Realtime Products

digital

Technical Summary
Fifth Edition

Part Number: EK–RPTSS–TM-004

Realtime Products Technical Summary

Order Number: EK–RPTSS–TM–004

Fifth Edition

This summary describes Digital's key realtime software and hardware offerings.

Revision/Update Information: This manual supersedes the *Realtime Products Technical Summary*, EK–RPTSS–TM–003.

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, August 1990
Revised, May 1991, December 1991, January 1992, December 1992

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

Trademarks of Digital Equipment Corporation and third-party trademarks are listed in Appendix B.

The cover photo for this manual appears courtesy of the National Aeronautics and Space Administration, Houston, Texas.

S1962

This document is available on CD–ROM.

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

25 Realtime Options

Tables

Preface

The *Realtime Products Technical Summary* surveys Digital's key realtime software and hardware products. Here you will find discussions of each product's features, development and runtime environments, and typical uses.

This summary is written for computer professionals who are investigating realtime solutions. Familiarity with Digital's portfolio of realtime offerings is not a prerequisite, but readers should be acquainted with basic realtime concepts. If you are a Digital system user with an interest in Digital's products for developing realtime solutions, this summary provides insight into Digital's product capabilities and uses.

The *Realtime Products Technical Summary* is organized in ten parts and contains two appendixes and a glossary. Each part discusses a different facet of Digital's realtime products. Appendix A provides pointers to additional documentation arranged by topic. The book parts are marked by dividers to help you locate specific areas of interest.

To further assist you in navigating the book, Chapter 1 discusses the term *realtime*, identifies realtime system characteristics, and gives an overview of Digital's commitment to standards and Open Systems. Also outlined here are Digital's key realtime offerings (which are described within the book) and some Digital-sponsored programs for promoting realtime solutions.

The fifth edition includes these enhancements and additions:

- The order in which products are presented was changed to emphasize recently introduced realtime products.
- For easy online navigation, the introductory material previously presented in the Preface is now in Chapter 1.
- Part I, DECelx, was updated to include functionality added for DECelx Version 2.0.
- Part II, VAXELN Toolkit, was updated to include functionality added for VAXELN Version 4.4 and to include VAXELN Window Server Version 1.2 functionality and the new VAXELN Window Station.

- Part IV, DEC OSF/1 Systems in Realtime Applications, was updated to reflect new POSIX 1003.4 Draft 11 functionality for DEC OSF/1 Version 1.2.
- Part V, High-Level Language Support for Realtime Applications, was revised to cover a wider range of languages now available on UNIX platforms.
- Part VI, The Digital COHESION Environment, was revised to reflect recent releases of VMS and UNIX CASE tools. Other layered products that contribute to Digital's realtime environment, such as DECmessageQ and the DECfactory products, were added to the section.
- Part VII, DEC Realtime Integrator, was updated to include functionality added for DEC Realtime Integrator Version 3.0 and the new product, DEC Realtime Integrator for VAXELN.
- Part VIII, Realtime Hardware, was revised to include new and updated hardware offerings for VAX and MIPS-based platforms. Information on older hardware offerings was eliminated.
- Earlier versions of this manual included an appendix listing the software requirements for products covered in the technical summary. General software requirements are now included in the chapters covering those products. For specific software requirements, refer to the *System Support Addendum (SSA)* for each product.

Digital's UNIX products include the ULTRIX and DEC OSF/1 operating systems running on multiple hardware platforms. Not all products discussed in this manual run on all of Digital's UNIX platforms. The following definitions will help you understand the context of the discussions.

- **UNIX** applies to all Digital UNIX products, including ULTRIX and DEC OSF/1.
- **ULTRIX** applies to Digital products running on either a VAX ULTRIX or MIPS-based ULTRIX system.
- **DEC OSF/1** applies to Digital products running on MIPS-based and Alpha hardware.
- **DECelx** applies to Digital products running on a DECelx system.

1 Introduction to Digital's Realtime Products

Digital offers a variety of products for developing realtime solutions that run on the PDP–11, VAX, MIPS-based, and Alpha processors. Most realtime solutions come from third-party vendors that specialize in areas such as process control systems, aircraft simulators, robotics, and medical diagnostics equipment. Digital's products help these vendors produce the best solutions in the market.

Not only does Digital furnish a comprehensive set of products for developing realtime solutions, it allows for connectivity as well. With Digital's interconnect products and networking software, companies can seamlessly integrate solutions throughout the corporate network. Using Digital's realtime products, you can bring distributed computing to the factory floor or laboratory. You can distribute computing power where it is needed. As application requirements change, you can easily add computing power or migrate the application to a more appropriate configuration.

This introduction covers the following topics:

- Realtime Applications, Section 1.1
- Realtime System Characteristics, Section 1.2
- Industry Standards, Section 1.3
- Open Systems Support, Section 1.4
- POSIX Standards, Section 1.5
- Realtime Products Summary, Section 1.6
- Digital-Sponsored Programs for Promoting Realtime Solutions, Section 1.7

Although this technical summary focuses on Digital's 32-bit realtime products, thousands of realtime solutions are running on Digital's 16-bit PDP–11 processors. Running RT–11, RSX–11, or MicroPower/Pascal, the PDP–11 processors and their boards and options have answered realtime computing needs for 20 years and will continue to do so.

The speed, responsiveness, and predictability of VAX and MIPS-based processors suit them to realtime application requirements. The Digital family supplies a wide range of hardware and software options for building realtime systems. You can choose among processor implementations, bus architectures, realtime device options, mass storage devices, and integrated software environments to configure a system that meets the requirements of your realtime application.

1.1 Realtime Applications

Realtime applications are equipment oriented. Rather than furnish interfaces for people, as desktop applications do, realtime applications supply interfaces for equipment, such as input/output (I/O) devices and busses. Realtime applications run on computers connected to equipment that collects and, in some cases, controls the processing of data. Examples of realtime, data-collecting applications include long-distance telephone testing systems, systems that receive data from satellites, and systems that use laboratory measuring equipment. Automobile automatic braking systems, process control systems in power-generating plants, and airplane automatic pilot systems are examples of realtime control applications.

Realtime applications must respond to events generated by equipment within a predetermined time limit. Typical realtime applications include data acquisition and analysis, simulations, process control, computer-integrated manufacturing (CIM), image processing, built-in test equipment, networked I/O servers, communication switching systems, and dedicated professional workstations.

The term *realtime* does not necessarily imply high speed. An environmental monitoring system is an example of a realtime application that might require that readings of wind speed, wind direction, and environmental pollutant levels be taken at intervals ranging from one second to several minutes. Although this application does not require high speed, it must obtain each reading at the specified interval. If a reading is missed, you cannot recover the lost data.

Many realtime applications require high I/O throughput. I/O throughput, the rate at which a computer system handles incoming and outgoing data, is dependent upon the speed and features of the central processing unit (CPU) and I/O device as well as the bandwidth of the I/O bus. Realtime applications that require high I/O throughput rely on continuous processing of large amounts of data. The primary requirement of such applications is to acquire a number of data points equally spaced in time. The collected data can then be stored or used in computations.

High I/O throughput requirements are typically found in signal-processing applications such as sonar and radar analysis, telemetry, vibration analysis, speech analysis, and music synthesis. Likewise, a continuous stream of data points must be acquired for many of the qualitative and quantitative methods used in applications including gas and liquid chromatography, mass spectrometry, automatic titration, and colorimetry. For some of these applications, the throughput requirements on any one channel are relatively modest. However, a system may need to handle multiple channels of data simultaneously, resulting in a high aggregate throughput requirement.

Some realtime applications require both high throughput and fast response to asynchronous external events. High I/O throughput and fast response are the key metrics for data acquisition systems, for which data collection is triggered by an external event.

The key metrics for realtime control systems include the speed at which the system responds to asynchronous external events (device interrupts) and the system's ability to schedule and provide communication between multiple tasks. High I/O throughput may be an important factor for realtime control systems as well. Realtime control systems (such as simulators) must capture input parameters, perform decision-making operations, and compute updated output parameters within a given time frame.

Consider a flight simulator application. The application might acquire several hundred input parameters from the cockpit controls; compute updated position, orientation, and speed parameters; and then send several hundred output parameters to the cockpit console and a visual display subsystem. Typically, these operations must be performed within 3 milliseconds. Other applications for which response time is critical include process monitoring and control, synchronous communications, and stimulus-response testing in biological and psychological research.

Predictability is another characteristic of realtime systems. Realtime systems should respond predictably to realtime tasks. Thus, realtime systems reduce system services and activities, such as paging and swapping, to minimize overhead and maximize predictability.

The following types of applications can incorporate realtime hardware and software; many of these are cited as examples throughout this technical summary:

- Acoustics
- Aerospace research and testing
- Analytical instrumentation

- Component evaluation
- Electronic and equipment testing
- Function generators
- Graphics displays and imaging
- Interprocessor communications
- Materials testing
- Medical research
- Physiological monitoring
- Product handling
- Programmable power supplies
- Seismic data collection
- Spectrum analysis
- Structural analysis and performance testing

1.2 Realtime System Characteristics

In addition to offering high throughput, fast response to asynchronous external events, and predictable responses to realtime tasks, realtime systems can also exhibit one or more of the following characteristics:

- **Dedicated resources.** A dedicated realtime system uses one or more computers to solve a specific problem or set of related problems. A dedicated system devotes all system resources to the realtime application; it does not provide time-sharing services. Dedicated realtime applications are somtimes referred to as hard realtime applications.
- **Distributed processing.** A distributed realtime system uses computing resources efficiently by distributing less time-critical functions to other processors. You can distribute applications, application components, and application resources over a network or between processors in multiprocessor configurations. This allows time-critical tasks to use realtime target processors more efficiently while maintaining communication with all application functions.
- **High I/O througput.** I/O throughput is the rate at which a computer system handles incoming and outgoing data. I/O throughput is dependent upon the speed and features of the central processing unit (CPU), the I/O device, and the bandwidth of the I/O bus. Realtime applications that

require high I/O throughput rely on continuous processing of large amounts of data. The primary requirements of such application is to acquire a number of data points equally spaced in time. The collected data might then be stored or used in computations.

- **Networking.** A realtime system connected to a network can communicate and share resources with other systems.
- **Read-only memory (ROM).** Some realtime systems must be loaded from ROM.
- **Self-sufficiency.** A self-sufficient realtime application has realtime and time-sharing computing capabilities. You might use such an application both to collect data in realtime and to reduce, manage, or write a report about the data. Self-sufficient realtime applications are often referred to as soft realtime applications.
- **Stability.** A stable realtime system remains relatively unchanged once it is implemented. Most realtime systems that remain stable are dedicated.
- **System resource control.** Most realtime systems (specifically, control systems) optimize the use of system resources by controlling their availability. Time-critical applications require that adequate CPU time, physical memory, and I/O bandwidth be available when needed. Realtime applications can control the availability of these resources through asynchronous execution of multiple code paths within a single application. In the broadest sense, this includes:
  - Code paths that external events invoke, such as interrupt service routines (ISRs) and I/O completion routines
  - Multiple detached processes and subprocesses executing on a single processor or on multiple processors in a distributed configuration

  Realtime systems must be able to perform operations such as context switching and interprocess communication and synchronization efficiently.

1.3 Industry Standards

Digital supports and promotes industry standards by providing the capability of developing applications in a POSIX environment for the VMS operating system, DECelx Realtime Tools for ULTRIX, DEC OSF/1, and VAXELN Toolkit.

You can develop realtime solutions using standard programming languages such as Ada, C, FORTRAN, or Pascal. Networking standards, such as Transmission Control Protocol/Internet Protocol (TCP/IP) and DECnet/OSI, let distributed applications and their components communicate across local or wide area networks (LANs or WANs). If your solution requires a graphics interface, you can integrate DECwindows, which is based on the industry-standard X Window System. If your solution incorporates third-party hardware or open I/O busses, the Q–bus and a variety of industry-standard busses (such as the VME [Versa Module Eurocard] bus) can be integrated with the rtVAX 300 processor component.

Digital supports key standards from the Open Software Foundation's Distributed Computing Environment (OSF/DCE) on VMS and DEC OSF/1 systems. DCE is a highly integrated set of open systems technologies, built in acccordance with OSF guidelines, that address transparent-computing multivendor environments. Digital has already announced support for and is currently shipping OSF/Motif, an industry-standard graphical user interface.

1.4 Open Systems Support

Open systems computing means multivendor integration—true systems integration regardless of your hardware platform or operating system. Digital offers true open systems computing through these key commitments:

- Standards compliance and support, Section 1.4.1
- Open Software Foundation (OSF), Section 1.4.2
- Network Application Support (NAS), Section 1.4.3

In 1988, Digital was one of the seven founding members of the Open Software Foundation. Today, OSF members together represent the mainstream of the computing industry. In addition to Digital, there are over 275 members of OSF, including major companies such as Hewlett-Packard/Apollo, IBM, Honeywell, Groupe Bull, and Siemens.

1.4.1 Standards Compliance

Industry standards determine how well a computer system works with other systems. Only through a vendor's adherence to standards can you reap the benefits of open systems. Achieving open systems requires a broad suite of standards, including networking protocols, common user interface specifications, and standard programming interfaces. Only when all these standards come together is a system truly *open*. Digital offers all the benefits of open systems based on industry standards.

Digital complies with major *de facto* and industry standards today, and plays an active role in developing new standards for the future. As a member of major international standards committees, Digital participates in and encourages the development of emerging standards. And Digital's products, offered to the appropriate standards bodies, have been accepted and formalized into industry standards. For example, Digital was instrumental in driving the development of the X and PEX (PHIGS Extension to X) standards.

Digital supports the following industry standards:

Category	Standard
Application Portability	OSF AES, X/Open, ANSI, NIST, FIPS
Operating Environment	POSIX, X/Open, OSF/1
Data Portability	SQL, ANSI, ODA, SGML, CGM
User Portability	OSF/Motif
Graphics	X Window System, GKS, PHIGS, PEX
Networking	TCP/IP, NFS, IEEE 802.3/Ethernet, FDDI, X.25, X.400, OSI, XTI

1.4.2 Open System Foundation

Open Software Foundation (OSF) is an independent corporation; it is not a consortium. It drives open systems standards with open cooperation, aggressively adopting specifications while developing source code. OSF is best known today for its specification and implementation of the popular graphical user interface known as OSF/Motif. OSF chose Digital's XUI (X User Interface) and UIL (User Interface Language) implementations, both part of Digital's DECwindows user interface, for the underlying technologies of OSF/Motif. OSF uses Digital's DECstation 3100 workstations for its software development.

The goal of OSF is to develop specifications for a completely open software environment, not just for an operating system. This ensures application portability and protects your investments. Under the OSF Application Environment Specification (AES), POSIX, X/Open, and other standards are being extended to form a complete applications environment. The AES provides standards-compliant interfaces that will make code truly portable across multivendor distributed systems.

1.4.3 Network Application Support

Today, computer users work in computing environments that include systems from many vendors. Digital unifies this complex environment with Network Application Support (NAS), a framework and set of products that enables new and existing applications that run in a multivendor environment to share information and resources. For example, Digital's family of PC LAN software, PATHWORKS, connects systems that run the ULTRIX, DOS, OS/2, VMS, and Macintosh operating systems. NAS is a path to true open computing and communication regardless of the operating platform.

1.5 POSIX Standards

Digital's DECelx, DEC OSF/1, VAXELN, and VMS operating systems support the widely accepted Portable Operating System Interface for Computer Environments (POSIX) standards of the Institute of Electrical and Electronics Engineers (IEEE). POSIX is a set of standards generated and maintained by standards organizations — they are developed and approved by the Institute of Electrical and Electronics Engineers, Inc.(IEEE) and adopted by the International Standards Organization (ISO) and the International Electrotechnical Commission (IEC). Digital's POSIX implementations follow the standards and drafts defined by the POSIX standards.

