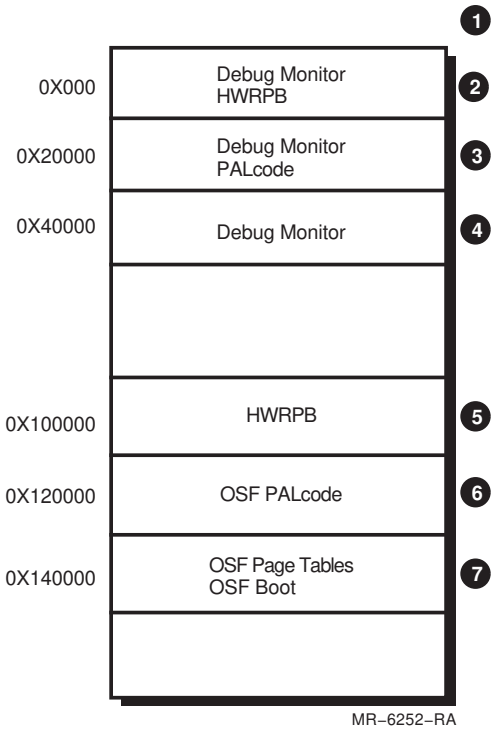
Table 5–1 describes the contents of the sample memory map shown in Figure 5–2 during an OSF/1 system bootstrap.

Table 5–1 Sample Memory Map During an OSF/1 System Bootstrap

Event	Description
SROM code loads and activates	Hardware reset event loads SROM code into Icache and executes the following:
	Performs a basic system initialization. ❶
	Loads the debug monitor into memory and transfers control to the debug monitor. ❶
Debug monitor image loaded and activated	Loaded image includes HWRPB and debug monitor PALcode.
	Control passes through HWRPB to debug monitor PALcode reset entry point. ❷
	Control passes through debug monitor PALcode reset to debug monitor main routine and displays the *EB64>* prompt. ❸
	User netloads the OSF/1 bootstrap image and then jumps to OSF/1 PALcode. ❹
Bootstrap image loaded and activated	OSF/1 bootstrap image includes HWRPB, OSF/1 kernel, and OSF/1 PALcode.
	Control passes through HWRPB to the OSF/1 PALcode reset entry point. ❺
	Control passes through OSF/1 PALcode reset to OSF/1 kernel entry point. ❻
OSF/1 operating system loaded and activated	OSF/1 operating system is loaded and activated from disk. ❼

Figure 5–2 shows a sample memory map during an OSF/1 system bootstrap.

Figure 5–2 Sample PALcode Implementation Memory Format



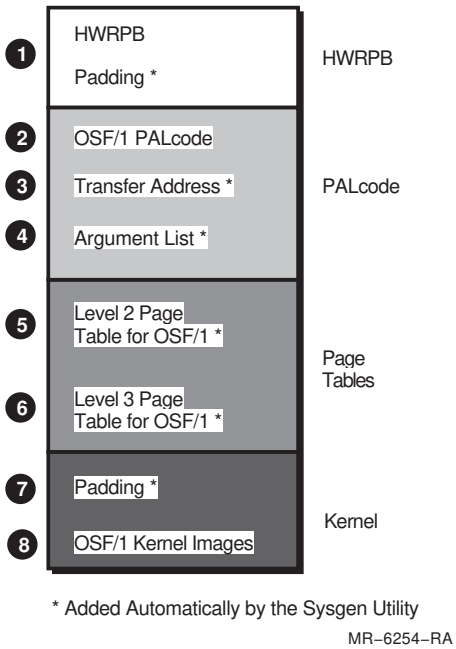
5.4.3 The Sysgen Utility

Sysgen is an *image building utility* that concatenates the HWRPB, PALcode, and the OSF/1 kernel images into one bootable image. While arranging this into one contiguous image, sysgen provides padding in memory (by filling locations with zeros) so that the OSF/1 PALcode and the OSF/1 kernel are aligned at page boundaries, and it creates a transfer address, argument list, and the level 2 and level 3 page tables for OSF/1. Refer to Table 5–2 and to Figure 5–3 for the order of events.

