PALcode for Alpha Microprocessors

System Design Guide

March 1995

This guide explains how to use the Privileged Architecture Library code (PALcode) to customize Alpha 21064, 21064A, 21066, 21066A, 21068, and 21164 microprocessor components to meet a variety of hardware and software application needs.

Order Number: EC–QFGLB–TE

Revision/Update Information: This document supersedes the *PALcode for Alpha Microprocessors System Design Guide* (EC–QFGLA–TE).

Software Version: EBSDK V1.4

Digital Equipment Corporation
Maynard, Massachusetts

March 1995

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

4 PALcode and the Evaluation Board

A Technical Support and Ordering Information

Glossary

Index

Figures

Tables

Preface

This preface provides information about the purpose, audience, and structure of the *PALcode for Alpha Microprocessors System Design Guide*. It also describes the conventions used in this document.

PALcode Product Definition

Privileged Architecture Library code (**PALcode**) is software that enables system designers and implementors to support their Alpha microprocessor-based system designs. The Alpha 21064, 21064A, 21066, 21066A, 21068, and 21164 microprocessor PALcode products are a design and implementation of the Privileged Architecture Library functions which support their respective microprocessors. The **Evaluation Board Software Developer's Kit (EBSDK)** provides system designers with a PALcode product that they can use as a sample. System designers can also customize PALcode to create their own code to support their Alpha microprocessor system designs. All of the EBSDK PALcode products provide a standard programming interface which:

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

Product Purpose

PALcode provides a common programming interface for the operating system across all Alpha 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 the basic PALcode product concepts that are necessary to customize PALcode for an Alpha microprocessor-based system design in an OSF/1 operating system or similar environment. It also provides PALcode installation procedures and detailed reference information.

Intended Audience and Prerequisites

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

Before using this document, system designers or implementors should:

- Be familiar with the interface between the hardware and the OSF/1 operating system.
- Be familiar with their system's hardware configuration and component characteristics (such as registers, backup caches, and I/O controller).
- Read the following documentation:
  - *Alpha Architecture Reference Manual*
  - *DECchip 21064 and DECchip 21064A Alpha AXP Microprocessors Hardware Reference Manual*
  - *DECchip 21066 and DECchip 21068 Alpha AXP Microprocessors Hardware Reference Manual*
  - *Alpha 21164 Microprocessor Hardware Reference Manual*
- Be familiar with the other Alpha software design tools and their documentation. (See Appendix A for technical support and ordering information.)

Document Organization

This document is organized as follows:

- Chapter 1 describes the basic PALcode fundamentals that apply to all hardware implementations.
- Chapter 2 describes PALcode concepts that relate to PALcode for Alpha microprocessors.
- Chapter 3 describes the PALcode product, file structure, and development process.
- Chapter 4 describes how PALcode fits into the Evaluation Board and how to build the bootstrap **image**.
- Appendix A lists technical support services and related documentation.
- The Glossary defines terms that may be new and are used in this document.

Document Conventions

In this document, the term **Alpha microprocessor** refers to the Alpha 21064, 21064A, 21066, 21066A, 21068, and 21164 microprocessors; and the term Evaluation Board refers to the 21064 Evaluation Board, the 21064 PCI Evaluation Board, the 21066 and 21068 Evaluation Board, the 21066A Evaluation Board, the 21164 Evaluation Board, and the AlphaPC64 Evaluation Board, unless noted otherwise.

The following conventions are used in this document:

Convention	Description
boldface type	Boldface type in text indicates the first instance of terms defined in the text, in the glossary, or in both places. It also indicates commands.
italic type	Italic type emphasizes important information, indicates variables, and indicates complete titles of manuals.
`monospaced type`	Monospaced type is used in interactive examples to indicate system output and user input. It is also used in code examples and other screen displays.
Extents	Extents are specified by a pair of numbers in angle brackets (<>) separated by a colon (:) and are inclusive. For example, registers <0:3> indicates an extent including registers 0, 1, 2, and 3.
Note	Notes provide general information about a topic.
UNPREDICTABLE	UNPREDICTABLE results and occurrences do not disrupt the basic operation of the processor; the processor continues to execute instructions.
Caution	Cautions provide 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 PALcode. The following topics are included in this chapter:

- PALcode Description
- The PALcode Environment: PALmode
- Opcodes Reserved for PALcode

1.1 PALcode Description

PALcode implements some necessary low-level hardware support functions, which are too complex, too costly, or otherwise impractical to implement directly in the microprocessor chip's hardware, and which 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 which 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 architecture is careful not to mandate the use of microcode for reasonable chip implementations. Therefore, PALcode offers a more flexible way to handle these hardware functions in a special environment situated between the chip and the operating system.

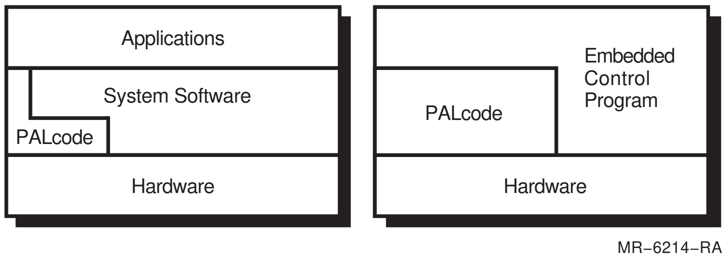
1.1.1 The Role of PALcode in the Alpha Architecture

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 to communicate through the system software. If the applications communicate through the system software, the system software can either have direct access to the low-level hardware functions or can 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 Architecture



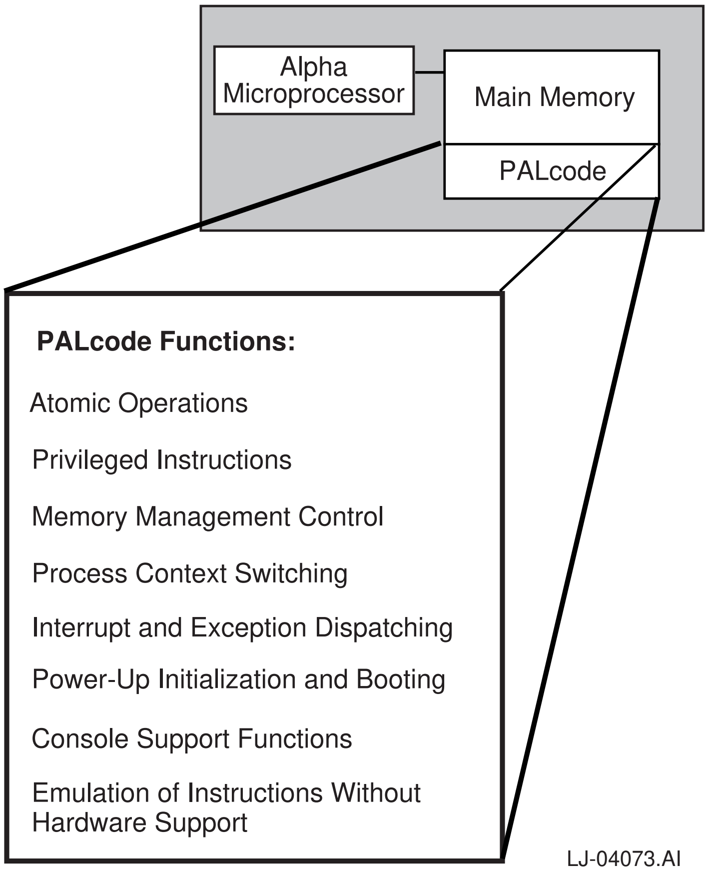
1.1.2 PALcode Functions

PALcode supports various functions, which 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 sequences, 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**, which 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 Opcodes Reserved for PALcode

PALcode is written with the standard Alpha instruction set plus five reserved opcodes to implement PALcode-specific instructions. These opcodes have implementation-specific extensions that provide access to low-level chip functions for changing states, reading, and modifying hardware control registers and performing hardware assists for various functions.

The following list provides examples of PALcode-specific instructions:

- Perform physical memory load or store operations without invoking memory management routines.
- Move data to and from **internal processor registers**.
- Transition the Alpha microprocessor from the PALmode environment to the **native-mode** environment. This includes restoring the PC, enabling interrupts and memory mapping, and disabling PALmode privileges.

These instructions produce an OPCDEC exception if executed while not in the PALmode environment.

2
PALcode Concepts

This chapter describes PALcode concepts that relate to Alpha microprocessors. The following topics are included in this chapter:

- Invoking PALcode
- PALcode Entry Mechanisms
- **CALL_PAL** Format
- CALL_PAL Entry Points
- Instruction-Issue Rules
- PALmode Restrictions
- Alpha Microprocessor Control and Status Registers

2.1 Invoking PALcode

PALcode can be invoked by the following hardware and software events:

- Reset
- System hardware exceptions (**machine check**, arithmetic trap)
- Interrupts
- Memory-management exceptions
- 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 invokes PALcode, the Alpha microprocessor 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 conditions listed in Table 2–1.

Table 2–1 Conditions for Invoking PALcode

If the Alpha microprocessor detects . . .	Then PALcode . . .	Comments
Reset	Resets the hardware and initializes as required.	PALcode is entered upon the successful completion of hardware reset.
Exception and error handling	Performs a certain level of error analysis, accesses PALmode-visible registers to save state information as required by the system software, and then dispatches to the system software.	Examples include hardware errors, OPCDEC errors, and arithmetic traps.
Interrupt	Performs a certain level of analysis, accesses PALmode-visible registers to save state information as required by system software, and then dispatches to system software.	Examples includes correctable hardware errors and device interrupts.
TB miss or memory-management fault	Calls a PALcode routine to perform a TB fill or accesses PALmode-visible registers to save state as required by the system software and then dispatches to the system software.	Examples include translation buffer misses and memory-management faults such as an access violation, invalid translation, or an unaligned access.
CALL_PAL instruction	Executes the specified CALL_PAL function.	Examples include privileged user (Kernel) CALL_PAL requests and **unprivileged** user CALL_PAL requests.