The only formal standards to date are POSIX 1003.1 for basic system interfaces and POSIX 1003.13, the test assertions a vendor must test to claim conformance to POSIX 1003.1. Draft standards, such as 1003.4 Draft 11, are not formal standards. These are working documents that will evolve over time into formal standards.

As Digital adds support for evolving and final standards, customers may want to modify their POSIX applications to conform to the latest version of these standards. Because draft standards are working documents and not formal standards, the level of backwards compatibility and formal support for older versions (drafts) will be less than that normally expected from a stable Digital product.

POSIX standards, or draft standards, for the programming interface (Standard 1003.1) and realtime programming extensions (drafts of P1003.4) are supported by DECelx, DEC OSF/1, VAXELN, and VMS. POSIX threads (P1003.4a/D4) are supported by DEC OSF/1 and VMS. In addition, VMS POSIX also supports the shell and utilities (P1003.2/D10). These drafts and standards are described as follows:

Standard	Description
POSIX 1003.1	Defines the standard for basic system services on an operating system, and describes how system services can be used by POSIX applications. These services allow an application to perform operations such as process creation and execution, file system access, and I/O device management.
POSIX 1003.2	Provides a callable and interactive interface to shell and utility services. POSIX 1003.2 support also includes a set of callable interfaces for executing shell commands, compiling and executing regular expressions, and performing pattern matching.

Standard	Description
POSIX 1003.4	Provides support for functions that support the needs of realtime applications, such as enhanced interprocess communication, scheduling and memory management control, and asynchronous I/O operations.
POSIX 1003.4a	Defines a set of thread functions that can be used in the design and creation of multithreaded realtime applications.

An application that strictly conforms to any combination of these standards and drafts can be developed on one system and then ported to another system that supports the same POSIX standards or drafts. (A strictly conforming application uses only the facilities within the applicable standards.) Similarly, an application developed on a non-Digital platform, if it strictly conforms to the POSIX standards and drafts supported by Digital systems, can be ported and run on a Digital system on which the POSIX software is installed.

It is the source code of an application that is portable. Most applications written for a POSIX environment use the C programming language. Each system that supports a POSIX environment includes POSIX runtime libraries as well as C runtime libraries. A portable application that requires an executable image must be compiled and linked on a system after being ported. It is important that you compile and link your POSIX applications against the runtime libraries on the system where they will be run.

The POSIX standards are based on the UNIX environment. However, POSIX specifies an interface to an operating system, not the operating system itself. VMS POSIX and VAXELN POSIX provide environments that include many aspects of both the VMS and VAXELN environments as well as the UNIX environment.

1.5.1 DECelx POSIX

DECelx Realtime Tools for ULTRIX (DECelx) supports the POSIX programming interface (POSIX 1003.1–1990), portions of the realtime programming extensions (P1003.4/D11), and portions of POSIX threads (P1003.4a/D4). These extensions let you develop and run realtime applications on a DECelx system.

1.5.2 DEC OSF/1 POSIX

The DEC OSF/1 operating system supports portions of the POSIX programming interface (POSIX 1003.1–1990) and many of the realtime programming extensions (P1003.4/D11). These extensions let you develop and run realtime applications on a DEC OSF/1 system.

You can use the DEC FUSE tools to develop realtime applications in a COHESION environment.

DEC OSF/1 supports the realtime functions used in the creation of realtime applications in the DEC OSF/1 POSIX environment. For this version of DEC OSF/1, Draft 11 of the proposed POSIX 1003.4 standard (P1003.4/D11) has been used.

1.5.3 VMS POSIX

On a VMS system that includes the VMS POSIX product, you can choose either the VMS environment or the POSIX environment (POSIX 1003.1) and can easily move between the two environments.

Programmers using VMS POSIX must choose either the VMS file system or the POSIX file system for their applications. The VMS file system in a POSIX application is the same as in other applications developed under the VMS operating system. The POSIX file system is portable to other systems supporting POSIX and looks like a UNIX file system.

The shell and utilities make up the command line, or interactive, part of VMS POSIX. You can use the VMS POSIX shell as your command line interpreter, and you can also write shell scripts using the VMS POSIX utilities.

VMS POSIX supports the functions used to create realtime applications in the VMS POSIX environment. For this version of VMS POSIX, Draft 9 of the proposed POSIX 1003.4 standard (P1003.4/D9) has been used.

1.5.4 VAXELN POSIX

The VAXELN Toolkit includes support for programming in the POSIX environment. The VAXELN Toolkit supports portions of the POSIX programming interface (POSIX 1003.1–1990) and realtime programming extensions (P1003.4/D11). VAXELN POSIX lets you develop and run POSIX applications on a VAXELN system.

Programmers using VAXELN POSIX can compile and link POSIX application modules in with VAXELN application modules to create a single program image. (However, you should avoid calling POSIX functions from VAXELN subprocesses.)

You can also use the VAXELN Debugger and VAXELN System Builder with VAXELN POSIX applications. (However, using the POSIX `fork` and `exec` functions may require some additional considerations.)

VAXELN POSIX supports the functions used to create realtime applications in the VAXELN environment. For the current version of VAXELN POSIX, Draft 11 of the proposed POSIX 1003.4 standard (P1003.4/D11) has been used.

1.6 Realtime Products Summary

Digital's realtime product offerings can meet the demands of any realtime application. Digital's operating systems furnish comprehensive realtime system development environments that can communicate with the rest of your enterprise using DECnet or TCP/IP networking products. You can assemble a configuration from a variety of execution environments and platforms.

As Figure 1–1 shows, software products available for realtime application development on VMS systems include the following:

- VMS and VAXELN POSIX support
- VAXELN Toolkit
- High-Level Language Compilers and Runtime Libraries
- DEC Realtime Integrator
- CASE Tools

The software products available for realtime application development on UNIX systems include the following:

- UNIX and DECelx POSIX support
- DECelx Realtime Tools
- DEC OSF/1 preemptive kernel
- High-Level Language Compilers and Runtime Libraries
- DEC Realtime Integrator
- CASE Tools

Depending on the tools you used to develop your VAXELN application, it can run as a VAXELN or VMS system on a VAX configuration or as an XD Ada system on a Motorola 68020 or MIL–STD–1750A microprocessor.

Figure 1–1 Realtime Software Products

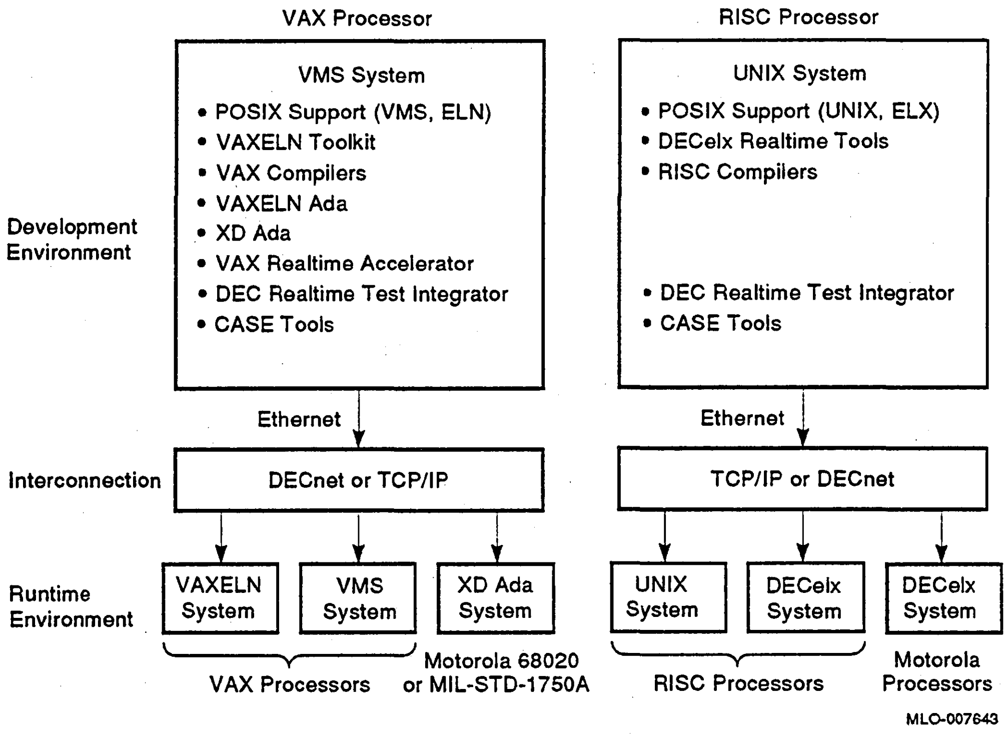


Figure 1–1 shows DEC Realtime Integrator graphical programming tools, UNIX CASE tools, and high-level language compilers also available for an ULTRIX on a MIPS-based system. As with OpenVMS on a VAX and VAXELN systems, your UNIX system can communicate with the rest of your enterprise using DECnet or TCP/IP networking software.

Digital's realtime hardware products include boards, workstations, and systems and a variety of realtime hardware options. These products are described in Part VIII.

Sometimes realtime applications have multiple requirements, some of which are addressed better by one product and some by another. You must weigh the relative importance of each of the application design goals and the comparative development cost required to achieve those goals to determine which approach provides the best solution.

1.6.1 DECelx Realtime Tools

DECelx Realtime Tools for ULTRIX (DECelx) is a powerful development environment for realtime applications. DECelx includes a high-performance runtime executive, powerful testing and debugging facilities, and an unparalleled ULTRIX cross-development package. Networking facilities allow DECelx and ULTRIX to combine to form a complete, integrated development and operational environment for realtime applications.

The ULTRIX operating system environment is used for software development and the non-realtime components of the application. DECelx is used for testing, debugging, and running realtime applications.

For more information about DECelx Realtime Tools, see Part I.

1.6.2 VAXELN Toolkit

The VAXELN Toolkit provides software for developing dedicated realtime (in some cases, embedded) applications that run on VAX platforms, ranging from the rtVAX 300 processor board to the rtVAX 9000 multiprocessor systems. The toolkit lets you create low-overhead systems that include only necessary services, drivers, and utilities. Typical VAXELN applications include industrial automation, process control, robotics, simulation, scientific data acquisition and analysis, image processing, built-in test equipment, file and print servers, and communication switching systems.

You develop a VAXELN application on a VAX processor by using VMS computer-aided software engineering (CASE) and VAXELN development tools. The resulting VAXELN system image includes user and toolkit program images. After creating a system image, you can load or boot the image onto a dedicated target VAX configuration from disk, tape, or ROM. If you have a DECnet license and the appropriate Ethernet hardware, you can downline load the system image from a VAXELN, VMS, or ULTRIX boot-host processor to the target processors. The memory-resident system image runs—independently of the VMS development system—on the target processor configuration.

VAXELN is appropriate for realtime systems that have the following characteristics:

- Predictable responses to realtime tasks in a realtime system
- Dedication to the runtime environment of the realtime application
- Context switching, which constitutes a major and highly demanding component of the processing requirements for fast response to external events
- Minimal memory or mass storage requirements

- Maximal user control of the operating environment with minimal programming effort

For more information about the VAXELN Toolkit, see Part II. In addition, this section presents an example of a dedicated application, the VAXELN Window Server, that was created using the VAXELN Toolkit.

1.6.3 VMS Operating System

The VMS operating system is a full-service operating system widely known in the industry. You can tune a VMS system for realtime performance. For example, you can shut down unused system services, adjust process priorities, and make critical sections of code memory resident. System features are also available for interprocess communication and synchronization, symmetric multiprocessing, and low-overhead I/O.

You can also use VMS POSIX, a product that allows you to develop and run portable realtime applications in a POSIX environment. VMS POSIX includes support for a number of POSIX standards and draft standards, including the realtime POSIX 1003.4 standard (Draft 9).

For more information about VMS realtime systems, see Part III.

1.6.4 UNIX Operating Systems

Digital offers two UNIX operating systems, ULTRIX and DEC OSF/1. Both operating systems are full-service systems that are highly regarded within the industry. The DEC OSF/1 operating system is Digital's implementation of the Open Software Foundations (OSF) operating system component V1.01.

DEC OSF/1 ships with an optionally installable realtime kernel. The realtime kernel provides support for many of the POSIX realtime functions, as specified in P1003.4 Draft 11.

For more information about the DEC OSF/1 realtime kernel, see Part IV.

1.6.5 High-Level Language Compilers and Runtime Libraries

Using Digital's realtime products, you can employ high-level languages to develop realtime applications, without sacrificing performance.

- **VAXELN Toolkit.** With the VAXELN Toolkit, you can use the VAX Ada, VAX C, VAX FORTRAN, or VAXELN Pascal compiler. The toolkit furnishes VAXELN runtime libraries for C, FORTRAN, Pascal, KAV30, and POSIX. The runtime libraries for Ada applications are supplied as part of a separate VAXELN Ada product.

VAXELN Ada has features required by government agencies for applications such as radar control, communication, and navigational systems. This product is well integrated with the VMS and VAXELN environments. In addition to supplying runtime software, the VAXELN Ada product includes an interactive remote debugger.

If your application is to run on a Motorola 680*n*0 or MIL–STD–1750A microprocessor, you can develop the application using Digital's XD Ada product. XD Ada is a set of development tools for implementing commercial and military embedded realtime systems.

- **ULTRIX.** With ULTRIX you can use the DEC C, DEC C++, DEC Fortran, DEC Pascal, or the Pascal for RISC compiler. ULTRIX furnishes runtime libraries for C, C++ Fortran, Pascal, and POSIX.

  Other languages available through Digital include Ada, COBOL, and Lisp.
- **DEC OSF/1.** With DEC OSF/1 you can use the DEC C, DEC Fortran, or DEC Pascal compiler. DEC OSF/1 furnishes runtime libraries for C, Fortran, Pascal, and POSIX.
- **DECelx.** With DECelx you can use either the MIPS C compiler or the GNU C compiler, depending on your target system.

For more information about Digital's high-level language support for realtime applications, see Part V.

1.6.6 Digital's COHESION Environment

Successful research organizations are increasingly using computer-aided software engineering (CASE) tools to reduce the time and effort required to write application code and leave more time for product research. Digital's COHESION environment is the industry's most comprehensive and inclusive CASE environment for developing, using, and managing software. The COHESION environment also furnishes a highly efficient, full-service development environment for developing realtime applications in a multivendor environment.

Encompassed in the COHESION vision are all industries and markets, all project sizes, and all styles of computing. To realize this vision, the following strategic goals are included in the COHESION solution:

- Provide a full range of quality software services and products that can run on multiple platforms
- Link technological and business planning
- Offer a common environment across the enterprise

- Provide a software development platform that can integrate software from Digital and other vendors
- Provide a high level of service

For more information about Digital's CASE tools and the COHESION environment, see Part VI.

1.6.7 DEC Realtime Integrator

For data acquisition, instrument control, and test and measurement applications, you can consider using DEC Realtime Integrator. This software provides an icon-based, graphical programming environment. Instead of using a conventional programming language, the user can create and run realtime applications by drawing them graphically as flow diagrams.

Each DEC Realtime Integrator icon represents a function, such as analog or digital input, an arithmetic operation, or a logical function. DEC Realtime Integrator provides several libraries of commonly used functions, to which you can add your own functions and icons. You create applications by using a mouse to select icons from different libraries, placing them on a work surface on the computer screen, connecting them up with data flow lines, and performing further setup with pop-up menus. To run a program, you simply click on the start button.

DEC Realtime Integrator is also available for ULTRIX on MIPS-based systems. Using DEC Realtime Integrator for ULTRIX, you can develop realtime applications using a graphical programming interface. Once developed, the application can be run on the same ULTRIX system or, with very few modifications, run on any other VAX or MIPS-based system appropriately configured and loaded with DEC Realtime Integrator software.

For the OEM and end user working with engineering or scientific realtime applications, DEC Realtime Integrator can simplify the development of high-quality test and research solutions.

For more information about DEC Realtime Integrator, see Part VII.