Table 5–2 Memory Layout of Bootstrap Image

Step	Event
❶	Places the HWRPB at the beginning of the image and appends padding to it.
❷	Appends the OSF/1 PALcode starting on an align page boundary.
❸	Appends the transfer address to the end of the OSF/1 PALcode.
❹	Appends the argument list to the end of the OSF/1 PALcode.
❺	Appends the level 2 page table for OSF/1.
❻	Appends the level 3 page table for OSF/1.
❼	Adds padding so that the OSF/1 kernel image starts on a page boundary.
❽	Appends the OSF/1 kernel image.

Figure 5–3 Memory Layout of Bootstrap Image



* Added Automatically by the Sysgen Utility

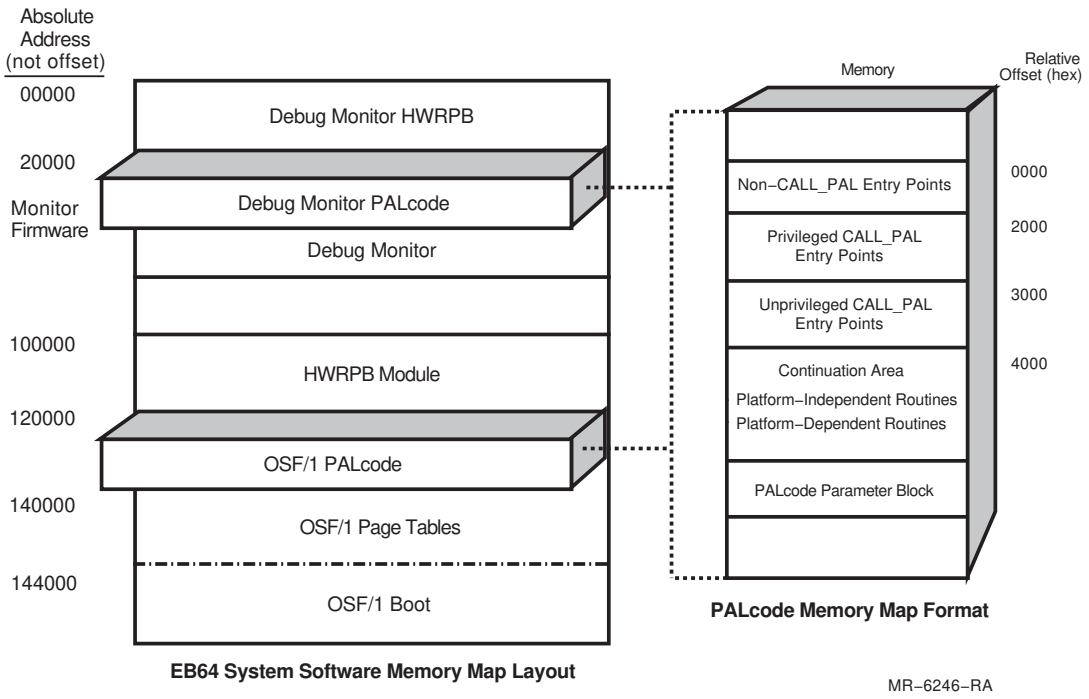
MR–6254–RA

5.5 Memory Map Relationship for PALcode

Figure 5–4 is an example of the relationship between the PALcode Memory Map and the system software on the EB64 Evaluation and Development System.

This document describes one method of incorporating PALcode into an overall design. Your software and hardware requirements may differ, and your PALcode implementation will vary accordingly.

Figure 5–4 Example of Relationship Between the PALcode Memory Map and the EB64 System Software Memory Map



See Section 3.3 for information about the PALcode Memory Map format.