Invoking PALcode

See the microprocessor-specific hardware reference manual for more information about how an Alpha microprocessor invokes PALcode.

2.2 PALcode Entry Mechanisms

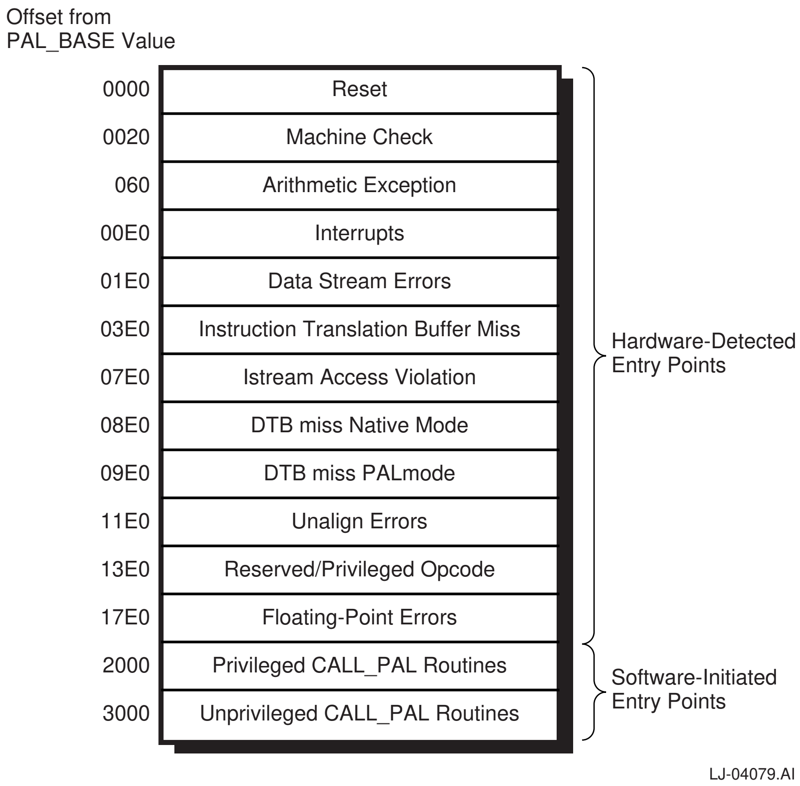
The Alpha architecture allows two methods of entry into PALcode:

- Hardware-detected—PALcode responds to a hardware event. However, the PALcode entry points and implementation are specific to the Alpha microprocessor implementation.
- Software-initiated—PALcode responds to a privileged or unprivileged CALL_PAL function. However, the PALcode entry point and method for determining the vector are specific to the Alpha microprocessor implementation.

2.2.1 21064, 21064A, 21066, 21066A, and 21068 PALcode Entry Points

Figure 2–1 shows the structure of the 21064, 21064A, 21066, 21066A, and 21068 PALcode entry points. For an explanation of the terms, see Table 2–1.

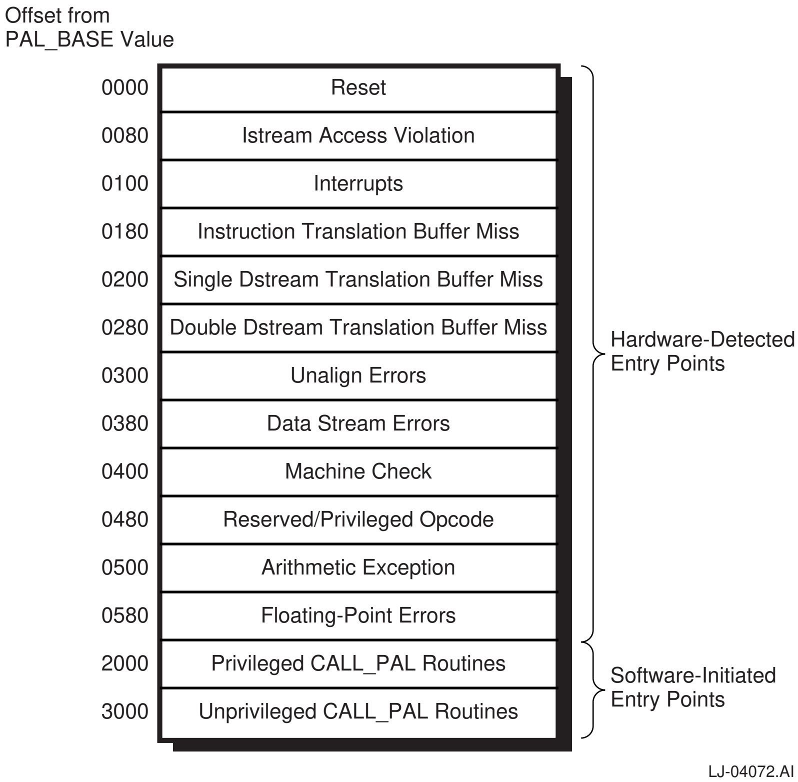
Figure 2–1 21064, 21064A, 21066, 21066A, and 21068 PALcode Memory Structure



2.2.2 21164 PALcode Entry Points

Figure 2–2 shows the structure of the 21164 PALcode entry points. For an explanation of the terms, see Table 2–1.

Figure 2–2 21164 PALcode Memory Structure



2.3 CALL_PAL Format

Figure 2–3 shows the format for a standard CALL_PAL function, which is composed of a 6-bit opcode and a 26-bit function field. The 26-bit function field specifies the entry point and indicates if the function is privileged or unprivileged. Table 2–2 describes the terms used in Figure 2–3.

Figure 2–3 Standard CALL_PAL Format

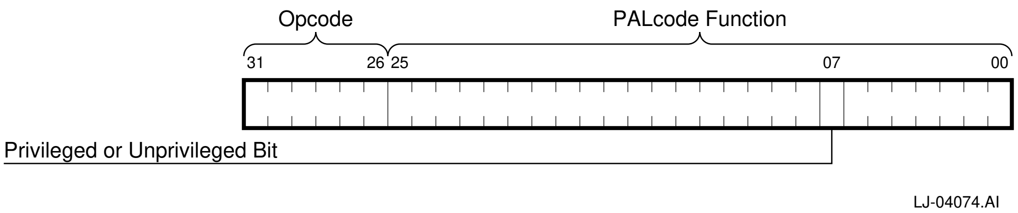


Table 2–2 Standard CALL_PAL Requirements

Term	Requirements
Opcode	A 6-bit opcode of all zeros indicates a CALL_PAL instruction.
PALcode Function	Bits in the 26-bit function code * field determine the entry point of the function. Typically, only the last 7 bits are used.
Privileged or Unprivileged bit	Bit 7 in the function code field determines whether this function is privileged or unprivileged. If bit 7 is set to a 1, the PALcode function is unprivileged.

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

2.3.1 Privileged and Unprivileged CALL_PAL Functions

All CALL_PAL functions are designated as either *privileged* or *unprivileged*. The privileged designation indicates that the function is reserved for use only in kernel mode. The unprivileged designation indicates that an instruction can be executed by any user.

The Alpha microprocessor can 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 Alpha microprocessor. When more than 128 functions are required, CALL_PAL functions can be implemented through the OPCDEC handler.

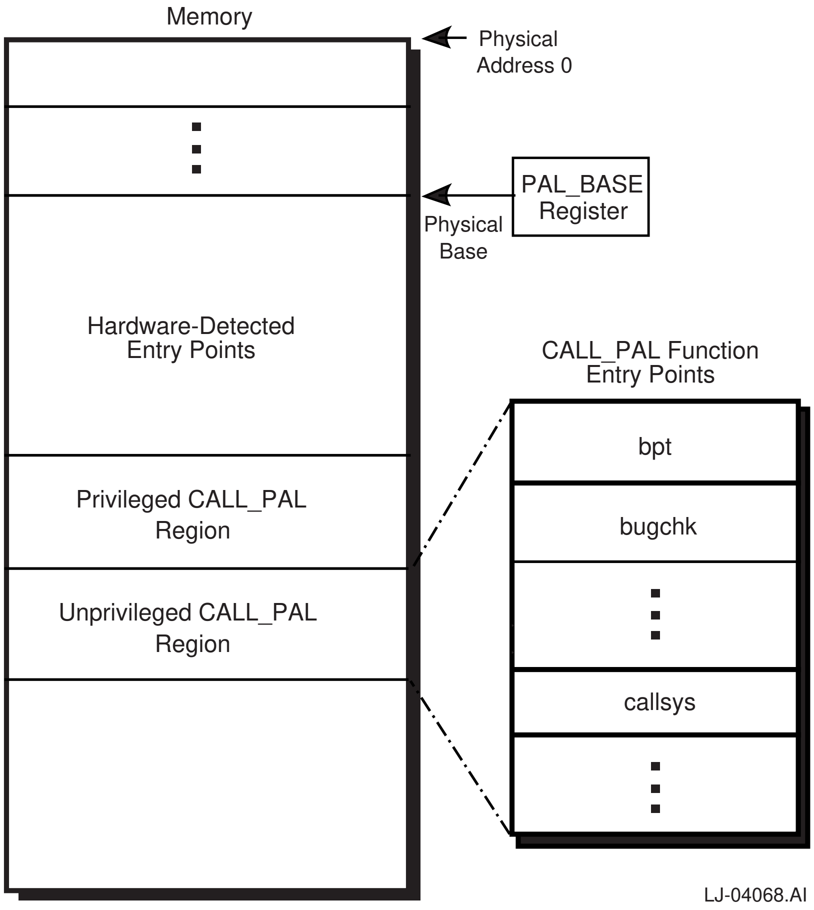
2.4 CALL_PAL Entry Points

This section describes how a CALL_PAL entry point is specified.

As described in Section 2.1, all of the PALcode entry points are offsets relative to the PAL_BASE register value. For CALL_PAL functions, bit 7 in the function field of the instruction is used to indicate if the function is privileged or unprivileged. To create the individual CALL_PAL entry point, an additional offset is created by shifting some of the bits in the CALL_PAL instruction's function field. These bits are then combined with the CALL_PAL region's base address value (privileged or unprivileged) to create the offset to the actual function entry point. The value of the function's entry point offset is then added to the physical PAL_BASE value, resulting in the physical address of a unique entry point.