1.6.8 Realtime VAX Hardware

Realtime hardware products are designed to meet the demanding requirements of many factory, laboratory, and simulation activities, and to perform either as standalone computing solutions or as integral parts of corporate-wide, distributed networks. Digital's realtime products include VAX and MIPS-based components, workstations, systems, software, and networking tools to provide the complete environment you need for developing realtime solutions for your organization. Realtime products can be migrated to equipment and options with higher performance and functionality.

The realtime hardware product family offers chip-level processors (CLPs), single-board computers (SBCs), workstations, and system-level configurations of Digital's 32-bit VAX and MIPS-based computers, housed in many types of enclosures. The realtime hardware product family spans a wide range of processing power and can meet realtime computing needs on many levels.

Digital also offers a variety of hardware options that you can attach to your realtime systems to:

- Generate high-speed analog signals
- Convert and transfer analog and digital data
- Scan data
- Perform high-speed DMA parallel I/O
- Interface with laboratory instrumentation
- Serve as a realtime clock
- Provide memory for high-performance data collection, reduction, and analysis

For manufacturing environments that require rugged electronic equipment (such as factory floors, assembly areas, and loading docks), Digital furnishes a family of industrialized products. Built to withstand harsh environments, this set of products includes two series of industrialized terminals. You can place these terminals, which are packaged in sturdy enclosures, wherever you need to access or collect data.

For more information about Digital's realtime hardware products, see Part VIII.

1.7 Digital-Sponsored Programs for Promoting Realtime Solutions

To encourage third parties to plan products that easily integrate into Digital computing environments, Digital develops and sponsors programs that promote realtime solutions. Four such programs are Digital's Cooperative Marketing Program (CMP), Digital's Enterprise Integration Services (EIS), the Technical OEM organization, and Digital's Third Parties with Add-On Products (TRI/ADD) Program.

1.7.1 Cooperative Marketing Program

Digital's Cooperative Marketing Program (CMP) encourages cooperation with third-party vendors that specialize in specific application areas. By sharing product goals and directions, CMP participants and Digital can provide customers with the best integrated systems available on the market.

CMP members are recognized leaders in specific market segments. Their application solutions complement Digital's strategies, support Digital's architectures, satisfy network and communications requirements, support current system software updates. Members must pass a rigorous technical evaluation before gaining acceptance to the CMP.

1.7.2 Enterprise Integration Services

Digital's Enterprise Integration Services (EIS) are provided for realtime customers through the Manufacturing and Government Enterprise Integration Center (M&G EIC). This center delivers complete integrated solutions according to customer specifications. This might include:

- Device drivers for third party VME or SCSI devices
- Custom hardware development
- System integration

In support of Digital's commitment to open buses, the M&G EIC has developed extensive knowledge of the VMEbus, and for embedded realtime applications, the DECelx and VAXELN software environments. One result of this knowledge is the delivery of the Digital R3000-based single board computer (SBC).
The M&G EIC exploits a variety of competencies to solve customer realtime computing needs. Expert-level knowledge of DECelx Realtime Tools for ULTRIX, the VAXELN Toolkit, DEC OSF/1, VMS device drivers, and VME hardware can help find the best solution for customers.

The scope of EIC involvement can range from presales support and consultation services to complete solution delivery. The scope is determined by customer needs.

Customers that might need this level of dedicated realtime project services include Original Equipment Manufacturers (OEMs) and Complimentary Solutions Organization (CSO) partners. The services these customers require can vary as follows:

- Design start-up

Customers that use Digital realtime products and third-party solutions as an integral part of their product or solution might need EIC project services during the start-up phase of the product's design. A feasibility study or functional specification might be the deliverable.

- Prototype development

  Customers might want to shorten their development cycle for a solution subsystem prototype by requesting the EIC to develop the prototype according to their specifications. The customer might then want to develop the production integration system on their own.

- Complete integration services

  Customers might want the M&G EIC to provide the highest level of system integration for complete packaged solutions. This saves customers from negotiating with suppliers for VME crates, software development environments, boards, and components.

In all of the preceding scenarios, the customer benefits from the EIC's experience with Digital products and knowledge of available VME products. If required, Digital's Customer Services organization can support EIC solutions worldwide.

1.7.3 Technical OEM Organization

A new Technical OEM (TOEM) business group was formed recently to provide focus for TOEM marketing and sales efforts around the world. One of the goals of the TOEM group is to provide products and support efficiently, on a world-wide scale. Technical OEMs provide customers with solutions, and increasingly, a large number of these solutions include realtime functionality, high performance I/O, and specialized peripherals.

With market needs growing and products and services becoming more globalized, no single vendor, Digital or otherwise, has the resources necessary to meet all industrial or realtime requirements. To meet customer demand for timely, high quality solutions, TOEMs make use of either Digital or third party components. Software and hardware vendors who have expertise and can provide quick and reliable solutions are valuable resources for industry demands.

1.7.4 TRI/ADD Program

When configuring Realtime Solutions, many third-party hardware add-on boards and peripherals are available to be used on Digital platforms. A significant source for many realtime component products is Digital's Third Parties with Add-On Products for MIPS UNIX Platforms (TRI/ADD) Program. However, a customer or Digital sales representative must contact vendors directly about a product. To learn which vendors to contact, the TRI/ADD Shippable Products Catalog lists the growing number of clocks, converters, adapters to other buses, and other third-party realtime products. Two versions of the catalog are updated monthly:

- The short version lists the product's name, the vendor's name, and the vendor's telephone number. This version of the catalog is organized around the interconnect (TURBOchannel, SCSI, and so on) on which the product is based.

  The file name for the short catalog is `shortTAcatalog.txt`, an ASCII file.

- The full version lists all the above information and also gives a description of the product, and addresses and phone numbers of the vendor's sales offices worldwide. This version of the catalog is organized according to type of product (3D peripherals, adapters and controllers, and so on). The Full Version catalog contains a "Coming Attractions" section that broadcasts TURBOChannel based products before their shipment.

  The file name for the full catalog is `TAcatalog.ps`, a PostScript file.

A Digital sales representative can call TRI/ADD for information about how to get the catalog.

U.S. and German DECdirect catalogs also include information about TRI/ADD vendors' products.

Digital's TRI/ADD Program provides technical and marketing support worldwide to third-party vendors using the SCSI, TURBOchannel, VME, ACCESS.bus, and Futurebus+ interconnects to develop add-on products for open systems. Complimentary membership and services are restricted to third-party vendors. Customers can recommend their third-party vendor contact TRI/ADD about membership in these circumstances:

- A third-party device, such as a SCSI disk, does not work on a particular Digital system, but the customer has successfully connected it to other systems.
- A hardware product used on another system needs porting to a Digital system.

A customer or sales representative should contact TRI/ADD when in need of a state-of-technology product or one in an emerging technological area and the product is not listed in the catalog. TRI/ADD specializes in several product areas: high-performance networking and database, and graphics, imaging, and multimedia. The TRI/ADD Program number is [1] 415.617.3452.

Part I
DECelx

Part I surveys DECelx Realtime Tools for ULTRIX, a software product for developing dedicated realtime applications for several processor boards based on the VMEbus architecture. A DECelx application is developed on an ULTRIX host system and executes on one of the supported DECelx target boards, based on the Motorola 680*n*0 or the R3000. This part contains the following chapters:

- Chapter 2, DECelx Overview, introduces the DECelx software, its features, use, and requirements.
- Chapter 3, DECelx Components, describes the development and runtime components of the DECelx software.

2 DECelx Overview

DECelx Realtime Tools for ULTRIX (DECelx) is a software product for developing dedicated realtime and distributed applications that run on a supported set of Motorola 680*n*0-based and MIPS R3000-based processor platforms. The DECelx realtime system includes a high-performance runtime executive, powerful testing and debugging facilities, and an unparalleled ULTRIX cross-development package, at the heart of which lies DECelx's extensive UNIX-compatible networking facilities.

The networking facilities allow DECelx and ULTRIX systems to combine to form a complete, integrated development and runtime environment. Each product is used for what it does best. The ULTRIX system is used for software development and the non-realtime components of applications; DECelx software is used for testing, debugging, and running realtime applications.

This chapter includes the following sections:

- DECelx Realtime System Features, Section 2.1
- DECelx Realtime Tools for ULTRIX, Section 2.2
- The Development Cycle, Section 2.3
- DECelx Board Support Packages, Section 2.4
- DECelx Target Hardware, Section 2.5
- DECelx System Hardware and Software Requirements, Section 2.6

Once development is complete, the DECelx system can operate standalone, embedded, or networked with other systems running DECelx, UNIX, or any other operating system with TCP/IP networking facilities.

An ULTRIX system that is usable as a UNIX host for DECelx system development has the DECelx Realtime Tools and either or both the DECelx BSP for MIPS or DECelx BSP for 68K board support package (BSP) kits installed.

The DECelx BSP for MIPS is based on the R3000. The DECelx BSP for 68K is based on the Motorola 680*n*0 series.[1] These BSP kits include support for the following hardware:

BSP kit	Supported Hardware
MIPS kit	Lockheed/Sanders STAR MVP
	Omnibyte VR3000
	Radstone SL–3000
	Personal DECstation 5000 models
68K kit	Motorola MVME133XT
	Motorola MVME147S–1
	Motorola MVME167

The VMEbus or TURBOchannel I/O devices are available on all supported target boards, allowing the use of a wide range of third-party I/O devices.

The DECelx software offers limited support, to be increased in future versions, for the following POSIX standards and draft standards: POSIX 1003.1–1990 and P1003.4/D11.

2.1 DECelx Realtime System Features

The features of the DECelx realtime system include the following:

- **High-performance realtime kernel facilities.** Multitasking with preemptive priority scheduling, intertask synchronization and communications facilities, interrupt handling support, watchdog timers, and memory management.
- **POSIX synchronization facilities.** Functions defined by the P1003.4/D11 standard that support semaphore, clock, and timer operations for realtime applications.
- **Network facilities.** Transparent access to other DECelx and UNIX systems via UNIX source-compatible sockets, remote command execution, remote login, remote procedure calls (RPC), source-level remote debugging, and remote file access. These all use TCP/IP network protocols both loosely coupled over standard Ethernet connections and tightly coupled over a backplane bus using shared memory.

[1] Refer to the SPD and SSA for a complete, updated list of supported boards.

- **Module loader and system symbol table.** Dynamic loading of object modules over the network or from a disk, with runtime relocation and linking.
- **Shell.** A C-interpreter interface that allows interactive execution of most C language expressions, DECelx functions, and any other loaded functions, and that also includes symbolic references to variables.
- **Debugging facilities.** Remote source-level debugging, a symbolic disassembler, symbolic C-subroutine traceback, task-specific breakpoints and single-stepping, system status displays, and exception handling to safely trap and report on interrupts and hardware exceptions such as bus or address errors.
- **I/O system.** A fast and flexible I/O system that is compatible with UNIX sources, including UNIX standard buffered I/O.
- **Local file systems.** Fast file systems appropriate for realtime and compatible with the MS–DOS and RT–11 file systems and a raw disk file system that treats an entire disk much like a large file.
- **Remote file system.** Network File System (NFS) facilities for accessing files transparently on any NFS server on the network, and a non-NFS network facility for accessing the host file systems using **rsh** or **ftp**.
- **Performance evaluation tools.** An execution timer for timing a routine or group of routines, and utilities to show processor utilization percentage by task.
- **Utility libraries.** An extensive set of utility functions available to application developers, including: message logging, string formatting and scanning, linear and ring buffer manipulations, linked-list manipulations, and symbol table manipulation.
- **I/O drivers.** The following drivers are included:

Driver	Device
Ty driver	Serial I/O devices
Network driver	Remote files
Pipe driver	Intertask communication
RAM disk driver	Memory resident files
SCSI library	SCSI hard disks and floppies

- **Board-support packages.** Routines for hardware initialization, interrupt setup, timers, memory mapping, and so on.

- **Boot ROMs.** Allow a target processor to be booted directly over the network.
- **System configuration utilities.** Allow reconfiguration and extension of DECelx and building applications in ROM.

Chapter 3 outlines each of the components listed above. Appendix A provides references to further documentation.

2.2 DECelx Realtime Tools for ULTRIX

The objective behind the DECelx Realtime Tools for ULTRIX product is to network two different but cooperating operating systems in a single cross-development environment. DECelx systems can be used to handle the critical realtime chores, while ULTRIX can be used for program development and for non-time-critical applications. The DECelx and ULTRIX systems work well together because DECelx was designed to be UNIX-compatible at many levels, especially in its extensive networking facilities. Distributed applications can have DECelx and UNIX systems in combination, or the entire distributed application can be based on DECelx systems.

As a cross-development host, the ULTRIX system is used to edit, compile, link, and store realtime code, which is then run and debugged on DECelx. The resulting DECelx application can then be downline loaded via the network or can run standalone in ROM with no further need for the network or the development system.

ULTRIX and DECelx systems can also work together in a hybrid application, with ULTRIX (and other UNIX systems) using DECelx as a realtime server in a networked environment. For instance, a DECelx system communicating with a robot can be controlled by an expert system running in a UNIX environment. A number of DECelx systems running factory equipment can be connected to UNIX systems tracking inventory or generating reports.

2.3 DECelx Development Cycle

To understand the environment provided by the DECelx Realtime Tools, consider a typical development cycle. This section outlines typical DEcelx hardware and software development environments.

2.3.1 Developing a DECelx Hardware Environment

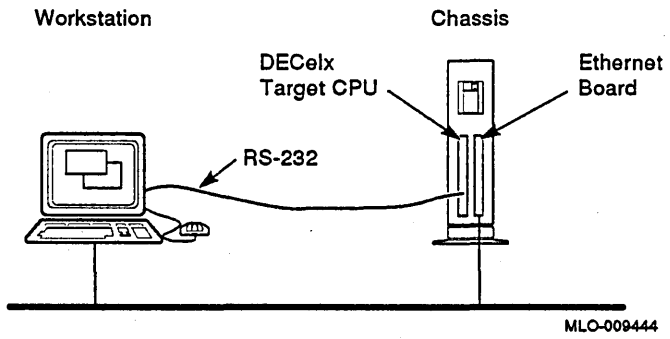
The hardware in a typical development environment includes one or more multi-user ULTRIX host systems and one or more single-user DECelx target systems connected by an Ethernet network.

The ULTRIX host system can be fully loaded with large main memory, large disks, backup media, printers, and terminals.

The target systems, on the other hand, usually have only the resources required by the realtime system, and software for testing and debugging. This may be as little as a processor, some serial I/O channels, and an Ethernet connection. Figure 2–1 shows a typical minimum target system configuration, including the following:

Chassis	A card cage with VMEbus backplane and power supply.
Processor board	A single-board target computer supported by DECelx software.
Ethernet board	An Ethernet controller board (some processor boards include the Ethernet controller on-board).
Console	An ASCII terminal or a serial port on a workstation; this is required for initial setup only.

Figure 2–1 Typical Minimum Configuration for DECelx



DECelx supports network communications between an ULTRIX host system and target systems connected to a common VME backplane. The ULTRIX backplane driver uses the TURBOchannel/VME adapter and provides a bsd socket interface between ULTRIX processes on MIPS-based DECstations and DECelx tasks running on the VMEbus.

All shared memory for the backplane network is allocated on the host system and the host system driver must be configured as the software backplane master. To use the uLTRIX VME backplane driver rebuild the ULTRIX kernel to include the driver and the TURBOchannel/VME adapter in the configuration.

2.3.2 Developing a DECelx Application

Software development for a realtime system begins on the ULTRIX host development system. Using the development and management tools on ULTRIX, the application team begins to design and implement the application modules. Developers are free to use the standard ULTRIX tools such as text editors, compilers, assemblers, the **make** utility, source code control, and so on. The following list explains the basic steps to develop a DECelx application.

1. Create the application source code

   On the host system, use an ULTRIX editor such as `vi` or EMACS, to create a file containing C code.