A

DECchip 21064 PALcode ECOs

This appendix includes a complete description of the new instruction for each of the following DECchip 21064 PALcode ECOs:

- cflush
- cserve
- rdmces
- wripir
- wrmces
- wrperfmon
- swppal

A.1 Cache Flush

Format:

cflush !PALcode format

Operation:

```
! a0 contains the page frame number (PFN)
!       of the page to be flushed

IF  PS<mode> EQ 1  THEN
   {Initiate OPCDEC fault}

{Flush page out of cache(s)}
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

cflush Cache flush

Description:

The cflush instruction may be used to flush an entire physical page specified by the PFN in a0 from any data caches associated with the current processor. All processors must implement this instruction.

On processors which implement a backup power option that maintains only the contents of memory in the event of a powerfail, this instruction is used by the powerfail interrupt handler to force data written by the handler to the battery backed up main memory. After a cflush, the first subsequent load (on the same processor) to an arbitrary address in the target page is either fetched from physical memory or from the data cache of another processor.

Note that in some multiprocessor systems, cflush is not sufficient to ensure that the data are actually written to memory and not exchanged between processor caches. Additional platform-specific cooperation between the powerfail interrupt handlers executing on each processor may be required.

On systems which implement other backup power options (including none), cflush may return without affecting the data cache contents.

To order cflush properly with respect to preceding writes, an MB instruction is needed before the cflush; to order cflush properly with respect to subsequent reads, an MB instruction is needed after the cflush.

A.2 Console Service

Format:

cserve	!PALcode format

Operation:

```
! implementation specific

if  PS<mode> EQ 1  then
{initiate OPCDEC fault}

else
{implementation-dependent action}
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

cserve	Console service

Description:

This instruction is specific to each PALcode and console implementation and is not intended for operating system use.

A.3 Read Machine Check Error Summary

Format:

rdmces !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
        {Initiate OPCDEC fault}
endif
v0 ←  MCES
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rdmces Read machine check error summary

Description:

The read machine check error summary (rdmces) instruction returns the MCES (machine check error summary) register in v0. On return from the rdmces instruction, registers t0 and t8..t11 are UNPREDICTABLE.

A.4 Write Interprocessor Interrupt request

Format:

wripir !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
        {Initiate OPCDEC fault}
endif
IPIR ←  a0
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wripir Write interprocessor interrupt request

Description:

The write interprocessor interrupt request (wripir) instruction generates an interprocessor interrupt on the processor number passed in register a0. The interrupt request is recorded on the target processor and is initiated when the proper enabling conditions are present. On return from wripir, registers t0, t8..t11, and a0 are UNPREDICTABLE.

Programming Note

The interrupt need not be initiated before the next instruction is executed on the requesting processor, even if the requesting processor is also the target processor for the request.

A.5 Write Machine Check Error Summary

Format:

wrmces !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
        {Initiate OPCDEC fault}
endif
if (a0<0> EQ 1) then MCES<0> ←  0
if (a0<1> EQ 1) then MCES<1> ←  0
if (a0<2> EQ 1) then MCES<2> ←  0
MCES<3> ←  a0<3>
MCES<4> ←  a0<4>
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wrmces Write machine check error summary

Description:

The write machine check error summary (wrmces) instruction clears the machine check in progress bit and clears the processor or system correctable error in progress bit in the MCES register. The instruction also sets or clears the processor or system correctable error reporting enabled bit in the MCES register. On return from the wrmces instruction, registers t0, t8..t11 are UNPREDICTABLE.

A.6 Performance Monitoring Function

Format:

wrperfmon !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
        {Initiate OPCDEC fault}
! a0 contains implementation specific input values
! a1 contains implementation specific output values
! v0 may return implementation specific values
! Operations and actions taken are implementation specific
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wrperfmon Performance monitoring

Description:

The performance monitoring (wrperfmon) instruction alerts any performance monitoring software or hardware in the system that the current (active) process is to have its performance monitored. The wrperfmon function arguments and actions are platform- and chip-dependent.

Registers a0 and a1 contain implementation-specific input values. Implementation-specific values may be returned in register v0. On return from the wrperfmon instruction, registers a0, a1, t0, and t8..t11 are UNPREDICTABLE.