Figure 2–4 shows the PALcode entry points into memory.

Figure 2–4 PALcode Entry Points into Memory



2.5 Instruction-Issue Rules

PALcode is subjected to the same scheduling and multi-issue rules as code that runs in non-PALmode. Carefully review the instruction-issue rules for performance reasons; violation of these rules will not affect the proper functioning of the instruction.

The following are examples of some of the scheduling and multi-issue rules that will affect the performance of the Alpha 21064 microprocessor:

- No LD instructions can be issued in the two cycles that immediately follow 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 microprocessor-specific hardware reference manual. Carefully review the entire list of scheduling and issuing rules before you customize the PALcode routines.

2.6 PALmode Restrictions

Alpha microprocessors have certain rules that govern the special PALmode environment in which PALcode routines operate. To ensure compliancy, a PALcode Violation Checker application has been provided with the Evaluation Board System Developer's Kit to locate rule violations. Refer to Section 3.5 for more information about the **pvc** tool.

Caution

Violations of these rules may result in unexpected chip behavior that may cause the processor to hang.

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 Alpha microprocessors can issue multiple instructions per cycle for particular sequences of instructions, you must carefully calculate the total number of wait cycles that the Alpha microprocessor actually consumes.

Inserting *n* instructions between the two time-sensitive instructions is the typical method of waiting for *n* cycles. For example, the Alpha 21064 microprocessor can issue up to two instructions per cycle, which may require you to write code that requires $2 * n + 1$ instructions in order to wait *n* cycles. Note that for the Alpha 21064 microprocessor, two copies of the identical instruction cannot be issued in the same cycle.

The following are examples of some of the rules for the Alpha 21064 microprocessor:

- A hardware move to processor register instruction requires at least four 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 rules is provided in the PALmode restrictions section of the microprocessor-specific hardware reference manual. Carefully review the complete list of rules before you customize the PALcode routines.

2.7 Alpha Microprocessor Control and Status Registers

As described in Section 1.3, PALcode is standard machine code with implementation-specific extensions that allow access to the **PAL_TEMP** and the **control and status registers** of Alpha microprocessors.

The control and status registers include the internal processor registers for all Alpha microprocessors, plus the control registers that are specific to certain Alpha microprocessors. These registers are addressed by memory addresses or internal processor register numbers. Some of the control and status registers that have memory addresses are accessible to privileged-mode programs. Most of the internal processor registers are only accessible from PALmode, requiring special opcodes such as "move from processor register."

Note

For more information about Alpha microprocessor registers, see the microprocessor-specific hardware reference manual.

3

PALcode Product Design and Development Concepts

This chapter describes the organization of the Evaluation Board System Developer’s Kit (EBSDK) PALcode files and the PALcode development process. The following topics are included in this chapter:

- What is the PALcode Product?
- EBSDK PALcode Files
- PALcode Development Concepts
- PALcode Build Process
- Using the PALcode Violation Checker
- Customization Decisions
- Design Decision Examples for Alpha Microprocessors
- Modifying PALcode

3.1 What is the PALcode Product?

The EBSDK PALcode product consists of the following:

- A library of routines, each of which is grouped into one of two separate **modules** or distinct sections of code so that you can easily change, supplement, or replace parts of the library with your own customized code.
- A model system design, based on the Evaluation Board, that serves as an example of a system implementation that you could achieve by using the PALcode routines.
- The *PALcode for Alpha Microprocessors System Design Guide* (this document), which explains basic concepts and how to customize PALcode routines.

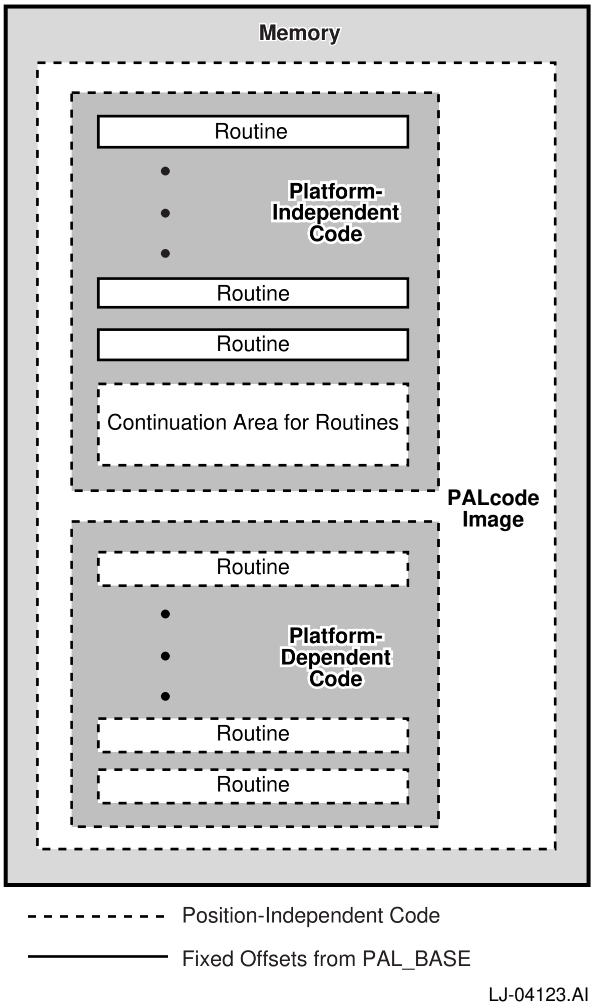
3.1.1 The EBSDK PALcode Structure

The EBSDK PALcode provides a common programming interface for Alpha microprocessor implementations that are tailored to the OSF/1 operating system and that can serve as a basis for an interface to other similar operating systems or control programs.

As shown in Figure 3–1, the EBSDK PALcode has the following characteristics:

- The PALcode image is position-independent and can be located anywhere in memory. The base address of PALcode is specified in the PAL_BASE register.
- The library of routines are grouped into one of two separate modules or distinct sections of code. These two sections are the **platform-independent** and **platform-dependent** modules.
- The platform-independent module is position-independent. The entry points within this module have a fixed (predetermined) offset relative to the base address of PALcode that limits the amount of space available for each routine. Routines that require additional space branch to a continuation area for completion.
- The platform-dependent module is position-independent.

Figure 3–1 The Structure of the PALcode Image



3.2 EBSDK PALcode Files

The EBSDK provides a complete set of associated PALcode files for each Evaluation Board. Each set of files has been tailored for a particular Evaluation Board.

The PALcode routines have been grouped into one of two source files: the osfpal.s file or the platform.s file. Both of these files are provided in source and pre-processed intermediate form. Two microprocessor-specific, platform-independent header files are provided. The dc21064.h is for the Alpha 21064, 21064A, 21066, 21066A, and 21068 Microprocessor Evaluation Boards, and the dc21164.h is for the Alpha 21164 Microprocessor Evaluation Board.

A **Makefile** that controls various building and processing tasks similar to a command file is also provided, along with a PostScript file of this document, the *PALcode for Alpha Microprocessors System Design Guide*.

Table 3–1 describes the header files, Table 3–2 describes the source files, Table 3–3 describes the intermediate and executable files, and Table 3–4 describes other PALcode files.

Table 3–1 PALcode Header Files

File Name	Can it Be Changed?	File Description
cserve.h	Yes	Contains CALL_PAL cserve sub-function encodings.
dc21064.h	Yes (but not recommended)	Contains specific definitions for Alpha 21064, 21064A, 21066, 21066A, and 21068 microprocessor implementations.
dc21164.h	Yes (but not recommended)	Contains specific definitions for Alpha 21164 microprocessor implementations.
impure.h	Yes	Contains the impure **scratch area** data structure definitions.
macros.h	Yes	Contains common macro definitions.
osf.h	Yes (but not recommended)	Contains definitions specific to the OSF/1 operating system.
platform.h	Yes	Contains platform-specific definitions.

Table 3–2 PALcode Source Files

File Name	Can it Be Changed?	File Description
osfpal.s	Yes (but not recommended)	Contains common OSF/1 PALcode for Alpha microprocessors.
platform.s	Yes	Contains platform-dependent PALcode.

Table 3–3 PALcode Intermediate and Executable Files

File Name	File Type	Can it Be Changed?	File Description
osfpal.i	Intermediate	Not applicable	Contains pre-processed osfpal.s and *.h files.
platform.i	Intermediate	Not applicable	Contains pre-processed platform.s and *.h files, including the platform.h file.
osfpal	Executable	Not applicable	Is the resultant OSF/1 PALcode image from assembling and linking the osfpal.i and platform.i files.

Table 3–4 Other PALcode Files

File Name	File Type	Can it Be Changed?	File Description
Makefile Makefile.nt	Command	Yes	Is used during the Modify PALcode Procedure to control the building and the pvc pre-processing of the updated PALcode image.
osfpal.ent	Data	Yes (but not recommended)	Contains a list of PALcode entry points and their corresponding addresses. The pvc tool inspects the PALcode entry points in this file for PALcode restriction and violations.

3.3 PALcode Development Concepts

Figure 3–2 shows the tools and applications used to build PALcode into an executable image. Table 3–5 describes the elements in the diagram.