2. Compile the source code

   On the host system, use the ULTRIX or DECelx GNU C compiler to process the source code. DECelx supports the loading of MIPS COFF files on all big-endian and little-endian target systems. This means that you no longer need to convert MIPS COFF files to bsd `a.out` format for loading.

   Ada application modules are compiled on the UNIX host system using the DEC Ada compiler for the runtime environment on MIPS R3000 little-endian targets.

3. Link the object modules

   You might want to link some of your application object modules on the host system. You might do this if two modules cross-reference each other or if you want to link related modules to reduce the number of modules that you will need to load onto the target system. To link modules on the host system, use an ULTRIX linker or the DECelx GNU linker.

4. Load the application modules

   On the target system, use the DECelx module loader from the DECelx shell to load the application modules into the target system's memory. If you chose not to link the object modules on the host system, the module loader performs runtime linking operations on the target system. The module loader provides you with the option of loading and debugging modules incrementally.

DECelx Realtime Tools provide host system tools for developing DECelx applications that can run on R3000-based or 680*n*0-based target processors. Target system tools include header files and a CPU variable, a shell, a module loader, a system symbol table, and a sample `Makefile` for rebuilding DECelx system images. Table 2–1 lists the host system development tools you can use for each type of target processor:

Table 2–1 DECelx Development Tools by Target

MIPS R3000	Motorola 680*n*0
ULTRIX C compiler	DECelx GNU C compiler
	DECelx GNU C preprocessor
	DECelx GNU assembler
`MIPSToBsd` utility	
ULTRIX `ld` linker	UNIX `ld68k` linker
	DECelx GNU linker
DECelx header files	DECelx header files
Configuration files	Configuration files
`Makefiles` and `make`	`Makefiles` and `make`

C application modules are compiled with a C compiler appropriate for the target system; the native MIPS/ULTRIX C compiler for R3000-based targets and the DECelx GNU C cross-compiler for Motorola 680*n*0-based targets.

2.3.3 Loading a DECelx Target

DECelx can load the compiler-generated object modules directly into a target, using the symbol table contained in all ULTRIX object modules to resolve external symbol references dynamically. The application modules do not need to be linked with the DECelx system libraries or even with each other.

For testing and debugging, a DECelx target system can be booted via Ethernet and selected modules can be dynamically loaded, across the network, onto the running DECelx target. The DECelx shell program can then be used to invoke and test individual application subroutines interactively, or to complete tasks.

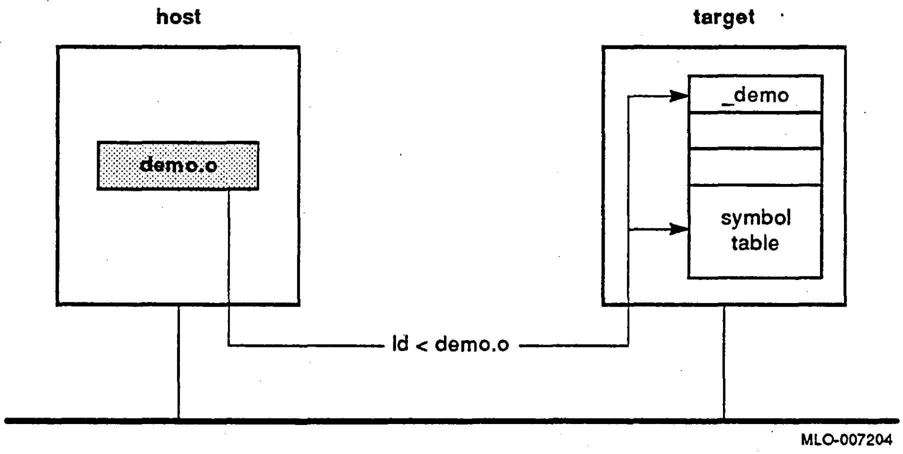
The command to load applications into DECelx system memory, `ld`, performs three functions:

1. Loads the program into memory
2. Adds the program's symbols to the system symbol table

3. Resolves the program's external references

Figure 2–2 shows the result of loading a program module, `demo.o`, from an ULTRIX host to a DECelx target.

Figure 2–2 Downloading a Program



DECelx stores the symbol tables from previously loaded object modules, allowing symbolic access to data and subroutine names. Users can examine data variables, call subroutines, spawn tasks, disassemble code in memory, set breakpoints, obtain subroutine call tracebacks, and so on, using the original symbolic names. Program errors detected by the hardware, such as illegal memory references or illegal instructions, are safely trapped and reported by DECelx, allowing further symbolic debugging.

DECelx offers three debuggers: a basic symbolic debugger for all targets; a ElxGDB debugger for the 680*n*0; and a ElxGDB debugger for the R3000. ElxGDB is a powerful remote source-level debugger that allows the application to be viewed and debugged in the original source code. For example, setting breakpoints, single-stepping, and examining variables can be done at the source level, using either commands at an ASCII terminal or a mouse-based menu-driven interface on a windowed workstation.

The cycle of building, downloading, and testing modules is repeated until the application is ready for the production environment. DECelx debugging facilities can be removed from the production system, if necessary, to produce a system requiring minimal resources. At that point, the application can easily be linked with DECelx, and put into ROM if so desired.

2.4 DECelx Board Support Packages

The DECelx Toolkit includes two board support packages consisting of libraries that adapt the kernel to the supported architectures. These libraries furnish an identical software interface for the hardware-specific functions of DECelx target computers. The libraries contain routines to handle hardware initialization, interrupts, memory mapping and sizing, clock and timer control, and device drivers.

This software makes most of the differences among target computers transparent to the kernel and to DECelx applications.

Digital provides a Board Porting Kit to help you create DECelx applications for unsupported hardware. The Board Porting Kit helps you modify selected DECelx kernel functions such that an application can run on unsupported target hardware, as long as the target hardware belongs to the MIPS R3000 or Motorola 680*n*0 series processors.

2.5 DECelx Target Hardware

DECelx supports two architectures, Motorola 680*n*0 and the MIPS R3000, and seven platforms based on those architectures.

You use a RISC-based Digital system running the ULTRIX operating system as the DECelx development system, or host. You use a single-board computer (based on the MIPS R3000 processor or the Motorola 680*n*0 processor) as a runtime system, or target.

When a target system is part of a network, the target can communicate with other DECelx systems, ULTRIX systems, UNIX systems, or any other computer systems that use the Internet Protocol Suite. Flexible DECelx network software permits multiple target systems to communicate across an Ethernet link or a backplane.

For many distributed applications, the host computer used for development subsequently functions as a file server or database server for the DECelx system and the clients in a distributed network. Or, the DECelx systems can be used as realtime servers in a networked environment. For example, a DECelx system controlling a robot or running factory equipment can communicate with an expert system running in a UNIX environment to track inventory or generate reports.

All supported DECelx single-board computers (except the Personal DECstation 5000) support the VMEbus. The VMEbus is currently one of the most popular industry-standard busses with many options that are well understood by systems and peripherals vendors. The VMEbus is an industry-standard, high-performance, open I/O interconnect that features microprocessor independence.

The Personal DECstation 5000 supports TURBOchannel, an open bus standard developed by Digital. Many third party TURBOchannel I/O options are available through Digital's Third Parties with Add-On Products for RISC UNIX Platforms (TRI/ADD) Program.

2.5.1 Microprocessors on the R3000 Architecture

DECelx applications can run on four supported target microprocessors on the R3000 architecture (MIPS kit).

- Personal DECstation 5000 models [1]
- Lockheed/Sanders STAR MVP
- Omnibyte VR3000
- Radstone SL–3000[2]

Table 2–2 compares common features of the supported R3000 target boards.

Table 2–2 R3000 Target Board Summary

Feature	DECstation 5000	STAR MVP	SL–3000	VR3000
Serial ports	1	2	2	1
On-board SCSI	Yes	No	Yes	No
VMEbus	No	Yes	Yes	Yes
TURBOchannel	Yes	No	No	No
On-board Ethernet	Yes	No	Yes	No
Clock Frequency	25.0 or 33.0 MHz	25.0 MHz	40.0 MHz	25.0 MHz

[1] Note that support for the DECstation 5000 models are without graphics and are rack mountable.

[2] Refer to the SPD for the Digital model name.

2.5.2 Microprocessors on Motorola Architecture

DECelx applications can run on three supported target microprocessors on the Motorola architecture (68K kit).

- Motorola MVME133XT
- Motorola MVME147S–1
- Motorola MVME167

Table 2–3 compares common features of the supported Motorola target boards.

Table 2–3 Motorola Target Board Summary

Feature	MVME133XT	MVME147S–1	MVME167
Serial ports	2	2	4
On-board SCSI	No	Yes	Yes
VMEbus	Yes	Yes	Yes
On-board Ethernet	No	Yes	Yes
Clock Frequency	25.0 MHz	25.0 MHz	25.0 MHz
Timers	3 (8-bit)	3 (16-bit)	4 (32-bit)

2.5.3 Realtime Options

Digital's TRI/ADD Program provides technical and marketing support worldwide to third-party vendors using the SCSI, TURBOchannel, VMEbus, ACCESSbus, and Futurebus+ interconnects to develop add-on products for open systems.

A TRI/ADD catalog lists vendor-supplied information on available products. Twelve catagories of products are listed, ranging from software to hardware. A division of the TRI/ADD program specializes in third-party products for the development and functioning of realtime systems. The TRI/ADD catalog provides easy access to third-party, add-on product information.

In addition to the services provided by the TRI/ADD Program, Digital offers services for system integration and the development of VME based solutions. through its Enterprise Integration (EIS). Section 1.7.4 provides additional information on the TRI/ADD Program.

2.6 DECelx Hardware and Software Requirements

This section identifies DECelx hardware and software requirements for developing and running realtime applications.

DECelx Hardware Requirements

You can install the DECelx software and run the host development tools on any ULTRIX system. If the ULTRIX development host includes Ethernet interfaces and the appropriate interconnecting hardware, you can download your DECelx object modules or fully-linked applications to a supported, network-connected target processor. The Ethernet hardware also provides the necessary communication link for remote debugging from the ULTRIX host or network communication between the DECelx target and the ULTRIX host.

A DECelx target configuration must include a target processor that the DECelx software supports. The DECelx Version 1.0 software supports the following 680*n*0-based or R3000-based target systems: Motorola MVME133XT (68020-based), Motorola MVME147S–1 (68030-based), Motorola MVME167 (68040-based), Lockheed/Sanders STAR MVP (R3000-based), Personal DECstation 5000 models (R3000-based), and Omnibyte VR3000 (R3000-based). For details on currently supported systems, see the latest *DECelx System Support Addendum*, obtained from your Digital sales representative. The target processors can be standalone systems or Ethernet-connected systems distributed on a local area network (LAN).

Application-specific target hardware requirements include Ethernet hardware, serial I/O hardware, and VMEbus peripheral devices. If your DECelx application requires downloading or remote debugging, or makes use of network facilities, the target configuration must include the appropriate Ethernet hardware. If you want to execute functions and commands interactively using the DECelx shell on a target terminal, serial hardware must be configured on the target. Application-specific peripheral devices supplied by Digital or a third party may also be required. If you use the DECelx backplane driver, the VME-to-TURBOchannel adapter is required.

DECelx Software Requirements

The software requirements for a DECelx system are as follows:

- ULTRIX Version 4.2A or higher
- MIPS C for ULTRIX (for R3000-based systems)
- GNU C (for Motorola 680*n*0-based systems)

3 DECelx Components

You develop a DECelx application on an ULTRIX development host by using a comprehensive set of ULTRIX and DECelx software tools and libraries. The code you develop — ranging from an object module to a fully linked application — is loaded into a supported target processor for execution, testing, and debugging.

This chapter describes the following DECelx host and target software components:

- Kernel facilities, Section 3.1
- POSIX synchronization facilities, Section 3.2
- Network facilities, Section 3.3
- Module loader and system symbol table, Section 3.4
- Shell, Section 3.5
- Debug facilities, Section 3.6
- Performance evaluation tools, Section 3.7
- I/O system and I/O drivers, Section 3.8
- Local file systems, Section 3.9
- Utility libraries, Section 3.10
- Board support packages, Section 3.11

For additional sources of information on the DECelx software, refer to Appendix A.

3.1 Kernel Facilities for Multitasking and Intertask Communications

Modern realtime systems are based on multitasking and intertask communications. A multitasking environment allows realtime applications to be constructed as set of independent tasks, each with its own thread of execution and set of system resources. The intertask communication facilities allow these tasks to synchronize and communicate to coordinate their activity.

The DECelx multitasking kernel uses interrupt-driven, priority-based task scheduling. It features fast context switch times and low interrupt latency. In a DECelx environment, any C subroutine may be spawned as a separate task, with its own context and stack. Task control allows tasks to be suspended, resumed, deleted, delayed, and moved in priority.

The DECelx software supplies several types of traditional task-blocking semaphores to perform task synchronization and mutual exclusion. DECelx semaphores are fast and efficient. In addition to being available to application builders, they have also been used extensively in building DECelx higher-level facilities.

For intertask communications, the DECelx software also supplies a fast and flexible message queue facility, intertask pipes, sockets, and signals. UNIX sockets are a method for exchanging byte streams in a networked application task regardless of location. UNIX signals are a method for asynchronous transfer of control within a task, based on hardware or software exceptions.

3.2 POSIX Synchronization Facilities

The DECelx software provides limited support for the following POSIX standards and draft standards:

- POSIX 1003.1–1990
- P1003.4/D11

The POSIX 1003.1 standard defines a set of functions for use in application programs using the C programming language. The purpose of using the POSIX 1003.1 standard in an application is to promote application portability among systems that support the POSIX 1003.1 standard. POSIX 1003.1 includes some functions that are identical to ANSI C functions, some functions that use the same syntax as ANSI C functions, but operate slightly differently in the POSIX environment, and some functions that are unique to POSIX.

The P1003.4/D11 draft standard defines a set of functions that can be used in the design and creation of realtime applications in the DECelx POSIX environment. The subset of these functions offered by the current version of the DECelx software facilitate realtime application development in the area of process synchronization.

Synchronization techniques and restrictions on resource access ensure that critical and noncritical activities execute at appropriate times with the necessary resources available. Table 3–1 lists the synchronization routines available in the DECelx POSIX runtime library.

Table 3–1 Synchronization Routines Available in DECelx POSIX

Types of Synchronization Routines	Header File	Purpose
Binary semaphores	`<binsem.h>`	Restricts access to resources
Clocks and timers	`<timers.h>`	Arms and disables timers

3.2.1 Binary Semaphores

A binary semaphore is a synchronization mechanism used to control access to systemwide resources. With DECelx POSIX binary semaphores, you can create and remove binary semaphores. You can also release, wait for, and lock binary semaphores. Each action on a semaphore requires an explicit function call and is under the control of the application programmer.

Semaphores are used by cooperating processes to synchronize access to resources, such as shared memory. Semaphores can protect resources such as global variables, hardware resources, and the kernel from uncontrolled access.

Semaphore protection works only if all communicating processes using the shared resource cooperate by waiting for the semaphore when it is unavailable and resetting the semaphore count when relinquishing the resource. For cooperating tasks, semaphores are mutual exclusion flags that lock and unlock a resource.

3.2.2 Clocks and Timers

Realtime clocks and timers allow an application developer to synchronize and coordinate activities according to a predefined schedule. The systemwide clock (CLOCK_REALTIME) provides the timing base for per-process timers. DECelx POSIX clock and timer functions allow you to retrieve and set the systemwide clock, suspend execution for a period of time, provide high-resolution timers, and use asynchronous event notification. Realtime timers are created, armed, and removed by the application programmer.

Realtime timers allow the application to set timers based on either absolute or relative time. Furthermore, a DECelx POSIX timer can fire as either a one-shot or periodic timer. The application creates timers in advance, but the timers can be manipulated according to the needs of the realtime application. Some applications may require only one or two timers; others may require multiple timers within a single process.