A.7 Swap PALcode Image

Format:

swppal !PALcode format

Operation:

```
! a0 contains the new PALcode identifier
! a1:a5 contain implementation-specific entry parameters
! v0 receives the following status:
!         0 success (PALcode was switched)
!         1 unknown PALcode variant
!         2 known PALcode variant, but PALcode not loaded

if  (PS<mode> EQ 1)  then
     (Initiate OPCDEC fault)

else
     if {0 < a0 < 256} then
         begin
            if {a0 invalid} then
               v0 ←  1
               {return}
            else if {PALcode not loaded} then
               v0 ←  2
               {return}
            else
               tmp1 ←  {PALcode base}
         end
     else
        tmp1 = a0
     {flush instruction cache}
     {invalidate all translation buffers}
     {perform additional PALcode variant-specific initialization}
     {transfer control to PALcode entry at physical address = tmp1}
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

swppal Swap PALcode image

Description:

This instruction causes the current (active) PALcode to be replaced by the specified new PALcode image. This instruction is intended for use by operating systems only during bootstraps and by consoles during transitions to console I/O mode.

The current PALcode must be able to locate the new PALcode image. After a successful PALcode switch, the new PALcode may perform minimal sanity checks and the operating system may need to update fields in the HWRPB.

PALcode switching is initiated by invoking the swppal instruction. Prior to invoking swppal, the caller should ensure that the system is quiescent.

Programming Note

It is recommended that swppal be invoked with interrupts either disabled or blocked.

PALcode switching does not implicitly load PALcode. During system bootstrap, the operating system must ensure that the desired PALcode variant is loaded. If loading is required, the operating system must allocate sufficient physically contiguous physical memory for the new PALcode image and any additional PALcode scratch space, then load the PALcode image in an implementation-specific manner.

The PALcode descriptor contained in a0 is interpreted as either a PALcode variant or the base physical address of the new PALcode image. If a variant, the PALcode image must have been previously loaded. No PALcode loading occurs as a result of this instruction.

To support PALcode replacement, all PALcode images must implement a PALcode switching entry point at the image base (offset 0). During PALcode switching, the new PALcode image receives control from the current PALcode image at this offset.

For the purposes of switching, a PALcode image is identified by one of the following:

- PALcode variant

  PALcode variants are in the range 0 < variant < 256 and permit switching between cooperating, previously loaded, PALcode images. PALcode variants are interpreted by the current PALcode without assistance from the console or operating system.

- The physical address of the switching entry point.

  Entry point addresses are used whenever the operating system or console must load a PALcode image. Entry point addresses must meet the alignment requirements of the processor implementation and may occupy the lowest memory page.

Programming Note

A common implementation is that the switching entry point is identical to the hardware reset entry point. PALcode must be able to distinguish the method of entry for each case:

- In the case of switching, PALcode must pass control to the code thread determined by the entry parameters.
- In the case of hardware reset, PALcode must perform any necessary hardware initialization and then pass control to the console or to the system software, depending upon the design of the system.

See Table A–1 for a list of the variant identifiers.

Table A–1 PALcode Variants Assignments

Variants	PALcode Type
0	Invalid
1	OpenVMS AXP
2	DEC OSF/1 (See Table A–2.)
3-127	Reserved to Digital
128-255	Reserved for non-Digital

The current PALcode is entered by the swppal instruction. After successfully switching the PALcode, the register contents are determined by the parameters passed in a1..a5 or are UNPREDICTABLE.

The PALcode image identifier (variant or switching entry point address) is contained in a0. Registers a1 through a5 contain parameters which will be passed without change to the new PALcode image. The interpretation of a1 through a5 is specific to the new PALcode image.