Figure 3–2 Tools and Applications to Build PALcode

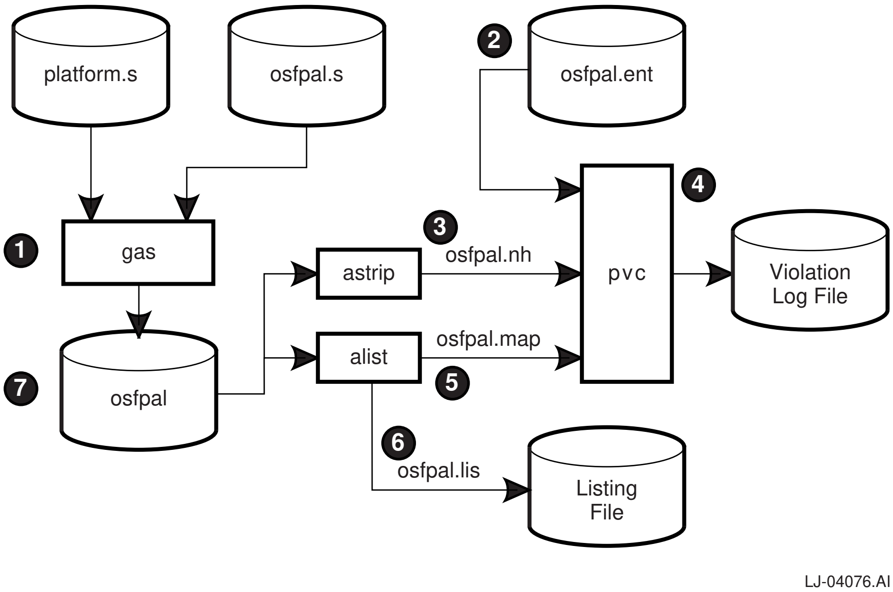


Table 3–5 File Names and Descriptions

Name	Description
❶ **gas**	GNU-based assembler, which produces the executable file osfpal
❷ osfpal.ent	Provides PALcode entry points to **pvc**
❸ osfpal.nh	Image file with header removed by astrip
❹ pvc	PALcode Violation Checker
❺ osfpal.map	Provides branch and jump information to pvc
❻ osfpal.lis	Source listing file created by alist
❼ osfpal	Image file of osfpal

3.4 PALcode Build Process

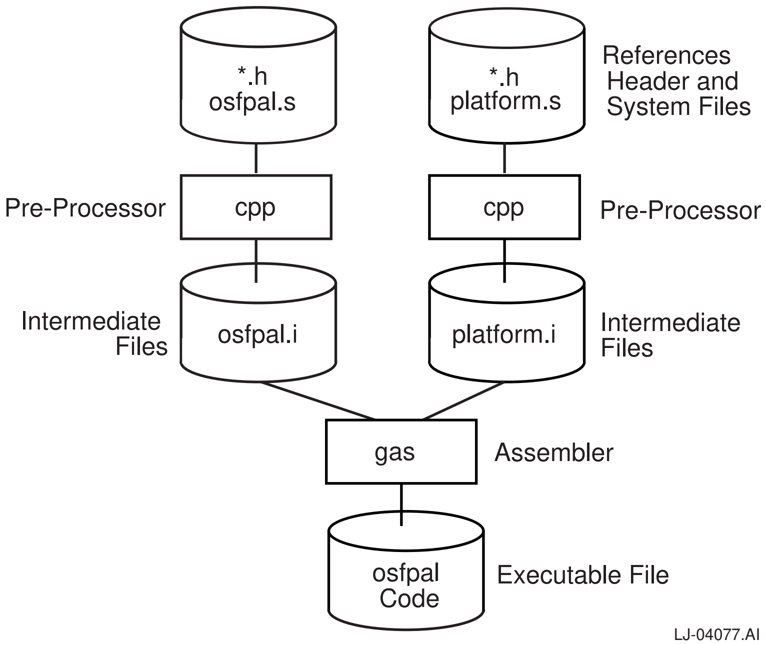
This section describes the process to build osfpal, the PALcode executable image.

The following list describes the tasks that are performed automatically with the make utility or that can be performed individually.

1. Pre-process the osfpal.s (source) file and *.h (header) files to create osfpal.i (intermediate) files.
2. Pre-process the platform.s (source) file and platform.h (header) files to create osfpal.i and platform.i (intermediate) files.
3. Assemble and link osfpal.i with platform.i to create an osfpal executable file.
4. Load and run the osfpal executable image on the target system.

Figure 3–3 shows the osfpal file creation process.

Figure 3–3 PALcode Build Process



After the executable image osfpal has been created, the PALcode must be checked for timing and other coding violations as described in Section 3.5.

3.4.1 PALcode Assembly Rules

Follow these rules when you customize your PALcode routines:

- The PALcode must be assembled in one monolithic assembly from two source modules.
- 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 see *The GNU Assembler* manual for more information about the .align directive.

3.5 Using the PALcode Violation Checker

The **PALcode Violation Checker** (pvc) is a tool used to check PALcode for timing and other coding violations. This tool searches for and identifies PALmode restrictions and violations as described in the microprocessor-specific hardware reference manual. It helps find and correct critical PALcode coding errors before you build the new PALcode image and load it onto your system. Section 3.5.1 briefly summarizes how to use pvc labels and error codes in your customized PALcode routines. See the *Alpha AXP Software Design Tool User's Guide* for more information about pvc features and benefits.

3.5.1 Label Format for pvc

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 they can be used to ignore a specific branch entirely. The labels are also used to ignore a specific pvc error from an instruction in a PALcode routine.

As shown in the following pvc label format, pvc dictates that you start the label with the phrase pvc$. Then you add a label name between the two dollar signs ($) 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.

```
pvc$<your_label_name>$<pvc_error_number>[.destination]:
```

3.5.2 Suppressing pvc Error Messages

The following is an example of a pvc error message with an error code of 82, at address 4940 on an Alpha 21064 microprocessor. This pvc error indicates that you cannot perform an HW_REI instruction during the two cycles immediately following an MT ITBZAP, ITBASM, or ITBIS instruction.

```
Error executing instruction HW_REI at address 4940 on cycle 11!!
(pvc #82) You can't HW_REI during the 2 cycles following a MT ITBZAP,
ITBASM, or ITBIS.
```