These timers use the nanosecond as the smallest unit of time, which makes them suitable for realtime applications.

3.2.3 POSIX Synchronization Functions

Table 3–2 categorizes synchronization functions and lists the corresponding DECelx POSIX functions.

Table 3–2 DECelx POSIX Process Synchronization Functions

Operation	Function
Make a binary semaphore set	`sem_mksem`

Open, close, and delete a binary semaphore set	`sem_open`
	`sem_close`
	`sem_destroy`

Return the number of semaphores in a semaphore set	`sem_getnsems`
Wait (or conditionally wait) for a binary semaphore	`sem_wait` `sem_ifwait`
Release a binary semaphore	`sem_post`
Get or set the value of the systemwide clock	`clock_gettime` `clock_settime`
Get the resolution of the clock	`clock_getres`
Get or set the clock drift rate	`clock_getdrift` `clock_setdrift`

Allocate or free a per-process timer, get the value of the per-process timer	`timer_create`
	`timer_delete`
	`timer_gettime`

Arm a per-process timer absolutely or relatively	`timer_settime`
Return the timer expiration overrun	`timer_getoverrun`
Suspend current task until time interval elapses	`nanosleep`

DECelx also provides the `clockLibInit` function to initialize the systemwide clock.

3.3 Networking Facilities

The key to the partnership between the DECelx and ULTRIX software is extensive networking facilities. By providing a fast, easy-to-use connection between the two systems, the network allows the ULTRIX system to be used as a development system, a debugging host, and a provider of non-realtime services in a final system.

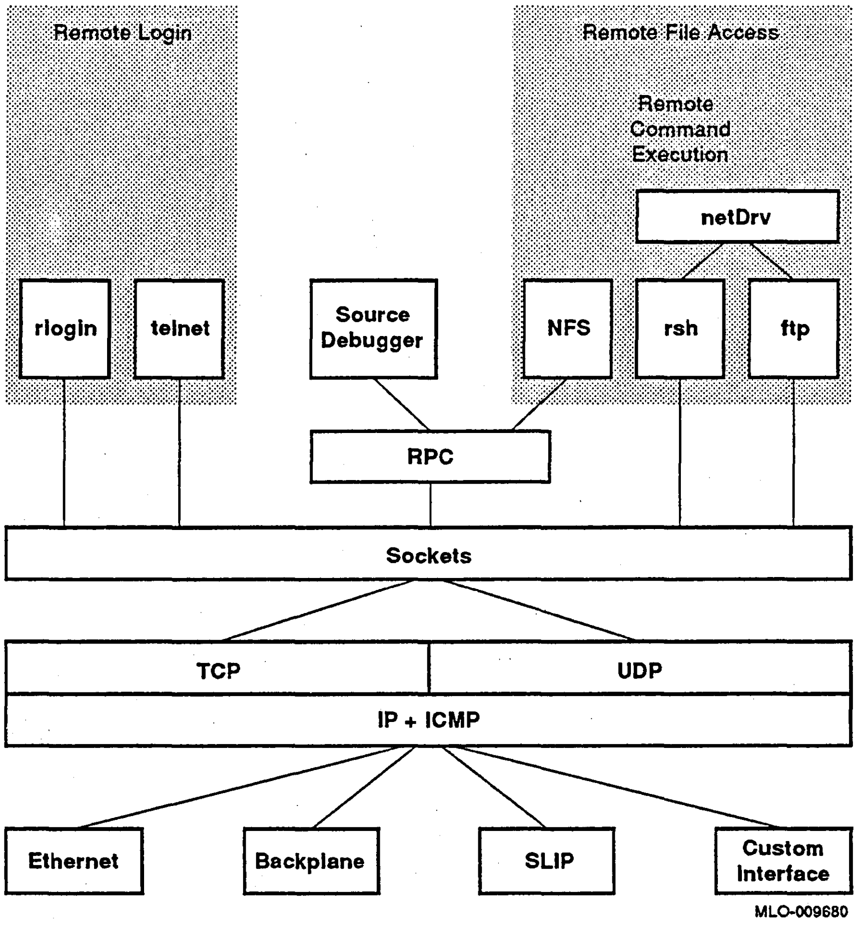
The DECelx network connects DECelx systems with other UNIX or DECelx systems over Ethernet, backplane bus, or serial line interconnections. DECelx uses the TCP/IP network protocols as implemented in BSD 4.3 for all network communications.

The hierarchy of DECelx network components is shown in Figure 3–1. At the lowest level, a DECelx system typically uses Ethernet as the basic transmission medium. A DECelx system can also use serial lines for long-distance connections or shared memory on a common backplane in more closely coupled environments. On top of the transmission media, the Internet protocols TCP/IP and UDP/IP are used to transport data between processes running under either a DECelx or UNIX system.

Using Internet protocols, the DECelx software makes several types of network facilities available to the user:

- Sockets. Allow communications between tasks, running either under a DECelx or UNIX system.
- Remote Procedure Calls. Allow a task on one machine to invoke procedures that actually run on other machines. Both the calling task and the called procedure may run under either a DECelx or UNIX system.
- Remote Login. Allows remote access to the DECelx shell from a UNIX system, and remote access to a UNIX shell from a DECelx system, using either **rlogin** or **telnet** protocols.
- Remote File Access. Allows DECelx tasks to access UNIX files remotely, via the Network File System (NFS), UNIX remote shell (**rsh**), or Internet File Transfer Protocol (**ftp**).
- Remote Command Execution. Allows DECelx tasks to invoke shell commands on a UNIX system over the network using the UNIX remote shell **rsh**.

Figure 3–1 DECelx Network Components



3.3.1 Sockets

DECelx software offers standard UNIX socket calls, which allow realtime DECelx processes and other processes, such as UNIX processes, to communicate in any combination with each other over the network. DECelx socket calls are source compatible with UNIX BSD 4.3.

Any process can open one or more sockets, to which other sockets may be connected. Data written to one socket of a connected pair may be read from the other socket. The network link is transparent in communications. In fact, the processes do not need to know whether they are communicating with other processes on the same CPU or another CPU, or with DECelx or UNIX processes.

3.3.2 Remote Procedure Calls (RPC)

Remote Procedure Call (RPC) is a facility that allows a process on one machine to call a procedure that is executed by another process on another machine. Using RPC, a DECelx task or UNIX process can invoke routines that are executed on other DECelx or UNIX machines, in any combination. RPC documentation is available in the public domain.

3.3.3 Remote Login: rlogin, telnet

The remote login feature allows users to log into DECelx or UNIX machines from any other DECelx or UNIX machine on the network. This is convenient from the programmer's point of view. For instance, on a UNIX workstation, the programmer can open an **rlogin** window that communicates with the DECelx shell. By opening such windows, programmers can monitor and control realtime DECelx systems right from their desks.

DECelx machines can also be accessed via **telnet**, for systems that do not have **rlogin**.

3.3.4 Remote File Access: NFS, ftp, rsh

Remote file access across the network is available. A program running on a DECelx system can use a UNIX system as a virtual file system. Files on any UNIX system may be accessed, via the network, as if they were local to the DECelx system. A program running under DECelx does not need to know where that file is, or how to access it. For example, `/dk/file` might be a file local to the DECelx system, and `host:file` might be a file located on another machine entirely.

DECelx software includes the industry standard Network File System (NFS). It runs as an NFS client with any other system that runs an NFS server. Alternatively, a DECelx system can use either of two older protocols to provide transparent remote file access: **rsh** or **ftp**. An **ftp** server furnishes remote access to a DECelx system from other DECelx or UNIX systems using **ftp**.

3.3.5 Remote Command Execution

The DECelx remote command execution facilities allow programs running on a DECelx system to invoke UNIX commands and have the results returned on standard out and standard error via socket connections. This is accomplished using the UNIX remote shell protocol, which is serviced by the remote shell daemon on UNIX.

3.4 Module Loader and System Symbol Table

The DECelx realtime operating system is unusual in that its facilities are made available to application programs simply as an extensive set of C subroutines. DECelx does not require you to use system traps to get to system functions. Instead, DECelx supplies a system symbol table and a loader with runtime linking to give dynamic and even interactive access to all loaded modules.

The DECelx module loader can load object modules, over the network or from a disk, and relocate them anywhere in memory. The loader also uses the symbol table contained in every object module to build a systemwide symbol table of loaded function and variable names. Names from both system and application modules are added to the system symbol table.

This symbol table is a key DECelx development tool. First, the loader itself uses the system symbol table to resolve undefined references in modules being loaded, dynamically linking newly loaded modules to previously loaded modules. DECelx uses the system symbol table to supply interactive access to all system and application modules that have been loaded. Finally, DECelx debugging facilities use the system symbol table to furnish symbolic references wherever possible.

Runtime linking makes it easy to share subroutine libraries. A single copy of a set of subroutines can be used by several tasks, rather than requiring each task to be linked with a separate copy of needed subroutines. Because there is no inherent distinction between DECelx system modules and user application modules, the system facilities are easy to access, modify, and extend.

3.5 Shell

DECelx Realtime Tools include an interactive program called the shell, which allows developers to interact with all DECelx facilities. The DECelx shell provides one simple but powerful capability: it can interpret and execute almost all C-language expressions, including calls to functions and references to variables whose names are found in the system symbol table.

Thus, the shell can call DECelx system functions or any application functions, examine and set application variables, create new variables, and serve as a general purpose calculator with all C operators.

In addition, the shell includes a command history facility and command-line editing similar to the **vi** editor. The shell can also be used to log into a remote UNIX or DECelx machine with the **rlogin** or **telnet** command.

3.6 Debugging Facilities

DECelx Realtime Tools furnishe three debuggers: a basic symbolic debugger for all targets; and ElxGDB for 680*n*0 and R3000, depending on your target environment. ElxGDB is a powerful, remote source-level debugger.

The DECelx debug facilities include:

- Routines to display system and task status
- A symbolic disassembler that can disassemble any loaded module
- A symbolic C-subroutine traceback facility that can be called at any time to list the current sequence of nested subroutine calls of any task (Motorola 680*n*0 target-specific)
- Trapping of hardware exceptions in a non-fatal way that allows symbolic debugging to continue
- A breakpoint and single-stepping facility that can be applied to specific tasks, even in shared code

These facilities use the system symbol table to provide symbolic references wherever possible.

ElxGDB enables developers to spawn and debug tasks running on networked DECelx targets. Already-running tasks spawned from the DECelx shell can also be debugged. While using ElxGDB, users can continue to take advantage of DECelx native development tools. By combining the DECelx shell, symbolic debugging and disassembly, and performance monitoring facilities with ElxGDB capabilities, developers will have a comprehensive high-level debugging solution.

3.7 Performance Evaluation

To understand and optimize the performance of a realtime system, you must often time various tasks that the system performs. DECelx Realtime Tools offer various timing facilities to help with this task.

The DECelx execution timer can time any C subroutine, or group of subroutines. If the system clock alone cannot provide the resolution necessary to time especially fast functions, the timer routine can repeatedly call a function until the time of a single iteration can be determined to a reasonable tolerance.

DECelx also provides a utility using an auxiliary clock that shows, for each task, the amount of CPU time utilized, the amount of time spent at interrupt level, and the amount of idle time. Time is displayed in ticks and in percentage.

3.8 I/O System

The DECelx I/O system offers uniform device-independent access to many kinds of devices. The user can call seven basic I/O functions: `creat`, `delete`, `open`, `close`, `read`, `write`, and `ioctl`. Higher-level I/O functions, such as UNIX `printf` and `scanf` routines, are supplied and built on these basic functions.

The DECelx software also furnishes a **stdio** buffered I/O package that includes UNIX routines such as `fopen`, `fclose`, `fread`, `fwrite`, `getch`, `putch`, and so forth. These routines increase I/O performance in many cases.

DECelx includes device drivers for serial communications lines, disks, RAM disks, intertask communication devices called pipes, and devices on a network. Application developers can easily write additional drivers, if needed. DECelx allows dynamic installation and removal of drivers without rebooting the system.

Internally, the DECelx I/O system is fast and flexible, allowing individual drivers complete control over how the user requests are serviced. Drivers can easily implement different protocols, unique device-specific functions, and even different file systems, without interference from the I/O system itself. DECelx Realtime Tools also supply several high-level packages that make it easy for drivers to implement common device protocols and file systems.

3.9 Local File Systems

DECelx includes different local file systems for use with block devices (disks). These devices all use a standard interface so that file systems can be freely mixed with device drivers. DECelx I/O architecture makes it possible to have several different file systems, even on a single DECelx system at the same time.

3.9.1 DOS File System

DECelx offers a file system compatible with DOS for personal computers. DECelx DOS is compatible with versions of MS–DOS up to and including Version 4.0. DECelx DOS capabilities offer considerable flexibility appropriate to the varying demands of realtime applications. Major features include:

- A hierarchical arrangement of files and directories, allowing efficient organization and permitting an indefinite number of files to be created on a volume.
- A choice of file fragmentation or contiguity on a per-file basis. File fragmentation results in more efficient use of available disk space while contiguity offers enhanced performance.

- Compatibility with widely available storage and retrieval media. Disks created with DECelx DOS and on DOS personal computers may be freely interchanged.

3.9.2 RT–11 File System

DECelx is supplied with a file system compatible with that of the RT–11 operating system. This file system is appropriate for many realtime file systems, since all files are contiguous. File accesses require exactly one disk access, and sequential file accesses involve minimal disk movement.

The RT–11 file system does have some drawbacks, however. It lacks a hierarchical file organization that is particularly useful on large disks. Also, the contiguous allocation scheme may result in fragmented disk space.

The DECelx implementation of the RT–11 file system includes byte-addressable random access (seeking) to all files. Each open file has a block buffer for optimized reading and writing.

3.9.3 Raw Disk File System

DECelx offers a simple file system for use with disk devices. This file system treats the entire disk like a single large file. Portions of the disk can be read and written, specified by byte offset, and simple buffering is performed. The file system offers the advantages of size of transfer and speed when only simple, low-level disk I/O is required.

3.9.4 Alternative File Systems

In DECelx, the file system is not tied to the device or its driver. A device may be associated with any file system. Alternative, user-supplied file systems can be written and used by drivers in the same way, by following the same standard interfaces between the file system, the driver, and the DECelx I/O system.

3.10 Utility Libraries

DECelx supplies many general utility subroutines to application developers. These routines are organized as a set of subroutine libraries, which are described next. Application developers are encouraged to use these libraries wherever possible to reduce both development time and memory requirements for the application.

Interrupt Handling Support	DECelx furnishes routines for handling hardware interrupts and software traps without having to resort to assembly language coding. Routines are provided to connect C routines to hardware interrupt vectors and to manipulate the processor interrupt level.
Watchdog Timers	A watchdog facility allows callers to schedule execution of their own routines after specified time delays. As soon as the specified number of ticks have elapsed, the specified timeout routine will be called at the interrupt level of the system clock, unless the watchdog is canceled first. Note that this mechanism is entirely different from the kernel's task delay facility.
Message Logging	A simple message logging facility allows error or status messages to be sent to a logging task, which then formats and outputs the messages to a systemwide logging device, such as the system console, disk, or accessible memory. The message logging facility can be used from interrupt level or task level.
Memory Allocation	DECelx supplies a memory management facility (source-compatible with UNIX) useful for dynamically allocating, freeing, and reallocating blocks of memory from a memory pool. The size of the pool can be set by the user. Blocks of arbitrary size can be allocated. This memory scheme is built on a much more general mechanism that allows DECelx to manage several separate memory pools.
String Formatting and Scanning	DECelx includes string formatting and scanning subroutines compatible with UNIX that implement `printf`/`scanf` format-driven encoding and decoding.
Linear and Ring Buffer Manipulations	DECelx furnishes buffer manipulation functions such as copying, filling, comparing, and so on, that are optimized for speed. Also provided is a set of general ring buffer routines that manage first-in first-out circular buffers. Additionally, these ring buffers allow a single writer and a single reader to access a ring buffer simultaneously, without being required to interlock their accesses explicitly.
Linked-List Manipulations	DECelx supplies a complete set of routines for creating and manipulating doubly-linked lists.