See Table A–2 for the use of the a1 through a3 parameters.

Table A–2 Parameter List for a1 Through a3

Register	swppal Parameter Usage
a1	New PC
a2	New PCBB
a3	New VPTB

When the PALcode is replaced, it uses the following process:

1. The current PALcode is entered by the swppal instruction. The PALcode image identifier (variant or switching entry point address) is contained in a0. Registers a1 through a5 contain parameters which will be passed without change to the new PALcode image. The interpretation of a1 through a5 is specific to the new PALcode image.
2. If the current (non-Digital supplied) PALcode does not support PALcode switching, then the current PALcode sets v0 = 1 and returns from the swppal.
3. The current PALcode determines if a0 contains a PALcode variant or switching entry point address. If the latter, execution continues at step 7.
4. The current PALcode validates the PALcode variant. If unsuccessful, the operation fails, the current PALcode sets v0 = 1 and returns from the swppal instruction.
5. The current PALcode determines if the PALcode associated with the PALcode token has been loaded. If not, the operation fails, the current PALcode sets v0 = 2 and returns from the swppal instruction.
6. The current PALcode determines the base physical address associated with the PALcode token.
7. The current PALcode branches to the new PALcode image at the switching entry point (physical) address determined in step 3 or 6.
8. The new PALcode performs any necessary implementation-specific PALcode initialization.

9. The new PALcode invalidates all TB entries and establishes the new memory management algorithm.
10. The new PALcode performs any implementation-specific actions using the entry parameters contained in a1..a5.
11. The new PALcode clears v0 and passes control to the code thread determined by the entry parameters. Control is always passed in kernel mode with interrupts disabled or blocked.

In the event of any hardware failure in accessing any of the addresses specified by the calling arguments or other dependent locations, a hardware reset and system initialization is performed.

Table A–3 shows the state of the DECchip 21064 upon completion of the swppal.

Table A–3 State of DECchip 21064 upon Completion of the swppal

Processor State	Definition	Upon exit from swppal:
ASN	Address space number	ASN in PCB passed to swppal
FEN	Floating enable	FEN in PCB passed to swppal
IPL	Interrupt priority level	7
mces	Machine check error summary	Zero
PCBB	Privileged context block	Address of PCB passed to swppal
PC	Program counter	PC passed to swppal
PS	Processor status	IPL=7, CM=K
PTBR	Page table base register	PTBR in PCB passed to swppal
unique	Processor unique value	unique in PCB passed to swppal
whami	who-am-i	Unchanged
sysvalue	System value	Unchanged
KSP	Kernel stack pointer	KSP in PCB passed to swppal
Other IPRs		UNPREDICTABLE
v0		Zero

(continued on next page)

Table A–3 (Cont.) State of DECchip 21064 upon Completion of the swppal

Processor State	Definition	Upon exit from swppal:
Integer and floating-point registers		UNPREDICTABLE, except SP and v0

B

Example of PALcode Interaction with System Software

This appendix provides an example and explanation of how PALcode interacts with the system software for an interrupt.

B.1 PALcode Interrupt Example

When the DECchip 21064 detects an interrupt, it dispatches into the PALcode to service it. The PALcode does some preliminary processing of the interrupt and then dispatches to the operating system which completes the processing of the interrupt. Upon completion of the interrupt service, the operating system then executes a CALL_PAL RTI (return from interrupt). PALcode processes the RTI and returns to the user to complete the interrupt.

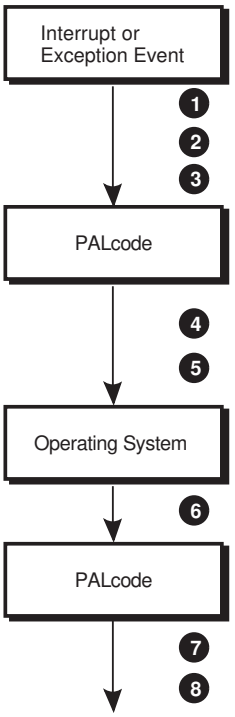
Table B–1 describes an example of how PALcode may handle an interrupt and Figure B–1 shows these steps.

Table B–1 PALcode Procedure Example for an Interrupt

Step	Description
❶	DECchip 21064 detects an interrupt or exception event, then disables interrupts, memory mapping, and drains the pipeline.
❷	The DECchip 21064 loads the current PC into the EXC_ADDR internal processor register.
❸	The DECchip 21064 dispatches to the interrupt PALcode entry point.