For the next example, assume that the system designers have determined that the previous error is harmless for their particular 21064 system implementation, and they want to suppress the error message. This error message can be suppressed 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_4940$82:
```

Note

See the pvc section in the *Alpha AXP Software Design Tool User's Guide* for more details about pvc labels, suppressing error messages, and pvc error codes.

3.6 Customization Decisions

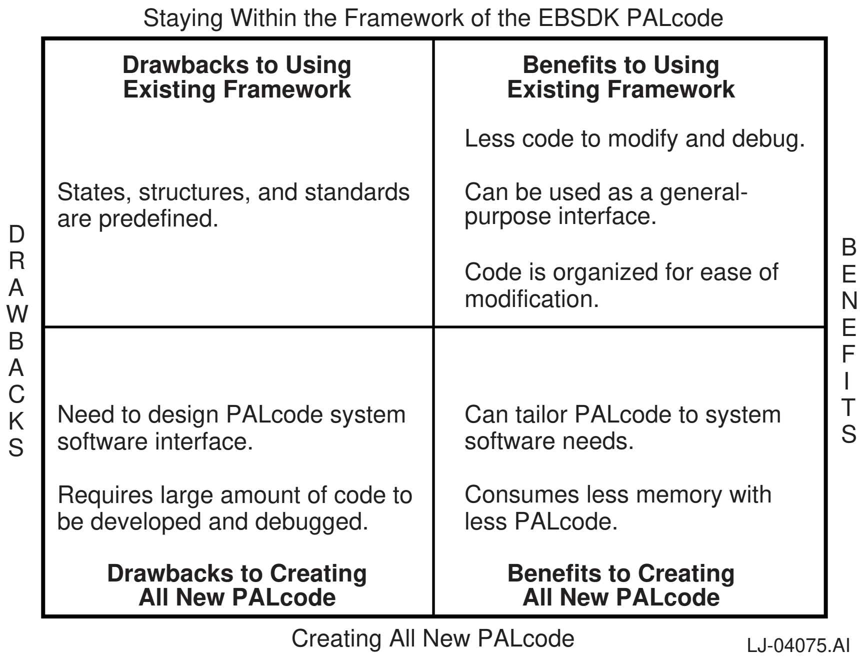
Consider the following issues before you customize PALcode:

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

3.6.1 How Much PALcode Should be Modified?

As system designers, you need to consider many issues before you modify PALcode. Figure 3–4 shows the benefits and drawbacks of using the existing framework of the EBSDK PALcode.

Figure 3–4 Customization Issues



3.7 Design Decision Examples for Alpha Microprocessors

The following list provides examples of some of the modifications that you may need to make for the Alpha microprocessors.

- Provide an alternative backup cache configuration.
- Provide an alternative interrupt or exception stack frame to enhance the information passed to the system software.
- Add privileged and unprivileged CALL_PALs to create a new operating system interface design.
- Provide an alternative memory management policy that has a different mapping or unique page table structure.
- Add new interrupt devices to the current interrupt design.
- Modify the interrupt pin assignments.

3.8 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 2.5 and Section 2.6 for rules and restrictions that may apply.

If you are creating a CALL_PAL, ensure that the entry-point address has been calculated correctly and that the osfpal.ent file has been updated with the new address. Otherwise, the hardware will not dispatch to the correct memory location to execute the function properly. See Section 2.4 and the PALcode entry points section in the microprocessor-specific hardware reference manual.

Table 3–6 describes the steps and tasks you need to perform to modify PALcode, and Figure 3–5 shows the procedure.

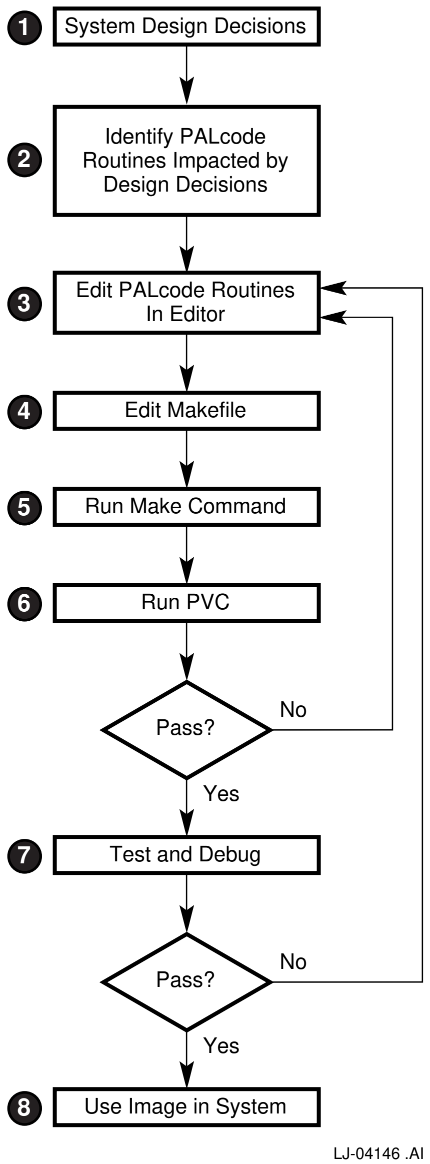
Table 3–6 Modify PALcode Procedure

Step	Task
❶	Determine the needs of your system, such as needing a larger cache or a different interrupt strategy.
❷	Identify the PALcode routines that are impacted by the design decisions.
❸	Edit the PALcode routines that you want to customize by modifying them in an editor of your choice on your host system.
❹	If you are adding or removing files, 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 and links the pre-processed source files using gas and creates one executable image called osfpal. This image is in a.out format. • Post-processes the updated PALcode image with alist and produces the following two files: – 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 the 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, debug your customized PALcode and repeat the steps necessary for this procedure.

Note

Refer to Chapter 4 for information about the steps for building a bootstrap image.

Figure 3–5 Modify PALcode Flow



4

PALcode and the Evaluation Board

This chapter provides information about how PALcode is incorporated into the Evaluation Board. The following topics are included in this chapter:

- Evaluation Board
- PALcode and the Evaluation Board
- Bootstrap Process
- Structure and Contents of the Bootstrap Image
- Relationship Between the Evaluation Board, PALcode, and Your Application
- Features of the EBSDK PALcode
- How PALcode Controls and Analyzes Interrupts
- Memory Management Modes
- Console Service Instructions
- Console Service Descriptions

4.1 Evaluation Board

The Evaluation Board allows you to develop code on a host system and then transfer the software into the Evaluation Board to perform software debugging functions. Software can easily be transferred to the Evaluation Board using either the serial line or the Ethernet port. This software includes embedded control products for communication engines and video products and system software for workstations and personal computers (PCs).

The following software is included with the Evaluation Board:

- **SROM** (Serial ROM) power-up code
- SROM Mini-Debugger
- **Debug Monitor**
- PALcode for the Evaluation Board

See Appendix A for information about technical support and ordering documentation related to Evaluation Boards.

4.1.1 SROM Power-Up Code

The SROM power-up code has the following functions:

- Provides minimal initialization of memory and I/O subsystems
- Runs from instruction cache of CPU
- Runs in PALmode
- Loads and executes the next level of **firmware**

4.1.2 SROM Mini-Debugger

The SROM mini-debugger, which provides troubleshooting assistance, is not part of the boot process or the normal operation of the Evaluation Board's Alpha microprocessor. The SROM mini-debugger has the following functions:

- Provides basic hardware debugging
- Runs entirely from the Icache of the Evaluation Board's CPU
- Does not rely on memory or I/O subsystems to operate
- Communicates through a special SROM RS232 interface with autobaud detection

4.1.3 Debug Monitor

The Debug Monitor provides the following functions:

- Downloads files through serial ports, Ethernet ports, diskettes, and ROM
- Examines and deposits the internal processor registers and I/O mapped registers
- Examines and modifies DRAM and I/O mapped memory
- Disassembles CPU instructions in memory
- Transfers control to programs loaded into memory
- Provides native debugging capabilities, including breakpoints and single stepping
- Provides full source-level debugging capabilities using **DECladebug** running on a remote host that communicates through an Ethernet connection

4.2 PALcode and the Evaluation Board

This section provides information about how PALcode is used with the Evaluation Board and supports the Debug Monitor.

The Evaluation Board determines the following:

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

The Evaluation Board does not affect the platform-independent PALcode routines.

4.2.1 Constants Changed in the EBSDK PALcode

The following constants were modified in the platform.h file to customize PALcode for the Evaluation Board.

- **Bcache** size was set to accommodate the Alpha microprocessor Evaluation Board.
- A mask was set to enable and disable certain interrupts at specific interrupt priority levels.
- Address space partitioning was defined to accommodate the Alpha microprocessor Evaluation Board.

4.2.2 Code Changes in the EBSDK PALcode

The following code was modified in the platform.s file to customize PALcode for the Evaluation Board.

Table 4–1 Code Changes in the platform.s File

What Changed?	Location of Code in platform.s File
An initialization procedure was created for the real-time clock and the interrupt controller on the Evaluation Board	system reset
The causes of interrupts needed to be determined (such as clock or a device), which is required by the OSF/1 operating system.	system interrupt
The current state of the Alpha microprocessor is saved in memory for diagnostic purposes. This state is accessible to the debug monitor *.	system enter console
Many functions were implemented that were required by the debug monitor.	system cserve

*The console uses a physical memory mode, which is described in Section 4.8.2.

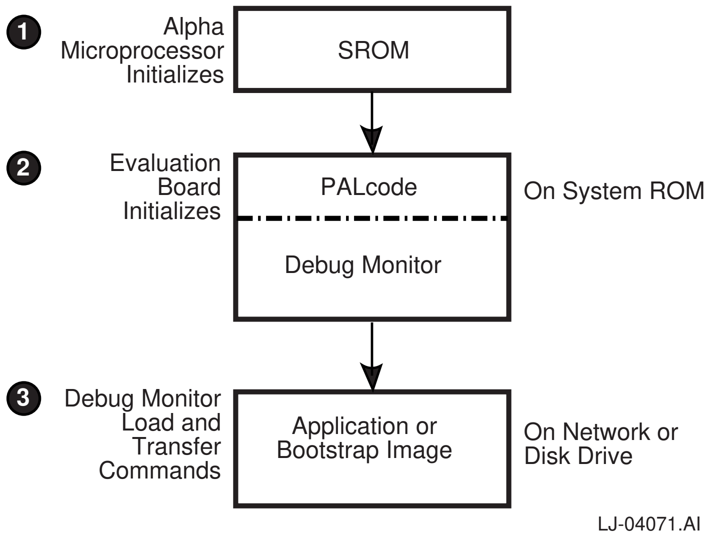
4.3 Bootstrap Process

The following table and figure provide an overview of the events in the bootstrap process. Note that a single PALcode can support both an operating system or control program that is similar to the OSF/1 operating system and the Debug Monitor.

Table 4–2 Events in the Bootstrap Process

Step	Events
1	SROM code loads PALcode and the debug monitor with its associated data structures. Control is then transferred to the PALcode.
2	PALcode performs system initialization functions. Control is then transferred to the Debug Monitor.
3	Debug Monitor commands load and transfer control to an application or bootstrap image.

Figure 4–1 Bootstrap Process



4.4 Structure and Contents of the Bootstrap Image

This section describes how a bootable image was created with the **Sysgen** utility, and it describes the structure and contents of memory during a system boot.

4.4.1 Creating a Bootable Image with Sysgen

Sysgen is an *image-building utility* that concatenates up to ten images into one bootable image. While arranging these images into one contiguous image, the Sysgen utility provides padding in memory (by filling locations with zeros) so that each image is aligned at a page boundary.

The following example shows how to use the Sysgen utility to combine PALcode and an executable image into one bootable image.