3.11 Board Support Packages

Board Support Packages (BSP) for each board include two target-specific libraries, **sysLib** and **sysALib**. These libraries provide an identical software interface to the hardware functions of all boards. They include routines for hardware initialization, interrupt handling and generation, hardware clock and timer management, mapping of local and bus memory spaces, memory sizing, and so on.

Part II
VAXELN Toolkit

Part II, contains the following chapters:

- Chapter 4, VAXELN Toolkit Overview, introduces the VAXELN Toolkit components and realtime programming features.
- Chapter 5, VAXELN Programming Concepts, discusses VAXELN processes and jobs and the concept of concurrency.
- Chapter 6, VAXELN Toolkit Components, describes VAXELN system development software, runtime software, and utilities including support for POSIX 1003.1, P1003.4/D11 and P1003.4a/D4
- Chapter 7, VAXELN DECwindows, outlines the basic DECwindows architecture and user environment.
- Chapter 8, VAXELN Window Server, describes the VAXELN Window Server (EWS), VAXELN Window Station, and DECwindows server software that runs on Digital workstations and X Window terminals.

4 VAXELN Toolkit Overview

The VAXELN Toolkit is a VMS layered product that provides powerful software for developing dedicated, realtime software applications that run on VAX processors. Supported platforms range from the low-end rtVAX 300 embeddable daughterboard to high-end rtVAX 9000 systems. A VAXELN application relies on the VAX processor and the VAXELN kernel to provide prompt, predictable responses to time-critical events. The VAXELN Toolkit caters to the memory and speed requirements of a dedicated, realtime application by allowing you to create tailored systems that include only the services, drivers, and utilities required by the application.

You develop a VAXELN application on a VMS development system using VMS and VAXELN development tools. With these tools, you create a bootable VAXELN system image, which incorporates user and Digital program images. You load the system image into the memory of a supported VAX target processor, where it runs independently of the development system.

This chapter presents the following topics:

- Survey of VAXELN Toolkit Components, Section 4.1
- VAXELN Toolkit Realtime Programming Features, Section 4.2
- VAXELN Hardware and Software Requirements, Section 4.3

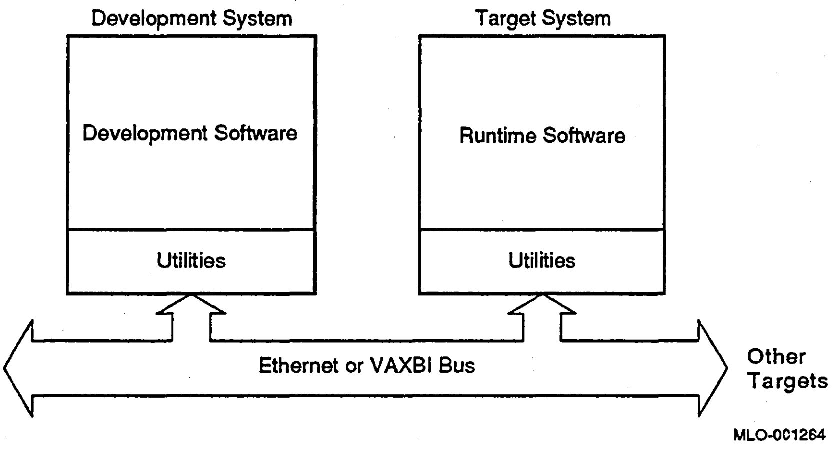
4.1 Survey of VAXELN Toolkit Components

The VAXELN Toolkit includes:

- System development software that runs on a VMS development system
- Runtime software that executes as part of an application on a supported VAX target system
- VAXELN utilities that run on the VMS development system, the VAXELN target system, or both

Figure 4–1 shows the relationship between the development and target systems.

Figure 4–1 Development System/Target System Relationship



You develop a VAXELN application on a VMS development system by using a comprehensive set of VMS CASE and VAXELN development tools. The resulting VAXELN system image contains user and toolkit program images, and can be booted on the VAX target system from disk, tape, or read-only memory (ROM). If you have a DECnet–VAX license and the appropriate Ethernet hardware, you can down-line load the system image from a VMS, ULTRIX, or VAXELN system (serving as boot host) to the target processors.

See Chapter 6 for details about the VAXELN Toolkit components.

4.2 VAXELN Toolkit Realtime Programming Features

Realtime application programmers formerly were required to be proficient in hardware architecture, operating systems, and assembly language programming, as well as realtime system design and high-level language programming. However, the VAXELN Toolkit eliminates the intricate, low-level knowledge requirements of realtime programming. Toolkit features, such as the following, free you to focus on system design and high-level language programming:

- **Small, optimized, realtime kernel executive.** The kernel controls the sharing of processor resources and employs a preemptive priority-based scheduler for supporting concurrent processing. Each of the supplied kernel images is customized for the system it supports.

- **Tailorability.** A VAXELN system includes only those services, drivers, and utilities that the application design requires; it does not serve as a general-purpose operating system. Omitting unnecessary services, drivers, and utilities reduces system size and overhead.
- **Predictability.** VAXELN systems are predictable in part because they are memory resident and do not incur the overhead associated with general-purpose, time-sharing systems. The VAXELN kernel's preemptive, priority-based scheduling also helps ensure predictable, timely response to events and interrupts.
- **Distributed processing.** Transparent local area network (LAN), shared CPU bus, and VAXBI bus support let you distribute application programs among multiple processors. By distributing an application's less time-critical functions to other processors, time-critical tasks can make better use of processor time, while maintaining communication with all application functions.
- **Expandability.** When an application requires more speed or computing power, you can move programs to a faster VAX processor or distribute the system's programs among processors in a multiprocessing configuration with little or no code modification.
- **DECnet end-node support.** VAXELN provides full DECnet end-node support, which lets a VAXELN system communicate over the Ethernet with other systems (including VAXELN and VMS) running on other DECnet–VAX nodes.
- **Internet network support.** Using an Ethernet network interface, a VAXELN application can communicate with other applications in an Internet network. The Internet interface supports the Internet Protocol (IP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Address Resolution Protocol (ARP), Reverse Address Resolution Protocol (RARP), and Boot Protocol (BOOTP).

  In addition, a File Transfer Protocol (FTP) client interface is available for transferring files from one host to another host in a heterogeneous operating system environment.
- **VMS file system compatibility.** VAXELN applications that use storage media can take advantage of a file service that supports file-oriented disk and tape I/O operations and remote file access. The file service maintains VMS file compatibility by using the VMS on-disk file structure (ODS II) and the record management services (RMS) file format.

- **Interoperability.** VAXELN applications can communicate with and gain access to files on other operating systems, such as the VMS, ULTRIX, and MS–DOS operating systems.
- **VAXELN POSIX functions.** VAXELN POSIX functions provide the capability of developing and running applications in a POSIX environment.
- **KAV30 support.** The VAXELN Toolkit provides the capability of developing and running applications on the KAV30.
- **High-level language programming.** You can code VAXELN applications entirely in structured, modular, high-level languages (VAX Ada, VAX C, VAX FORTRAN, and VAXELN Pascal), without sacrificing performance. Thus application code is easier to write, read, maintain, and translate. Language constructs that implement VAXELN architectural features are either predefined in the compilers or provided as callable kernel or runtime library utility routines. By using a high-level language to program device drivers and interrupt service routines (ISRs), you eliminate the need for assembly language programming.
- **Interactive debugger.** The debugger offers two operating environments. A remote debugger environment supports symbolic debugging and lets you debug your target application from a development system user terminal across an Ethernet DECnet connection. If your application does not include DECnet support, you can include a local debugger environment in your VAXELN system and debug the application from the target system's console.
- **Development, command language, and network utilities.** VAXELN utilities can be included in your system image to help you debug and fine-tune your application, and issue interactive commands to the VAXELN system.

  You can use these utilities to perform LAT and SET HOST network functions. These utilities include the VAXELN Performance Utility (EPC and EPA), VAXELN Display Utility (EDISPLAY), error logging tools, VAXELN Command Language Utility (ECL), LAT Control Program (LATCP), Outbound Remote Terminal Utility (SET_HOST), and TELNET server.
- **DECwindows support.** The Toolkit's DECwindows support allows you to create network-transparent distributed applications that perform drawing and windowing operations. The DECwindows components include a server, runtime libraries and tools, a Window Manager, and terminal emulators.

4.3 VAXELN Hardware and Software Requirements

This section identifies VAXELN system development and runtime hardware and software requirements. The *VAXELN System Support Addendum (SSA)* includes a list of the supported hardware and minimum software requirements.

VAXELN Hardware Requirements

You can run the VAXELN Toolkit system development software on any VAX, MicroVAX, or VAXstation processor. If the VMS development system includes Ethernet interfaces and the appropriate interconnecting hardware, you can downline load your VAXELN system image to your VAX target configuration. The Ethernet hardware also provides the necessary communication link for applications that require remote debugging, remote error logging, performance data collection, or remote command-language sessions.

A VAXELN target configuration must include a target processor that the VAXELN Toolkit supports. The toolkit supports target processors ranging from the rtVAX 300 daughterboard to rtVAX 9000 systems. The target processors can be standalone systems or Ethernet-connected systems distributed on a LAN. For a list of supported target processors, see the latest *VAXELN Toolkit System Support Addendum (SSA)*. You can obtain a copy of the *SSA* from your Digital sales representative.

Application-specific target hardware requirements include Ethernet hardware and peripheral devices. If your VAXELN application requires downline loading, remote debugging, remote error logging, Performance Utility support, or Remote Terminal Utility support, the target configuration must include the appropriate Ethernet hardware. Application-specific peripheral devices supplied by Digital, or a third party may also be required.

VAXELN Software Requirements

To develop a VAXELN system, the following software is required:

- VMS Version 5.0 or higher
- DECnet–VAX Version 4.6 or higher
- One of the following high-level languages:

  VAXELN Pascal (included in the VAXELN Toolkit)
  VAX Ada, Version 1.4 or higher *and* VAXELN Ada Version 2.0 or higher
  VAX C, Version 2.4 or higher
  VAX FORTRAN Version 4.4 or higher

Optional VAXELN Development Software

Other software products can be used to augment the VAXELN development environment provided by VMS CASE tools and the VAXELN Toolkit:

- DECnet–VAX
- Additional programming language compilers:
  - VAX Ada
  - VAX C
  - VAX FORTRAN
- VAXELN Ada Toolkit (to supplement the VAX Ada compiler)
- DECmessageQ for VAXELN
- DEC Realtime Integrator for VAXELN

5 VAXELN Programming Concepts

A VAXELN application's design and development are based on the concept of concurrency, the logically simultaneous execution of multiple programs and parts of programs. Concurrency is a proven approach for applications that require programs to work together efficiently and quickly to solve a specific problem.

VAXELN programs execute as independently scheduled jobs. A typical VAXELN application consists of multiple jobs, each with functionally independent components called processes.

This chapter discusses the following VAXELN programming concepts:

- Processes, Section 5.1
- Jobs, Section 5.2
- Concurrency, Section 5.3

5.1 Processes: Program Execution Threads

A VAXELN process is an independent thread of program execution, essentially similar to a VAX process as defined by the VAX architecture. Associated with a process are hardware and software context, program code, and an address space. Each process in a VAXELN system represents a specialized task. The main section of program code (the innermost procedure block for Ada programs, the main routine for C programs, the main program for FORTRAN programs, and the program block for VAXELN Pascal programs) executes as the master process. The kernel creates this process automatically when the program starts executing.

5.2 Jobs: Process Families

The processes associated with a running program collectively constitute a job. A job consists of a master process and zero or more subprocesses. A job also represents a single activation of a program, with all processes in the job sharing access to the program's executable code.

You can think of a job as a family of processes. A job's master process and subprocesses create other subprocesses dynamically. Once created, a process stays active until one of the following conditions occurs:

- The process exits.
- Another process deletes the process.
- Its master process terminates.
- It encounters an error from which it cannot recover.
- It finishes executing the associated code segment.

The exit operation provides the most controlled means of forcing process termination.

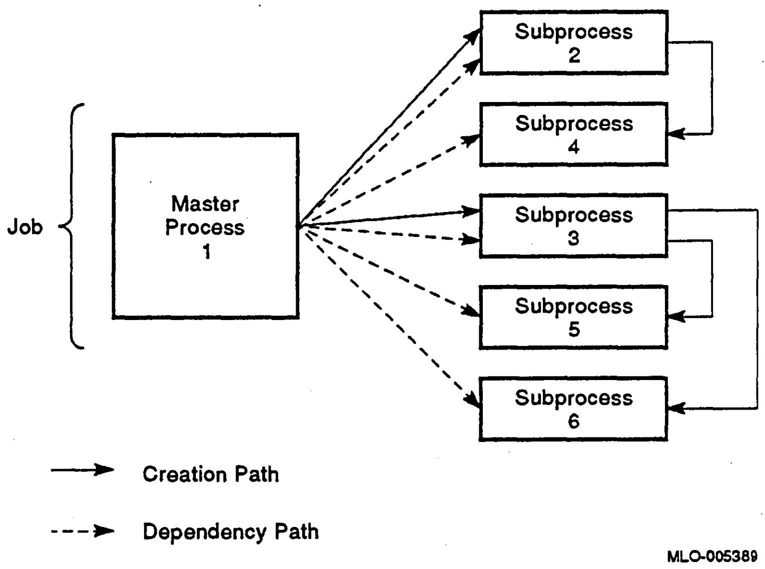
Figure 5–1 illustrates the creation and dependency paths for a job consisting of a master process and five subprocesses. The creation path consists of two levels. The master process creates two subprocesses: Subprocess 2 and Subprocess 3. Subprocesses 2 and 3 subsequently create one and two subprocesses, respectively.

The dependency path, however, has one level. All subprocesses that the master process or any of its subprocesses spawn depend on the master process. When a master process terminates under any circumstances, the kernel removes the corresponding job, its master process and associated subprocesses, and shared data from the system. The kernel returns the job's memory resources to the system.

5.3 Concurrency: Processes Sharing Processor Resources

To take advantage of a VAXELN realtime system's efficiency, you design applications with the concept of concurrency in mind. Concurrency is built into VAXELN so that cooperating processes can share resources. While some processes wait to execute (wait for an event to occur or a resource to become available), others can execute. The kernel manages system resources such that

Figure 5–1 Process Family



all jobs and processes appear to execute simultaneously, although only one process actually executes on a processor at a time.

You determine whether jobs and processes should execute concurrently when you design your application. Concurrent programming has numerous system design advantages, including improved performance.

The VAXELN kernel supports three levels of concurrency: multitasking, multiprogramming, and multiprocessing.

5.3.1 Multitasking

Multitasking lets you divide an application program's functionality into a set of smaller, focused tasks that can execute concurrently. Each task executes as a separate dedicated process. For example, a program controlling a wing in a flight simulation application might consist of processes that specialize in tasks such as surface control and engine fire-up.

5.3.2 Multiprogramming

Multiprogramming is the concurrent execution of entire programs, including multitasking programs. The programs execute as jobs that may or may not coordinate their execution; Job A may or may not depend on Job B. However, in most VAXELN systems, jobs work together to accomplish mutual goals. For example, in a flight simulation application, a collection of cooperating jobs might emulate major components of an airplane, such as cockpit controls and instrumentation, navigation equipment, left and right wings, and tail structure.

5.3.3 Multiprocessing

A VAXELN application's jobs can reside on one processor or they can be distributed among multiple processors. The concurrent execution of a VAXELN application's programs on multiple processors is called multiprocessing. The VAXELN kernel supports loosely, tightly, and closely coupled multiprocessing configurations.

In a loosely coupled multiprocessing configuration, connections to a common Ethernet link the processors, as shown in Figure 5–2. Each processor runs its own VAXELN system image with a separate collection of jobs.