❹	The PALcode saves all state information on the stack and does some initial processing.
❺	The PALcode places the address of the operating system handler into EXC_ADDR. It then issues a HW_REI (hardware return from exception or interrupt) instruction to dispatch to the operating system handler.
❻	The operating system processes the interrupt and issues a CALL_PAL RTI instruction indicating the end of the interrupt processing.
❼	The PALcode executes the CALL_PAL RTI routine. It fetches the PC from the stack, and puts it into EXC_ADDR to issue an HW_REI to return to the user.
❽	The DECchip 21064 then loads the new PC, enables interrupts and memory mapping, and then dispatches back to the user.

Figure B–1 shows the PALcode procedure example for an interrupt.

Figure B–1 PALcode Procedure Example for an Interrupt



MR–6253–RA

C

Technical Support, Ordering, and Related Documentation

This appendix contains information you may need to know about the DECchip 21064 microprocessor and PALcode, such as how to call for technical support, how to order parts, and how to order related documentation.

C.1 Technical Support

Call the following phone numbers for technical support and information:

United States and Canada - 1-800-332-2717 (1-800-DEC-2717)
TTY (U.S. Only) - 1-800-332-2515 (1-800-DEC-2515)
Outside North America: +1-508-568-6868

C.2 Ordering Information and Part Numbers

To order the DECchip 21064, contact your local Digital Sales Office. Working with your sales representative, you may be able to take advantage of discounts and volume pricing.

In North America, you may also call **1-800-344-4825 (1-800-DIGITAL)** to order items. Outside North America, you must contact your local Digital Sales Office.

Part numbers are as follows:

- 21064-AA - DECchip 21064 Microprocessor (150 MHz)
- 21064-BA - DECchip 21064 Microprocessor (200 MHz)
- 21064-SA - DECchip 21064-AA Sample Kit (150 MHz)
- 21064-SB - DECchip 21064-BA Sample Kit (200 MHz)
- 2106H-AA - Heat Sink Assembly

You may also purchase the *Alpha Architecture Reference Manual* by calling the number listed previously.

C.3 Related Documentation

You can order the following documents from Digital.

Table C–1 Related Documentation

Title	Order Number
Alpha Architecture Handbook	EC-H1689-10
AMP PGA Burn-In and Product Sockets for DECchip 21064 with Alpha AXP Architecture AXP	EC-X3013-72
DECchip 21064 Bus Transactor User's Guide	EC-N0448-72
DECchip 21064 Documentation Set DECchip 21064 Microprocessor Hardware Reference Manual Alpha Architecture Handbook	EE-E0083-72
DECchip 21064 Evaluation Board Debug Monitor User's Guide	EC-N0392-72
DECchip 21064 Evaluation Board Design Package Read Me First	EC-N0352-72
DECchip 21064 Evaluation Board Product Brief	EC-N0353-72
DECchip 21064 Evaluation Board SROM Mini-Debugger User's Guide	EC-N0357-72
DECchip 21064 Evaluation Board User's Guide	EC-N0351-72
DECchip 21064 Microprocessor Hardware Reference Manual	EC-N0079-72
DECchip 21064 Software Design Tools User's Guide	EC-N0441-72
DECchip Sample Kit Brochure	EB-N0108-72
Designing a Memory/Cache Subsystem for the DECchip 21064 Microprocessor: An Application Note	EC-N0301-72
Designing a System with the DECchip 21064-AA, -BA Microprocessor: An Application Note	EC-N0107-72

(continued on next page)

Table C–1 (Cont.) Related Documentation

Title	Order Number
Alpha Architecture Reference Manual *	EY-L520E-DP-YCH*

*This book must be ordered and purchased by calling 1-800-DIGITAL.

To order any of the previously listed documents, with the exception of *Alpha Architecture Reference Manual*, use either of the following methods. To order and purchase the *Alpha Architecture Reference Manual*, call 1-800-DIGITAL.

- FAX: 1-508-351-4467 (available 24 hours) and include the following information:

  Name
  Company Name
  Street Address
  City, State or County, and Country
  Document Order Number
  Quantity

- Mail: Include the information as requested for a FAX, and send it to:

  Digital Equipment Corporation
  Order Administration NRO2-2/J6
  444 Whitney Street
  Northboro, MA 01532 USA

Glossary

atomic

An operation or sequence of events that, once begun, completes without interruption.

Bcache

An external backup cache, not physically located on the DECchip 21064 microprocessor.

CALL_PAL