```
sysgen -e0 palcode -e300000 your_application > your_image
```

The following list describes how the Sysgen utility combines PALcode and the system software into one bootable image, as shown in Figure 4–2.

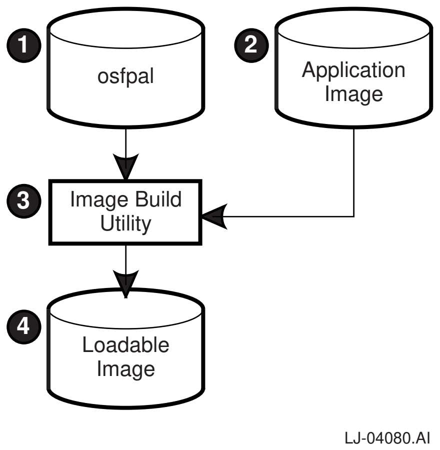
❶ Image file of osfpal.

❷ Image file of system software.

❸ Build utility, which combines PALcode and system software into a bootable image.

❹ Resultant bootable image that is loaded on your system.

Figure 4–2 Building a Bootable Image Process



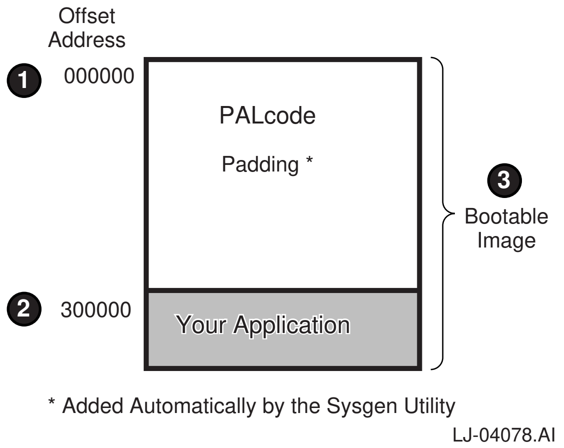
4.4.2 Memory Layout of Bootstrap Image

Table 4–3 defines the creation steps, and Figure 4–3 shows the memory layout of a bootstrap image created with the Sysgen utility.

Table 4–3 Creating a Bootstrap Image with the Sysgen Utility

Step	Sysgen . . .
❶	Places the PALcode image at offset address 0000 and appends padding to it.
❷	Appends your application image at base address 300000.
❸	Combines the three images to produce a single bootable image.

Figure 4–3 Memory Contents of Bootstrap Image



4.5 Relationship Between the Evaluation Board, PALcode, and Your Application

This section describes two methods of incorporating your application onto an Evaluation Board. The first method applies when your image is compatible with the EBSDK PALcode. In this situation, a single PALcode supports the debug monitor and a compatible image. Figure 4–4 shows the relationship between the Evaluation Board software and a compatible application image.

The second method applies when your image is not compatible with the EBSDK PALcode. In this situation, the EBSDK PALcode still supports the Debug Monitor application, but new PALcode that is compatible with your image is also required. Figure 4–5 shows the relationship between the Evaluation Board software and an image that is compatible with the new PALcode.

Figure 4–4 Image Compliant with EBSDK PALcode

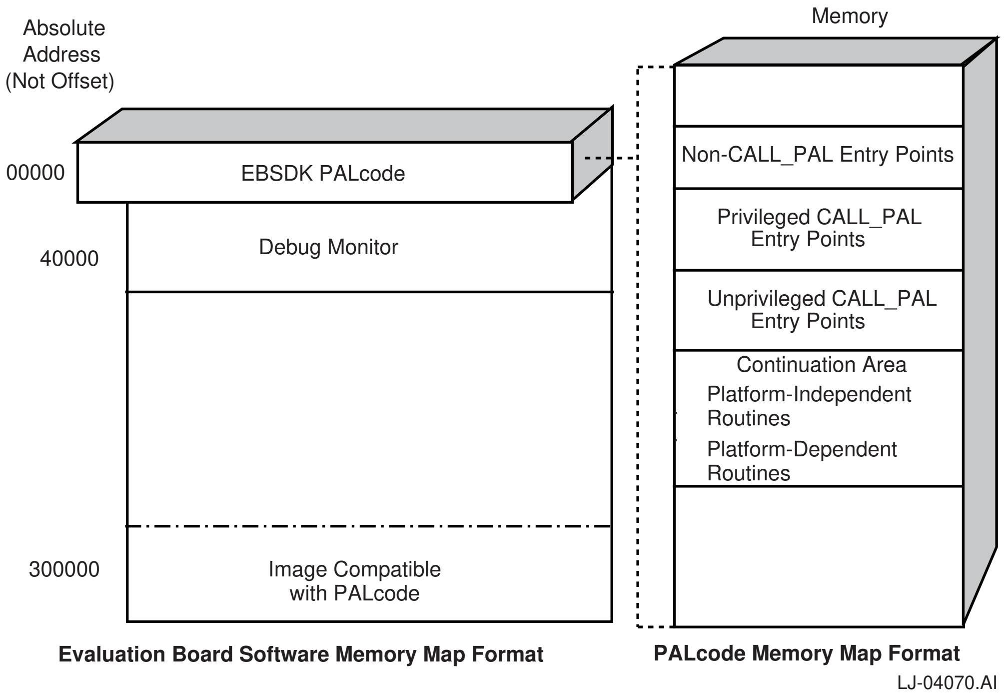
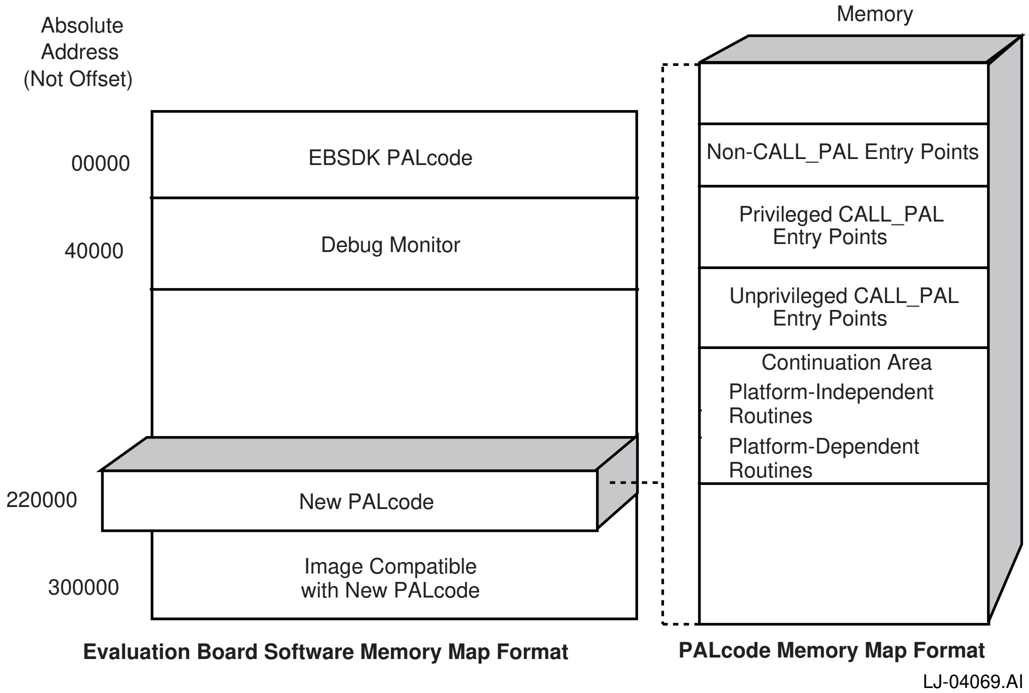


Figure 4–5 Image Requiring New PALcode



See Section 2.4 for information about the PALcode Memory Map format.

4.6 Features of the EBSDK PALcode

The PALcode provided in the EBSDK is an example of the PALcode that you can create using the tools and processes described in this document. This PALcode is fully functional with your Evaluation Board. PALcode provides minimal support for interrupts; it dispatches to the system software to finish processing the interrupt.

This PALcode provides the following features:

- Supports an I/O strategy that is appropriate for the Evaluation Board.
- Provides basic support of interrupts from the timer, system errors, and devices.
- Supports virtual-to-physical address translations if page tables are provided.

- Provides a simple one-to-one, virtual-to-physical address translation without the use of page tables.

4.7 How PALcode Controls and Analyzes Interrupts

All interrupt requests are received as signals on a limited number of pins on the Alpha microprocessor. These interrupt requests can be enabled or masked by the onchip registers of the Alpha microprocessor.

Interrupt requests may originate from I/O devices, memory controllers, and other Alpha microprocessors. Generally, there is a pin for the timer interrupt, one or more pins for system errors, and one or more pins for device interrupts—the exact number of pins varies with the particular Alpha microprocessor device. However, even though there is a limited number of pins for interrupt devices, the total number of interrupt devices can be increased by using interrupt controllers and bridges.

When an interrupt is detected, PALcode is invoked to start processing the interrupt. This PALcode can be adapted to a variety of design strategies for servicing the interrupt.

Note

For more information about interrupts, see the microprocessor-specific Evaluation Board User's Guide.

4.7.1 Adapting PALcode to Service Interrupts

When modifying PALcode for interrupts, you need to consider the I/O strategy and how PALcode should respond when servicing the interrupts. The following list summarizes some of the design considerations for determining how PALcode should respond when servicing interrupts:

- What is the interrupt structure for the platform?
- How much control and analysis of the interrupt is required by PALcode before transferring control to the system software?

4.7.2 Processing Interrupts

When an interrupt is detected, the Alpha microprocessor enters PALmode to service the device. Depending on the interrupt strategy, PALcode may perform some analysis of the interrupt and access PALmode-visible registers to save state information. PALcode then dispatches to the system software, which then processes the interrupt. When the system software has completed processing the interrupt, the system software executes a return-from-interrupt CALL_PAL. PALcode processes the return-from-interrupt CALL_PAL and restores the program counter to the code that was executing at the time of the interrupt.

4.7.3 How the EBSDK PALcode Processes Interrupts

The EBSDK PALcode distinguishes between three types of external interrupts: timer, system error, and device. After the type of interrupt is identified, the EBSDK PALcode performs some minimal processing of the interrupt; it dispatches to the system software to analyze and control the servicing of the request.

How PALcode Services a Timer Interrupt

Upon detecting a timer interrupt, the EBSDK PALcode:

1. Saves state on the stack.
2. Indicates the type of interrupt entry with values in a0 . . . a2, and sets the interrupt priority level (IPL). Setting the IPL to the level of the interrupt provides onchip masking for lower priority sources.
3. Clears the interrupt in the device's control register.
4. Dispatches to the system software and indicates a timer interrupt in a0.

How PALcode Services a System Error Interrupt

Upon detecting a system error interrupt, the EBSDK PALcode:

1. Saves state on the stack.
2. Indicates the type of interrupt entry with values in a0 . . . a2, and sets the interrupt priority level (IPL). Setting the IPL to the level of the interrupt provides onchip masking for lower priority sources.
3. Saves error logging information in logout frame.
4. Performs some minimal clearing of the condition.
5. Dispatches to the system software to complete the servicing of the machine check routine.

How PALcode Services a Device Interrupt

Upon detecting a device interrupt, the EBSDK PALcode:

1. Saves state on the stack.
2. Indicates the type of interrupt entry or machine check with values in a0 . . . a2, and sets the interrupt priority level (IPL). Setting the IPL to the level of the interrupt provides onchip masking for lower priority sources.
3. Dispatches to the system software to complete the servicing of the interrupt.

4.8 Memory Management Modes

Memory management provides a mechanism to map the active part of the virtual address space to the available physical address space. The system software controls the virtual-to-physical mapping tables (also called page tables) and saves the inactive parts of the virtual address space on external storage media. PALcode uses these page tables to perform the virtual-to-physical address translations and to manage the translation buffers. In a true virtual memory environment, the Page Table Base Register (PTBR) would contain the physical page frame number (PFN) of the highest level page table.

4.8.1 Virtual Memory Mapping

The EBSDK PALcode supports virtual-to-physical address translation as described in the OSF/1 memory management section of the *Alpha Architecture Reference Manual*.

4.8.2 Physical Memory Mapping

In the absence of a virtual-memory environment, the EBSDK PALcode provides an additional feature to allow a simple one-to-one, virtual-to-physical address translation without the use of page tables. This mode, which is referred to as *physical mode* in the EBSDK PALcode, can be enabled by three methods:

- In the enter console routine prior to entering the next level of software, by setting the least significant bits in the PAL_TEMP registers that contain the PTBR and Virtual Page Table Base Registers (VPTBR) values. The system remains in physical mode until a CALL_PAL explicitly changes it to virtual memory mapping.
- In a swap PALcode (swppal) CALL_PAL function setting bit<63> to a 1 of the PTBR field in the new Process Control Block (PCB). The system remains in physical mode until another CALL_PAL explicitly changes it to virtual memory mapping.

- In a swap process context (swpctx) CALL_PAL function by setting bit<63> to a 1 of the PTBR field in the new Process Control Block (PCB). The swpctx CALL_PAL function changes to physical mode on a per-process basis.

Note

See the sections about privileged OSF/1 PALcode instructions and OSF/1 process structure in the *Alpha Architecture Reference Manual*.

The EBSDK PALcode uses the following algorithm to generate a page table entry (PTE) that maps a one-to-one, virtual-to-physical address translation:

```
PTE <63:32> <- left_shift (VA, {32 - lg(PageSize)}) ! Fabricate PFN
PTE<13> <- 1 ! Enable writes from user mode
PTE<12> <- 1 ! Enable writes from kernel mode
PTE<9> <- 1 ! Enable reads from user mode
PTE<8> <- 1 ! Enable reads from kernel mode
PTE<6:5> <- 3 ! Treat a block of 512 pages as a single larger page
PTE<4> <- 1 ! Make this PTE match all address space numbers
PTE<0> <- 1 ! Set PTE valid
```

The EBSDK PALcode writes the PTE and tag, corresponding to the translation of the specified virtual address, into the appropriate translation buffers using the internal processor registers.

4.9 Console Service Function Overview

This section provides a summary of the console service functions included with the EBSDK PALcode. Some console service functions (Section 4.9.1) apply to all Alpha microprocessors. Others apply only to specific Alpha microprocessors (Sections 4.9.2 through 4.9.4).

4.9.1 Console Service Functions for All Alpha Microprocessors

The following table lists the console service functions that apply to all Alpha microprocessors.

Description	Mnemonic	See Section . . .
Jump to PALcode	jtopal	4.10.1
Load quadword physical	ldqp	4.10.2
Output a character to the serial port	putc	4.10.3
Read impure pointer	rd_impure	4.10.11
Store quadword physical	stqp	4.10.12
Write interrupt mask register	wr_int	4.10.20

In addition to these console service functions, there are microprocessor-specific console service functions. See Sections 4.9.2, 4.9.3, and 4.9.4.

4.9.2 Console Service Functions for Alpha 21064 and 21064A Microprocessors

The following table lists the console service functions that are specific to the Alpha 21064 and 21064A microprocessors.

Description	Mnemonic	See Section . . .
Read ABOX_CTL internal processor register	rd_abox	4.10.4
Read BIU_CTL internal processor register	rd_biu	4.10.7
Read ICCSR internal processor register	rd_iccsr	4.10.9
Write ABOX_CTL internal processor register	wr_abox	4.10.13
Write BIU_CTL internal processor register	wr_biu	4.10.16
Write ICCSR internal processor register	wr_iccsr	4.10.18

4.9.3 Console Service Functions for Alpha 21066, 21066A and 21068 Microprocessors

The following table lists the console service functions that are specific to the Alpha 21066, 21066A, and 21068 microprocessors.

Description	Mnemonic	See Section . . .
Read ABOX_CTL internal processor register	rd_abox	4.10.4
Read ESR internal processor register	rd_esr	4.10.8
Read ICCSR internal processor register	rd_iccsr	4.10.9
Write ABOX_CTL internal processor register	wr_abox	4.10.13
Write ESR internal processor register	wr_esr	4.10.17
Write ICCSR internal processor register	wr_iccsr	4.10.18

4.9.4 Console Service Functions for the Alpha 21164 Microprocessor

The following table lists the console service functions that are specific to the Alpha 21164 microprocessor.

Description	Mnemonic	See Section . . .
Read BC_CONFIG internal processor register	rd_bcCfg	4.10.5
Read BC_CONTROL internal processor register	rd_bcCtl	4.10.6
Read ICSR internal processor register	rd_icsr	4.10.10
Write BC_CONFIG internal processor register	wr_bcCfg	4.10.14
Write BC_CONTROL internal processor register	wr_bcCtl	4.10.15
Write ICSR internal processor register	wr_icsr	4.10.19

4.10 Console Service Function Descriptions

This section provides descriptions of the console service functions for Alpha microprocessors.

4.10.1 Jump to PALcode

Alpha Microprocessors:

All

Format:

jtopal !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
      {Initiate OPCDEC fault}
endif
tmp <- a0 AND {NOT 3}
PC  <- tmp OR 1
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

jtopal	Transfer control in PALmode

Qualifiers:

None

Description:

The PC is loaded with the target address. The new PC is supplied from register a0. The least significant bit of a0 is set to indicate transfer of control in PALmode. Registers t0..t4, t8..t11, s6, and a0..a5 are UNPREDICTABLE.

4.10.2 Load Quadword Physical

Alpha Microprocessors:

All

Format:

ldqp !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
v0 <- (a0)<63:0>
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

ldqp	Load Quadword Physical from Memory to Register

Qualifiers:

None

Description:

The physical address is passed in register a0 and the source operand is fetched from memory and written to register v0.

4.10.3 Output a Character to the Serial Port

Alpha Microprocessors:

All

Format:

putc !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
        {Initiate OPCDEC fault}
endif
if {LSR<THRE> EQ 1} then
        THR<7:0> <- a0
        v0 <- 1
else
        v0 <- 0
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

putc Output a Character to the Serial Port

Qualifiers:

None

Description:

Outputs an 8-bit character passed in register a0 to the serial port. Returns 1 in v0 if the character was successfully transmitted; otherwise, returns zero.

4.10.4 Read ABOX_CTL Internal Processor Register

Alpha Microprocessors:

21064, 21064A, 21066, 21066A, 21068

Format:

rd_abox !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
tmp <- ptImpure
v0 <- (tmp + CNS_Q_BASE + CNS_Q_ABOX_CTL)
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rd_abox	Read ABOX Control Register

Qualifiers:

None

Description:

The read ABOX_CTL internal processor register function returns the shadow copy of the ABOX control register, fetched from the PALcode impure scratch area, in register v0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.5 Read BC_CONFIG Internal Processor Register

Alpha Microprocessors:

21164

Format:

rd_bcCfg !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
tmp <- ptImpure
v0 <- (tmp + CNS_Q_BC_CFG)
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rd_bccfg	Read BC_CONFIG register

Qualifiers:

None

Description:

The read Bcache configuration internal processor register function returns the shadow copy of the BC_CONFIG control register, fetched from PALcode **impure area**, in register v0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.6 Read BC_CONTROL Internal Processor Register

Alpha Microprocessors:

21164

Format:

rd_bcCtl !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
tmp <- ptImpure
v0 <- tmp + CNS_Q_BC_CTL
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rd_bcctl	Read BC_CONTROL register

Qualifiers:

None

Description:

The read Bcache control internal processor register function returns the shadow copy of the BC_CONTROL register, fetched from PALcode impure area, in register v0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.7 Read BIU_CTL Internal Processor Register

Alpha Microprocessors:

21064, 21064A

Format:

rd_biu !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
tmp <- ptImpure
v0 <- (tmp + CNS_Q_BASE + CNS_Q_BIU_CTL)
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rd_biu	Read BIU_CTL register

Qualifiers:

None

Description:

The read bus interface unit control internal processor register function returns the shadow copy of the BIU_CTL control register, fetched from PALcode impure area, in register v0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.8 Read ESR Internal Processor Register

Alpha Microprocessors:

21066, 21066A, 21068

Format:

rd_esr !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
v0 <- ESR<63:0>
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rd_esr	Read Error Status register

Qualifiers:

None

Description:

The read error status internal processor register function returns the value of the memory controller error status register, addressed in physical memory, in register v0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.9 Read ICCSR Internal Processor Register

Alpha Microprocessors:

21064, 21064A, 21066, 21066A, 21068

Format:

rd_iccsr !PALcode format

Operation:

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

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rd_iccsr Read instruction cache control and status register

Qualifiers:

None

Description:

The read instruction cache control and status register function returns the shadow copy of the instruction cache control and status register in register v0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.10 Read ICSR Internal Processor Register

Alpha Microprocessors:

21164

Format:

rd_icsr !PALcode format

Operation:

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

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rd_icsr Read ICSR register

Qualifiers:

None

Description:

The read Ibox control and status internal processor register function returns the value of ICSR in register v0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.11 Read Impure Pointer

Alpha Microprocessors:

All

Format:

rd_impure !PALcode format

Operation:

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

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

rd_impure	Read Impure Pointer

Qualifiers:

None

Description:

The read impure pointer function returns the base address of the PALcode impure scratch area in v0. On return from the rd_impure function, registers t0 and t8..t11 are UNPREDICTABLE.

4.10.12 Store Quadword Physical

Alpha Microprocessors:

All

Format:

stqp !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
(a0) <- a1
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

stqp Store Quadword Physical from Register to Memory

Qualifiers:

None

Description:

The physical address is passed in register a0 and the a1 operand is written to memory at this address.

4.10.13 Write ABOX_CTL Internal Processor Register

Alpha Microprocessors:

21064, 21064A, 21066, 21066A, 21068

Format:

wr_abox !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
tmp <- ptImpure
tmp <- tmp + CNS_Q_BASE + CNS_Q_ABOX_CTL
(tmp) <- a0
aboxCtl <- a0
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wr_abox Write ABOX Control Register

Qualifiers:

None

Description:

The write ABOX_CTL internal processor register function writes both the ABOX control internal processor register and the shadow copy of the ABOX control register, stored in the PALcode impure scratch area, with the value passed in a0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.14 Write BC_CONFIG Internal Processor Register

Alpha Microprocessors:

21164

Format:

wr_bcCfg !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
tmp <- ptImpure
tmp <- tmp + + CNS_Q_BC_CFG
(tmp) <- a0
bcCfg <- a0
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wr_bccfg	Write BC_CONFIG Register

Qualifiers:

None

Description:

The write Bcache configuration internal processor register function writes both the BC_CONFIG internal processor register, addressed in physical memory, and the shadow copy of the BC_CONFIG register, stored in the PALcode impure area, with the value passed in a0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.15 Write BC_CONTROL Internal Processor Register

Alpha Microprocessors:

21164

Format:

wr_bcCtl !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
tmp <- ptImpure
tmp <- tmp + + CNS_Q_BC_CTL
(tmp) <- a0
bcCtl <- a0
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wr_bcctl	Write BC_CONTROL Register

Qualifiers:

None

Description:

The write Bcache control internal processor register function writes both the BC_CONTROL register, addressed in physical memory, and the shadow copy of the BC_CONTROL register, stored in the PALcode impure area, with the value passed in a0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.16 Write BIU_CTL Internal Processor Register

Alpha Microprocessors:

21064, 21064A

Format:

wr_biu !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
tmp <- ptImpure
tmp <- tmp + CNS_Q_BASE + CNS_Q_BIU_CTL
(tmp) <- a0
biuCtl <- a0
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wr_biu	Write BIU Control Register

Qualifiers:

None

Description:

The write bus interface unit control internal processor register function writes both the BIU control internal processor register and the shadow copy of the BIU control register, stored in the PALcode impure area, with the value passed in a0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.17 Write ESR Internal Processor Register

Alpha Microprocessors:

21066, 21066A, 21068

Format:

wr_esr !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
ESR<63:0> <- a0
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wr_esr	Write Error Status Register

Qualifiers:

None

Description:

The write error status internal processor register function writes the value passed in a0 to the memory controller error status register, addressed in physical memory. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.18 Write ICCSR Internal Processor Register

Alpha Microprocessors:

21064, 21064A, 21066, 21066A, 21068

Format:

wr_iccsr !PALcode format

Operation:

```
if (PS<mode> EQ 1) then
       {Initiate OPCDEC fault}
endif
ptIccsr <- a0
iccsr <- a0
```

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wr_iccsr Write Instruction Cache Control and Status Register

Qualifiers:

None

Description:

The write ICCSR internal processor register function writes both the instruction cache control and status register and its PALtemp shadow copy with the value passed in register a0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.19 Write ICSR Internal Processor Register

Alpha Microprocessors:

21164

Format:

wr_icsr !PALcode format

Operation:

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

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wr_icsr	Write ICSR Register

Qualifiers:

None

Description:

The write Ibox control and status internal processor register function writes the ICSR with the value passed in a0. Registers t0 and t8..t11 are UNPREDICTABLE.

4.10.20 Write Interrupt Mask Register

Alpha Microprocessors:

All

Format:

wr_int !PALcode format

Operation:

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

Exceptions:

Opcode reserved to Digital

Instruction Mnemonics:

wr_int	Write Interrupt Mask Register

Qualifiers:

None

Description:

The write interrupt mask function writes the value passed in a0 to the interrupt mask register. Registers t0, and t8..t11 are UNPREDICTABLE. On the Alpha 21064, 21064A, 21066, 21066A, and 21068 microprocessors, this mask provides mapping between the OSF/1 interrupt priority level and interrupts to be enabled or disabled through the HIER (hardware interrupt enable register). On the Alpha 21164 microprocessor, this mask provides a translation between the OSF/1 interrupt priority level and the hardware interrupt priority level. The hardware interrupt priority level determines which interrupts are enabled or disabled.

A

Technical Support and Ordering Information

This appendix describes how to obtain information and technical support, and order products and associated literature.

Technical Support

If you need technical support or help deciding which literature best meets your needs, call the Digital Semiconductor Information Line:

United States and Canada	**1–800–332–2717**
TTY (United States only)	**1–800–332–2515**
Outside North America	**+1–508–568–6868**

Ordering Digital Semiconductor Products

To order Alpha microprocessors and evaluation boards, contact your local distributor.

You can order the following semiconductor products from Digital:

Microprocessor and Peripheral Chips

Product	Order Number
DECchip 21030 PCI Graphics Accelerator	21030–AA
DECchip 21040 Ethernet LAN Controller for PCI	21040–AA
DECchip 21050 PCI-to-PCI Bridge	21050–AA
Alpha 21064-150 Microprocessor	21064–AA
Alpha 21064-166 Microprocessor	21064–CA
Alpha 21064-200 Microprocessor	21064–BA
Alpha 21064A-233 Microprocessor	21064–BB
Alpha 21064A-275 Microprocessor	21064–DB
Alpha 21066-166 Microprocessor	21066–AA

Product	Order Number
Alpha 21068-66 Microprocessor	21068–AA
Alpha 21066A-100 Microprocessor	21066–CB
Alpha 21066A-233 Microprocessor	21066–AB
Alpha 21164-266 Microprocessor	21164–AA
Alpha 21164-300 Microprocessor	21164–BA
DECchip 21071 Core Logic Chipset	21071–AA
DECchip 21072 Core Logic Chipset	21072–AA

Evaluation Board Kits

Product	Order Number
Alpha 21064 Evaluation Board Kit –150 MHz	21A01–03
Alpha 21064 PCI Evaluation Board Kit –200 MHz	21A02–01
Alpha 21064A PCI Evaluation Board Kit –275 MHz	21A02–02
Alpha 21066 and Alpha 21068 Evaluation Board Kit –166 MHz	21A03–01
Alpha 21066A Evaluation Board Kit –233 MHz	21A03–02
Alpha 21164 Evaluation Board Kit –266 MHz	21A04–01
AlphaPC64 Evaluation Board Kit –275 MHz	21A02–03

Ordering Associated Literature

The following table lists some of the available Digital Semiconductor literature. For a complete list, contact the Digital Semiconductor Information Line.

Title	Order Number
Alpha Architecture Reference Manual[1]	EY–L520E–DP–YCH
Alpha Microprocessors Evaluation Board Debug Monitor User's Guide	EC–QHUVA–TE
AlphaPC64 Evaluation Board Product Brief	EC–QEZMB–TE
AlphaPC64 Evaluation Board User's Guide	EC–QGY2A–TE
Alpha AXP Evaluation Board Software Design Tools User's Guide	EC–N0662–72

[1]To order and purchase the *Alpha Architecture Reference Manual*, call **1–800–DIGITAL** from the U.S. or Canada, or contact your local Digital office, or technical or reference bookstore where Digital Press books are distributed by Prentice Hall.

Title	Order Number
Alpha 21064 and Alpha 21064A Microprocessors Hardware Reference Manual	EC–Q9ZUA–TE
Alpha 21066, 21066A, and 21068 Microprocessors Hardware Reference Manual	EC–QC4GA–TE
Alpha 21066A Microprocessor Evaluation Board (EB66+) User's Guide	EC–QDVCA–TE
Alpha 21066A Microprocessor Evaluation Board (EB66+) Product Brief	EC–QDVEA–TE
Alpha 21164 Microprocessor Hardware Reference Manual	EC–QAEQB–TE
Alpha 21066/21066A Microprocessors Data Sheet	EC–QC4HA–TE
Answers to Common Questions about PALcode for Alpha AXP Systems	EC–N0647–72
DECchip 21064 and DECchip 21064A Alpha AXP PCI Evaluation Board User's Guide	EC–N0640–72
DECchip 21066 and DECchip 21068 Alpha AXP Evaluation Board User's Guide	EC–N0626–72

Glossary

Alpha microprocessor

In this document, the term Alpha microprocessor refers to the Alpha 21064, 21064A, 21066, 21066A, 21068, and 21164 microprocessors, unless noted otherwise.

atomic

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

Bcache

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

CALL_PAL

Call Privileged Architecture Library. An Alpha 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 hardware of the Alpha microprocessor directly supports up to 64 privileged and unprivileged CALL_PALs with dispatches to specific address offsets.

control and status registers

Include the internal processor registers for all Alpha microprocessors, and the memory controller registers plus the control registers that are specific to certain Alpha microprocessors.

cpp

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

debug monitor

A program that system designers use on the Evaluation Board to enter commands for debugging and processing their customized PALcode routines and other applications they have written for Alpha microprocessors.

DECladebug

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

EB64+

21064 PCI Evaluation Board. An Evaluation Board that serves as a target to load, examine, and debug an operating system or embedded application.

EB66

21066 and 21068 Evaluation Board. An Evaluation Board that serves as a target to load, examine, and debug an operating system or embedded application.

EB66+

21066A Evaluation Board. An Evaluation Board that serves as a target to load, examine, and debug an operating system or embedded application.

EBSDK

Evaluation Board Software Developer's Kit. A kit that contains PALcode for Evaluation Boards and tools for creating PALcode.

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.

Evaluation Board

A development module that allows a system designer to load, examine, and debug an operating system or embedded application on an Alpha microprocessor. In this document, the term Evaluation Board refers to the 21064 Evaluation Board, the 21064 PCI Evaluation Board, the 21066 and 21068 Evaluation Board, the 21066A Evaluation Board, the 21164 Evaluation Board, and the AlphaPC64 Evaluation Board unless noted otherwise.

Evaluation Board Software Developer's Kit.

See EBSDK.

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.

image

Any executable file. Examples include operating systems, O/S loaders, programs, and embedded control programs.

impure area

A common data area maintained by PALcode, which may be accessed by the operating system and the debug monitor software. The impure area shadows the write-only internal processor registers, which contain internal state and MCHK logout information that is saved during a system halt.

internal processor register (IPR)

See IPR.

IPR

Internal processor register. Registers that indicate status and control the processor state, which are only accessible through PALcode.

Istream

Instruction stream. Instructions that are executing.

machine check

See MCHK.

Makefile

Command file that controls the building of a large program from multiple source files.

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.

native-mode

All processes or programs that do not run in PALmode.

PAL

Privileged Architecture Library.

PALcode

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

PALcode Violation Checker

See pvc.

PALmode

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

PAL_TEMP

A set of 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 Alpha microprocessor-based design without modifications.

platform-dependent (OSF/1 PALcode)

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

privileged

Functions that can be accessed only from kernel mode.

Privileged Architecture Library code (PALcode)

See PALcode.

PTBR

The page table base register. The PTBR contains the physical page frame number (PFN) of the highest level (level 1) page table.

pvc

PALcode Violation Checker. A tool, used in the build procedure, 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 Alpha microprocessor and Evaluation Board system.

sysgen

A sample utility (used on the Evaluation Board 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

R

S

T

U

V