Figure 5–2 Loosely Coupled Multiprocessing Configuration

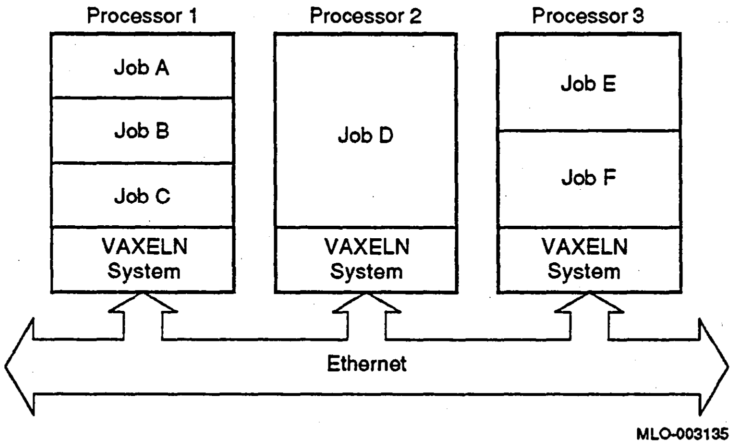
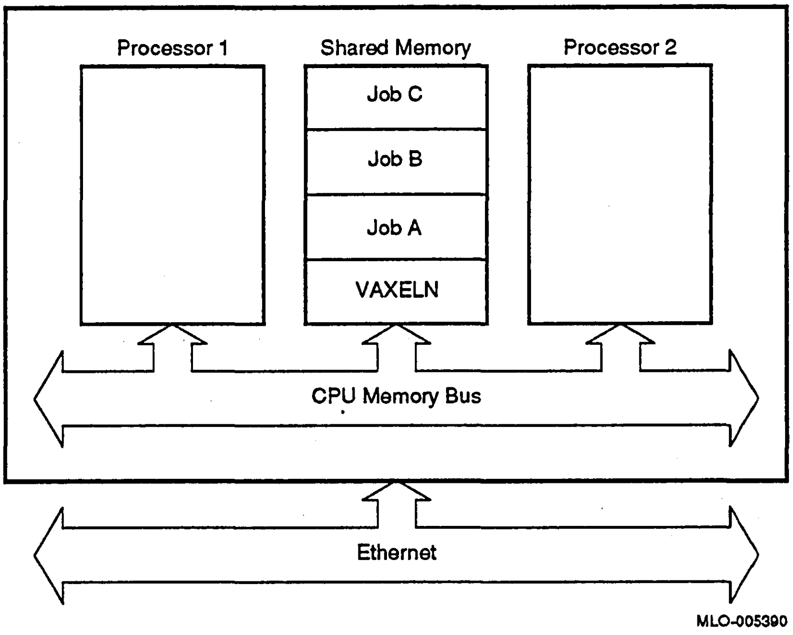


Figure 5–3 Tightly Coupled Symmetric Multiprocessing Configuration



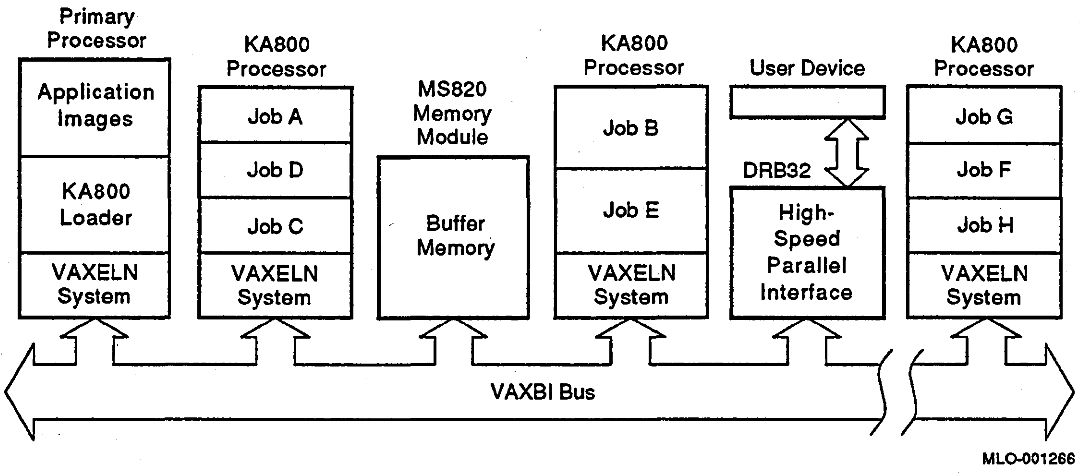
In a tightly coupled configuration, VAXELN supports multiple symmetric processors on the same processor bus, as shown in Figure 5–3.

VAXELN supports tightly coupled symmetric multiprocessing on VAX 6000 series and VAX 8800 multiprocessor configurations. All processors share a single copy of a VAXELN system image and thus share the same kernel, runtime components, and program images. A job can be eligible to execute on any processor (the default), or you can limit it to a specific subset of processors.

A closely coupled multiprocessing configuration consists of a VAX primary system and one or more KA800 VAX processor boards. The KA800 boards operate as secondary processors, for example, as I/O processors (IOPs) to offload I/O handling duties from the primary system. The primary system can be a single processor or a tightly coupled multiprocessor, such as a VAX 6000 multiprocessor. Each KA800 system is connected to the primary system's VAXBI bus and runs its own VAXELN system image, with its own kernel, runtime components, and program images.

As shown in Figures 5–4 and 5–5, the primary system in a closely coupled environment can run a VAXELN or VMS system. A VAXELN primary system can run on a VAX 6000 series, 8500, 8530, 8550, 8700, 8800, 8810, or 8820–N processor. When an application uses a VAXELN primary system, you downline load VAXELN systems into the KA800 processors by using a configuration file, a runtime procedure call, or an ECL command.

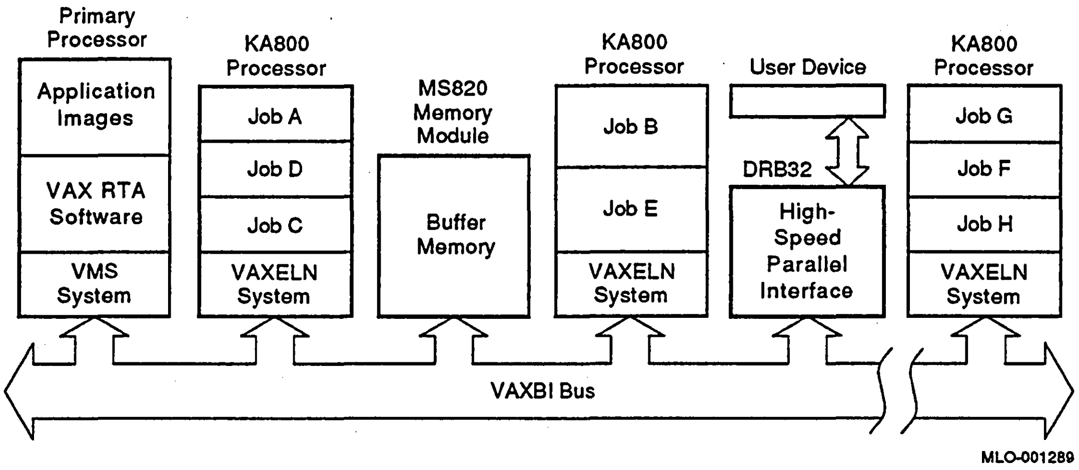
Figure 5–4 Closely Coupled Multiprocessing Configuration with VAXELN Primary System



A VMS primary system can run on any VAX processor that supports VAX Realtime Accelerator (RTA) software.

A common application for closely coupled multiprocessing is the distribution of realtime I/O functions. You can achieve superior performance by offloading interrupt-intensive tasks to KA800 systems, thus freeing the primary system for other functions. The KA800 systems can control the DRB32 direct memory access (DMA) parallel port device directly to distribute I/O control for high-speed data transfers and fast, predictable interrupt response time.

Figure 5–5 Closely Coupled Multiprocessing Configuration with VMS Primary System



Data-sharing between primary and secondary processors in closely coupled configurations is limited to sharing of primary system memory. Data in the primary system's memory can be accessed by the attached KA800 systems. However, the primary system cannot gain access to data in the memory of the KA800 systems.

6 VAXELN Toolkit Components

You develop a VAXELN application on a VMS development system using a comprehensive set of VMS CASE and VAXELN development tools. This chapter discusses the following VAXELN Toolkit components:

- VAXELN System Development Software, Section 6.1
- VAXELN Runtime Software, Section 6.2
- VAXELN Utilities, Section 6.3

6.1 VAXELN System Development Software

The VAXELN development software runs under the VMS operating system. You use this software and VMS CASE tools to prepare your VAXELN system on the development system. The VAXELN Toolkit's system development components include the following:

- VAXELN Pascal compiler, which generates object modules from VAXELN Pascal source files, Section 6.1.1
- VAXELN_SERVICES Ada package, VAXELN C, FORTRAN, and Pascal runtime libraries, which contain source modules that support realtime, I/O, math, KAV30, DECwindows, and other routines called from VAX Ada, VAX C, VAX FORTRAN, and VAXELN Pascal programs, Section 6.1.2
- VAXELN System Builder, which creates VAXELN system images by combining application program images with the VAXELN kernel and optional toolkit components, Section 6.1.3

If you choose to program your application in Ada, C, or FORTRAN, you need to use the VAX Ada, VAX C, or VAX FORTRAN compiler. Like the VAXELN Pascal compiler, these compilers generate object modules from your source files.

If you use Ada, you also need the VAXELN Ada runtime libraries, which are available as part of the VAXELN Ada Toolkit. This toolkit packages the runtime libraries with a remote debugger for developing VAXELN Ada applications. For more information about the VAXELN Ada Toolkit, see Chapter 14.

To combine your program object modules and shareable images into program images that you can include in your VAXELN system, you use the VMS Linker. Section 9.4.3 provides more information about the linker.

Once you create the image, you can downline load or boot the image onto a target processor configuration. Section 6.1.4 discusses the ways you can load and boot a system image.

6.1.1 VAXELN Pascal Compiler

The VAXELN Pascal compiler is an extended version of the ANSI/IEEE770X3.97–1983 Pascal compiler that supports data types and operations for concurrent programming. The optimizing VAXELN Pascal compiler generates position-independent code and features realtime programming extensions and system programming enhancements. The following list identifies the realtime programming extensions:

- **Process blocks.** You can declare blocks of code with the reserved word PROCESS_BLOCK. Process blocks are similar to procedures but execute as concurrent subprocesses.
- **Kernel support.** You have direct access to kernel objects. You can declare kernel data using the predeclared data types AREA, DEVICE, EVENT, MESSAGE, NAME, PORT, PROCESS, and SEMAPHORE. You manipulate kernel objects using predeclared kernel routines. The compiler applies type-checking to the data types and the kernel routine arguments.
- **Mutex and area-lock variable support.** The MUTEX and AREA_LOCK_VARIABLE data types and associated routines are provided as library declarations.
- **Queue support.** You can use the predeclared data type QUEUE_ENTRY and associated routines for representing and using absolute queues.
- **Interrupt service routine (ISR) declarations.** You can declare ISRs with the reserved word INTERRUPT_SERVICE. These constructs are similar to procedures, but they are called by the kernel to handle device interrupts.

System programming enhancements include the following:

- **Flexible data types.** You can use the predeclared data types STRING(*n*), VARYING_STRING(*n*), and BYTE_DATA(*n*) to represent fixed-length strings, variable-length strings, and series of bytes. You can also declare your own flexible data types.
- **Flexible parameter lists.** You can use variable-length and nonpositional parameter lists.
- **Typecasting.** You can typecast variable references to access data using a data type other than that declared for the data.
- **Procedure and function type declarations.** You can declare procedure and function types to categorize constructs that have arguments of the same data types.
- **Modular compilation.** You can compile large programs in separate smaller modules and still maintain type-checking.

6.1.2 VAXELN_SERVICES Ada and Runtime Libraries

The VAXELN Toolkit includes the VAXELN_SERVICES Ada package and runtime libraries for languages, VAXELN and VAXELN POSIX, and support for the KAV30 kernel. The VAXELN Toolkit includes the following:

- **VAXELN_SERVICES Ada package.** This package provides the types, operations, constants, and so forth for calling the VAXELN kernel and utility routines from Ada programs.
- **VAXELN C Runtime Library.** This library is a compatible subset of the VAX C runtime environment. Many VAX C programs originally written for VMS or UNIX can run in a VAXELN system with only minor modifications. However, the VAXELN C Runtime Library does not support all VAX C or UNIX emulation functions.
- **VAXELN FORTRAN Runtime Library.** This library provides an extensive subset of the VAX FORTRAN runtime environment. In VAXELN systems, VAX FORTRAN is used primarily for computation and is limited to sequential or direct access to sequential files. Most VAX FORTRAN programs that you migrate to VAXELN systems require minimal change. Some features are restricted or unavailable in the VAXELN environment, such as ISAM or relative file access, VMS logical names, QIO and RMS service calls, and some VMS services.

- **VAXELN Pascal Runtime Library.** This library extends the VAXELN Pascal programming language to include modules for runtime routines that have local read/write data, define entry points for kernel routines, and contain modules that support the VAXELN system service and device driver interfaces.
- **VAXELN POSIX Runtime Library.** This library consists of POSIX 1003.1–1990, P1003.4a/D4, and P1003.4/D11 functions and is compatible with the VAX C runtime environment. Many VAX C programs originally written for VMS or UNIX can run in a VAXELN POSIX environment with only minor modifications. However, the VAXELN POSIX Runtime Library does not support all POSIX 1003.1, P1003.4a/D4, P1003.4/D11 functions.
- **KAV30 Runtime Library.** This library contains language bindings, symbols, sharable images, and messages for using the KAV30 as the target processor. You can use the KAV30 kernel from your Ada, C, FORTRAN, or Pascal application,

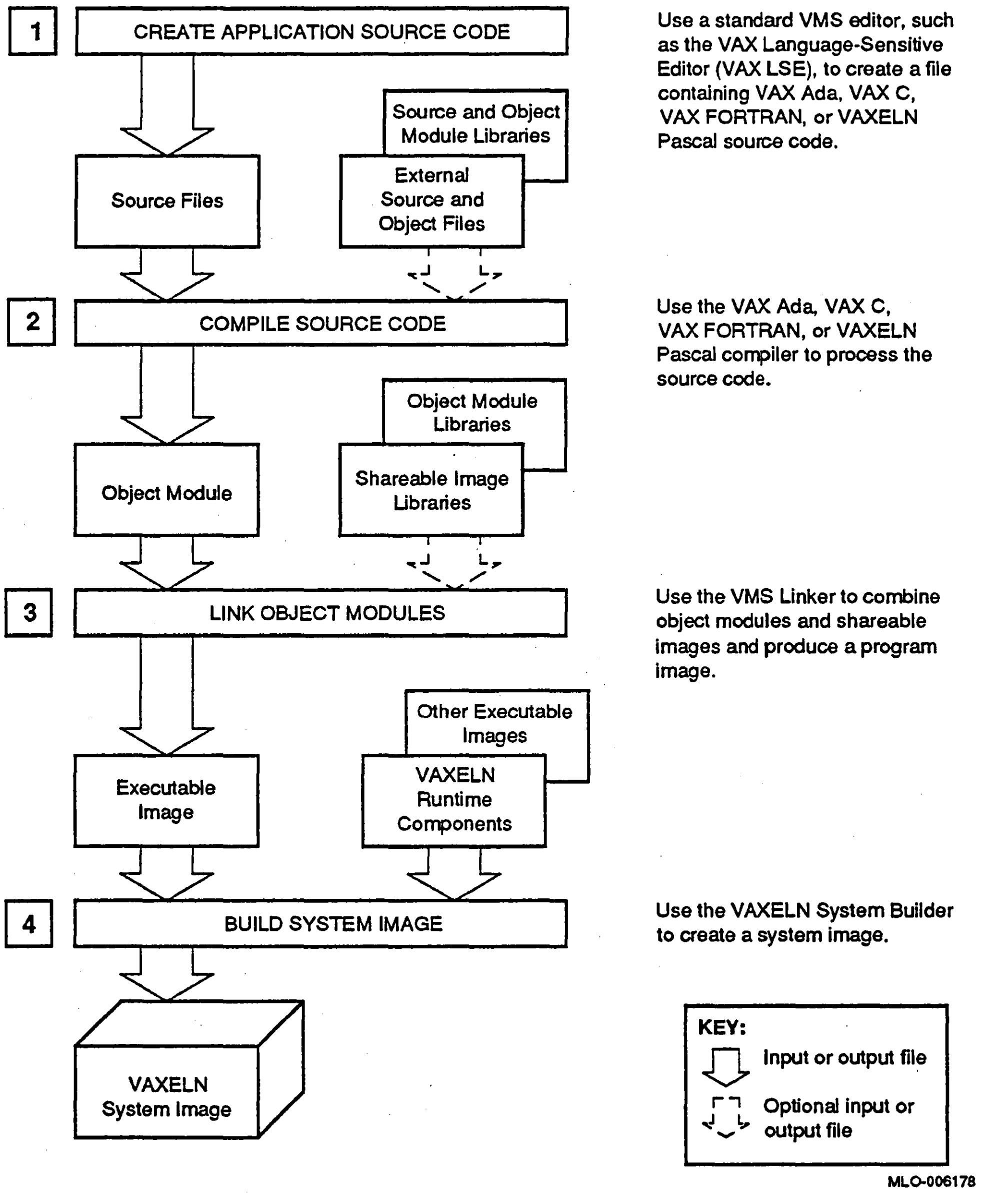
6.1.3 VAXELN System Builder

After you develop your application program images, use the VAXELN System Builder to create a system image. The VAXELN System Builder is a menu-driven facility that generates a bootable system image by combining application program images with the VAXELN kernel, runtime services, and utilities.