Call Privileged Architecture Library. An Alpha AXP instruction that can be either privileged or unprivileged, and functions only in PALmode. CALL_PALs can emulate instructions without hardware support, can execute complex sequences such as atomic operations, and can provide support for instructions that require an interlocked memory access. The DECchip 21064 hardware directly supports up to 64 privileged and unprivileged CALL_PALs with dispatches to specific address offsets.

composite user

A user that is implementing an OSF/1 PPI-compliant system design. This user needs to customize only the platform-dependent header and source files. Modifications for this category of user may include Bcache size and speed parameters or the implementation of a unique I/O subsystem.

custom user

A user that may customize and change any source file for a system design and may or may not remain OSF/1 PPI-compliant.

CPP

C language pre-processor used in the PALcode build process.

Dcache

Data cache of 8K bytes in size, located on the DECchip 21064 microprocessor.

DECladebug

A Digital software product that provides interactive and remote debugging capabilities.

Dstream

Data stream. Data fetched from cache or memory for writing and reading.

debug monitor

A program that system designers use on the EB64 development system to enter commands for debugging and processing their customized PALcode routines and other applications they have written for the DECchip 21064 microprocessor.

DECchip 21064 Evaluation Board (EB64)

See EB64.

EB64

DECchip 21064 Evaluation Board. A DECchip 21064-based system that serves as a target to load, examine, and debug an operating system or embedded application.

embedded control program

An application that has control of all system resources (similar to an operating system). Examples include communication engines and video products.

entry point

The starting point within a program that receives control to begin a new function.

firmware

Software or a set of instructions that is stored in a fixed (wired-in) or *firm* way, usually in a read-only memory, designed to help hardware perform its assigned functions.

GAS

The GNU assembler ported to the Digital OSF/1 operating system.

GLD

The GNU loader ported to the Digital OSF/1 operating system.

GPR

General purpose registers.

hardware restart parameter block (HWRPB)

See HWRPB.

HWRPB

Hardware restart parameter block. An important data structure that is common between the firmware, PALcode, and the operating system. The HWRPB is linked into the bootable image in this example implementation.

impure area

A common data area, sometimes referred to as the scratch area, that is accessed by PALcode, operating system, and monitor software. The impure area shadows the ABOX_CTL and BIU_CTL registers, and contains MCHK logout information that is saved during a system halt.

Istream

Instruction stream. Instructions that are executing.

IPR

Internal processor register. A register internal to the DECchip 21064 microprocessor.

internal processor register (IPR)

See IPR.

machine check

See MCHK.

MCHK

Machine check. The indication that a hardware error condition was detected which may or may not be successfully corrected by hardware or PALcode.

memory map

Physical layout of code in memory.

module

A small, distinct, section of a source program. Each PALcode routine is implemented as a module. System designers use the Make command during the Modify PALcode Procedure to assemble these modules and link them into one image.

OSF/1 PPI

OSF/1 PALcode Programming Interface. A hardware implementation independent interface between the hardware and the Digital OSF/1 operating system.

PAL

Privileged Architecture Library.

PALcode

Alpha AXP Privileged Architecture Library code. Software that provides an architecturally defined programming interface that is common across all Alpha AXP processors.

PALcode Programming Interface (PPI)

See PPI.

PALcode Violation Checker

See PVC.

PALmode

The special, privileged operating environment that the DECchip 21064 microprocessor invokes whenever it executes PALcode. PALmode enables the use of reserved opcodes, and disables Istream memory mapping and interrupts.

PPI

PALcode Programming Interface. A hardware implementation independent interface between the hardware and the operating system or embedded application that conforms to the Alpha AXP architecture.

PAL_TEMP

A set of 32 internal processor registers that are used by PALcode for temporary storage.

platform-independent (OSF/1 PALcode)

The portion of the OSF/1 PALcode that executes on any of the DECchip 21064-based designs without modifications.

platform-dependent (OSF/1 PALcode)

The portion of the OSF/1 PALcode that manages the platform specific needs of a particular DECchip 21064-based design and requires modifications for different platforms.

privileged

Functions that can only be accessed from kernel mode.

Privileged Architecture Library code (PALcode)

See PALcode.

PVC

PALcode Violation Checker. A tool that checks PALcode for timing and other coding violations.

scratch area

See impure area.

SROM

Serial read-only memory. System designers can use the SROM interface to power up and initialize the DECchip 21064 microprocessor and EB64 Evaluation and Development System.

sysgen

A sample utility (used on the EB64 system) that concatenates the HWRPB, PALcode, and the OSF/1 kernel images into one bootable image.

TB

Translation buffer. A cache that contains recent virtual address translations and page protection information.

translation buffer

See TB.

unprivileged

Functions that can be accessed from any mode of execution.

Index

A

Alpha AXP documentation, C–3
Assembly rules
  programming considerations, 4–6
Audience, x

B

Bootstrap process, 5–3
Building a bootable image, 5–5

C

CALL_PAL Entry points
  determining, 2–12
  privileged, 2–6
  unprivileged, 2–6
CALL_PAL Instructions
  addressing for new instructions, 4–10
  customizing considerations, 4–10, 5–2
  physical addressing of, 2–14
Comparison
  of user categories, 3–12
Conventions
  of document, xi
Customization
  decisions, 4–2
Customize
  procedure, 4–7
Customizing
  CALL_PAL instructions, 4–10
Custom user product files, 3–17

D

Decisions
  about customizing PALcode, 4–2
Design decisions
  examples, 4–3
Development concepts
  PALcode, 3–10
Document
  audience, x
  conventions, xi
  purpose, x
  structure of, x
Documentation, C–3

E

EB64
  system software, 5–2
EB64 System
  how the example design process fits, 5–2
Example
  of PALcode interaction with system
    software, B–1

F

Files
  and user categories, 3–12
  complete listing, 3–19
  for composite users, 3–13
  for custom users, 3–14
  for object users, 3–15

Files (cont'd)
routines cross-referenced, 3–19
sample implementation file changes, 3–25

I

Instruction issue rules
programming considerations, 2–8
Intended audience, x
Internal processor register access
access to registers, 2–10
Invoking conditions
for PALcode, 2–2

M

Memory map
PALcode described, 3–5
Modifying
PALcode, 4–7
Modifying PALcode
process, 4–5
tools, 4–5

O

Object users
files listing, 3–15
Ordering information, C–2
Organization
of document, x

P

PALcode
and instruction issue rules, 2–8
and PALmode, 1–6
and PALmode restrictions, 2–8
and PVC, 3–11
and PVC label format, 4–5
assembly rules, 4–6
complete files listing, 3–19
contents of, 1–2
customizing, 4–2
description of, 1–2
PALcode (cont'd)
environment of, 1–6
files and user category, 3–12
hardware support functions, 1–2
invoking conditions, 2–2
memory map, 3–5
modify procedure, 4–7
parameter block, 3–7
purpose, ix
reserved opcodes, 2–6
sample implementation, 3–25
special functions, 1–6
structure in memory, 2–4
support considerations, 3–2
tools, 3–10
user categories, 3–2
PALcode development concepts
description, 3–10
21064 PALcode ECO
list, A–1
PALcode interrupt example, B–2
PALmode
a privileged environment, 1–6
restrictions, 2–8
Parameter block
description, 3–7
Part numbers, C–2
Physical addressing
of CALL_PAL instructions, 2–14
of new CALL_PAL instructions, 4–10
Prerequisites, x
Procedure
to modify PALcode, 4–7
Programming considerations
instruction issue rules, 2–8
PALcode assembly rules, 4–6
PVC label format, 4–5
Purpose
of document, x
of PALcode, ix
PVC
and PALcode, 3–11
PVC label format
programming considerations, 4–5

R

Related documentation, C–3
Restrictions
  PALmode, 2–8
Routines
  files cross-referenced, 3–19

S

Sample implementation of PALcode, 3–25
Sample PALcode Implementation
  memory format, 5–9
Source user product files, 3–17
Structure
  of document, x
Support considerations
  for PALcode routines, 3–2
Sysgen, 5–7
System boot process, 5–5

T

Technical support, C–2

U

User categories
  and PALcode files, 3–12
  comparison, 3–12
  description, 3–2
User prerequisites, x
Users
  description, 3–2
Utility
  sysgen, 5–7