You invoke the System Builder from VMS with the DCL command EBUILD. The System Builder executes in an interactive screen-editing mode that lets you modify a series of menus in order to define your system's characteristics. Using the menus, you can:

- Select a target processor type
- Edit systemwide characteristics
- Edit DECnet and Internet network characteristics
- Edit or add descriptions of programs and devices
- Edit console characteristics
- Edit error log characteristics
- Edit DECwindows characteristics
- Build a system image

Figure 6–1 Preparing a VAXELN System Image



Use a standard VMS editor, such as the VAX Language-Sensitive Editor (VAX LSE), to create a file containing VAX Ada, VAX C, VAX FORTRAN, or VAXELN Pascal source code.

Use the VAX Ada, VAX C, VAX FORTRAN, or VAXELN Pascal compiler to process the source code.

Use the VMS Linker to combine object modules and shareable images and produce a program image.

Use the VAXELN System Builder to create a system image.

If you specified your system's characteristics in a previous menu-editing session, you can invoke EBUILD with a qualifier that directs the System Builder to rebuild the system image without reentering the interactive screen-editing mode. You can also specify qualifiers to generate additional information, such as a system map file.

Figure 6–1 illustrates the steps for preparing a VAXELN system image.

6.1.4 Loading and Booting a VAXELN System Image

After you use the System Builder to create your system image, you load or boot the system image onto a target processor with one of the following methods:

- Downline system load
- Disk or tape device
- Read-only memory (ROM)

To downline load an image from a VMS, ULTRIX, or VAXELN system, serving as boot host, you must have a DECnet license and the appropriate Ethernet hardware configured into your system. The VAXELN Toolkit supplies the programs and command procedures necessary for downline loading to each type of target processor.

For those applications that cannot use the downline load feature, the Toolkit supplies a command procedure that copies a system image to a disk, tape, cartridge, or diskette device. Once the image is copied, you can transfer the storage medium to the target processor, then boot the volume.

Additionally, you can program a VAXELN system into ROM. After installing that memory on the target processor, you can load and boot your system.

6.2 VAXELN Runtime Software

A VAXELN system image includes user-application and Digital program images. The user application programs, which you write in high-level languages, can be data acquisition and reduction programs, process control supervisors, and user-written device drivers, to name a few. The program images that Digital supplies include the following:

- The VAXELN Toolkit's highly optimized kernel executives, which control the sharing of the target processor's resources, Section 6.2.1
- The VAXELN POSIX runtime library, which allows you to develop and run portable applications in a POSIX environment, Section 6.2.2

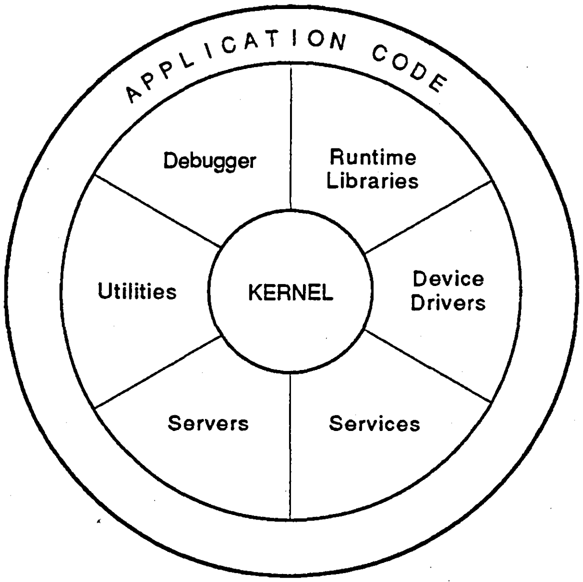
- The VAXELN Toolkit's support of for the KAV30 kernel, which allows you to develop and run realtime applications for the KAV30 using the VAXELN Toolkit, Section 6.2.3
- VAXELN_SERVICES Ada package and runtime libraries, which contain object modules and shareable images that support realtime, KAV30, I/O, math, DECwindows, and other routines called from VAX Ada, VAX C, VAX FORTRAN, and VAXELN Pascal programs, Section 6.2.4
- Device drivers, which control communication between application programs and external devices, Section 6.2.5
- Network communication services, which include a DECnet Service, an Ethernet/IEEE 802 Datagram Service, Internet Services, and Local Area Transport (LAT) Host Services, Section 6.2.6
- An Authorization Service, which maintains a database of a system's authorized users and identifies users that issue network requests, Section 6.2.7
- A File Service, which provides support for file-oriented disk and tape I/O operations and remote file access, Section 6.2.8

In addition to the preceding runtime software, the toolkit includes a DECwindows Server, a Window Manager, and terminal emulators. For information about these runtime components, see Chapter 7.

Using the VAXELN Toolkit's System Builder, you combine the application and Digital program images into a VAXELN system image. When building the system, you can specify the programs that are to start executing as soon as you load or boot the system.

Figure 6–2 shows the system image components. As shown in the diagram, the kernel executive is the heart of a VAXELN system; the kernel schedules and controls an application's execution and access to system resources. The appropriate kernel image is included for your target configuration. The second tier of the diagram represents optional user and Digital software that provides kernel extensions; you tailor your VAXELN system by including only those services and utilities that your application requires. The outermost tier represents a VAXELN system's highest level of code: your application program images.

Figure 6–2 VAXELN System Software



MLO-003133

6.2.1 VAXELN Kernel

The VAXELN kernel is a small, realtime executive that controls target hardware resources and the execution of VAXELN system software. The kernel defines a set of objects that it uses to control the sharing of resources and to synchronize communication between the jobs in a system. The kernel manipulates these objects in response to routine calls issued from application programs. In addition, the kernel maintains system-level and user-level data.

VAXELN applications typically require fast, predictable responses to interrupts. To meet this crucial need, the highly optimized kernel takes advantage of the VAX architecture and imposes minimal overhead between the application code and the hardware.

The following sections outline the data structures that the kernel defines and the operations that it performs.

Data Structures for Realtime Programming

The VAXELN kernel recognizes and operates on a group of realtime programming data structures, which processes use to synchronize and communicate with each other. These structures include a set of kernel objects and two specialized structures called mutexes and area-lock variables. The objects represent ongoing activities, such as process execution, and hardware and software resources, such as devices, memory regions, events, and messages. Mutexes and area lock variables are structures for optimizing kernel object operations. Table 6–1 describes the kernel objects, and Table 6–2 describes the optimized structures.

Table 6–1 Kernel Objects

Object	Description
Area	Represents a region of physical memory accessible to all jobs executing on the same node in a local area network (LAN).
Device	Represents a channel to an I/O device and associates an interrupt service routine (ISR) with the device's interrupt. Device objects synchronize ISR and the execution of device driver processes.
Event	Represents a flag that identifies the occurrence of a realtime event. Events synchronize process execution and access to shared data.
Message	Represents data that is transmitted between processes. Messages can be sent between two processes, two jobs, or two nodes in a LAN.
Name	Represents an entry in a name table that associates a character string name with a message port or process. Port names can be local (known only on their own node) or universal (known on any node in the LAN).
Port	Represents a system-maintained store for messages that are sent or waiting to be received. A program can connect two ports in the same or different jobs to form a circuit, which simplifies and increases the reliability of communication between jobs. Programs can use ports to communicate with any system in a LAN. If a port is connected in a circuit, the processes in the job that creates the port can receive messages from that port. However, any process in any job can send a message to the port.

(continued on next page)

Table 6–1 (Cont.) Kernel Objects

Object	Description
Process	Represents a functionally independent entity that provides the execution context for a program image or part of a program image. The main program executes as a master process, which can control subprocesses. Collectively, a master process and its subprocesses constitute a job.
Semaphore	Represents a synchronization gate that controls access to a shared resource. Binary semaphores enforce exclusive access to a resource. Counting semaphores permit metered access, allowing a specified number of processes simultaneous access to units of a resource.

Table 6–2 Data Structures for Optimizing Area and Binary Semaphore Operations

Structure	Description
Area-lock variable	Represents a variable that resides in an area object for synchronizing job access to the associated area. Using this variable, a process can lock an area to gain exclusive access. When the process locks the area, the process does not have to issue a wait before accessing the associated area unless the area is already locked.
Mutex	Represents a binary semaphore that a process can lock and unlock. A process can lock a mutex to gain exclusive access to a shared resource. When the process locks the mutex, the process does not have to issue a wait before accessing the resource unless the mutex is already locked.

VAXELN Pascal predeclared data types represent the kernel objects. To create and use an object, a program declares a variable of the object's type. The variable then takes on the object's identifying value. To use mutexes and area lock variables, you must include the appropriate definition module.

The VAXELN Toolkit also provides kernel interfaces for VAX Ada, VAX C and VAX FORTRAN programming.

Operations for Manipulating Kernel Data Structures

Each VAXELN kernel data structure is associated with a corresponding set of operations that are implemented as routine calls. The operations let VAXELN systems manipulate the structures and the resources associated with them. To perform the operations, VAXELN Ada, VAX C, VAX FORTRAN, and VAXELN Pascal application programs call kernel routines directly.

Kernel operations are categorized according to three types of services: process synchronization, communication, and device handling. Table 6–3 categorizes the kernel operations and lists the corresponding routines.

Table 6–3 Kernel Operations

Operation	Routines
Process Synchronization	

Create an area, event, job, mutex, process, or semaphore	CREATE AREA
	CREATE AREA EVENT
	CREATE AREA SEMAPHORE
	CREATE EVENT
	CREATE JOB
	CREATE MUTEX
	CREATE PROCESS
	CREATE SEMAPHORE

Initialize an area lock variable	INITIALIZE AREA LOCK
Wait for an area, event, mutex, process, or semaphore to be set to an available state, or wait for a timeout to occur	LOCK AREA LOCK MUTEX WAIT ALL WAIT ANY
Signal an area, event, mutex, process, or semaphore	SIGNAL UNLOCK AREA UNLOCK MUTEX
Delete an area, event, job, mutex, process, or semaphore	DELETE DELETE MUTEX
Get the identification value of the current process	CURRENT PROCESS
Disable or enable process switching	DISABLE SWITCH ENABLE SWITCH
Exit from a process	EXIT

Set job or process priorities	SET JOB PRIORITY
	SET JOB PRIORITY ANY
	SET PROCESS PRIORITY

Suspend or resume a process	RESUME SUSPEND
Get the job control block (JCB) address of the currently running job	GET JCB

(continued on next page)

Table 6–3 (Cont.) Kernel Operations

Operation	Routines
Process Synchronization	
Specify the processors on which a job is eligible to run in a tightly coupled symmetric multiprocessing system	SET JOB ELIGIBILITY
Clear an event	CLEAR EVENT
Allocate or free memory for a job	ALLOCATE CLEAR HEAP ALLOCATE HEAP ALLOCATE MEMORY FREE HEAP FREE MEMORY
Communication	
Use a queue to share data between a job's processes	INSERT ENTRY REMOVE ENTRY START QUEUE

Create an area, message, name, or port	CREATE AREA
	CREATE AREA EVENT
	CREATE AREA SEMAPHORE
	CREATE MESSAGE
	CREATE NAME
	CREATE PORT

Get a job's port value	JOB PORT
Get the port value of a named port	TRANSLATE NAME
Establish a virtual circuit between two ports	ACCEPT CIRCUIT CONNECT CIRCUIT DISCONNECT CIRCUIT
Transmit a message	RECEIVE SEND
Initialize an area lock variable	INITIALIZE AREA LOCK
Wait for an area or port to be set to an available state	LOCK AREA WAIT ALL WAIT ANY WAIT ALL EXPEDITED WAIT ANY EXPEDITED

(continued on next page)

Table 6–3 (Cont.) Kernel Operations

Operation	Routines
Communication	
Signal an area	SIGNAL UNLOCK AREA
Clear an event associated with an area	CLEAR EVENT
Delete an area, message, name, or port	DELETE
Device Handling	
Create a device	CREATE DEVICE
Wait for a device to be set to an available state	WAIT ALL WAIT ANY
Signal a device	SIGNAL DEVICE
Delete a device	DELETE

Processes and Process States

A process is a thread of program execution, which the kernel can schedule and execute independently as part of a VAXELN job. A job represents a single activation of a program and can be initiated automatically at system startup or dynamically at runtime. The job consists of a master process, created automatically at job creation to execute the program's main section of code, and zero or more subprocesses created dynamically by the master process and other subprocesses. While the job is active, a job's processes are always in one of four process states: Run, Ready, Wait, or Suspended. Table 6–4 describes these states, and Figure 6–3 illustrates valid transitions from one state to another.

Table 6–4 Process States

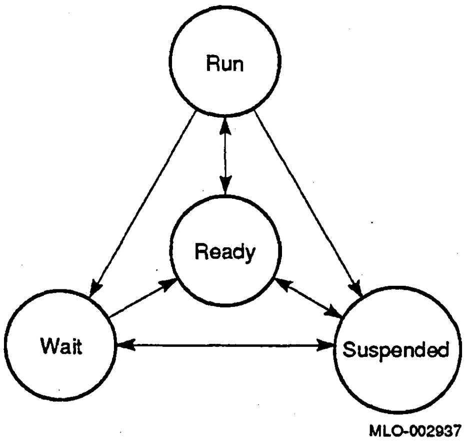
State	Description
Run	If its job is in the Run state, the process has control of the processor and is currently executing.
Ready	The process is not executing but is ready to execute as soon as the scheduler allows. When an application creates a process, the process enters the Ready state.

(continued on next page)

Table 6–4 (Cont.) Process States

State	Description
Wait	The process is waiting for a specified set of conditions to be satisfied, such as an amount of time to elapse, an event or series of events to occur, or a message to be received. A process enters the Wait state by calling one of the following routines: • WAIT ANY, WAIT ANY EXPEDITED. Wait for any of the listed conditions to be satisfied. • WAIT ALL, WAIT ALL EXPEDITED. Wait for all the listed conditions to be satisfied. • RESUME. Reenter the Wait state if the process was waiting when it was suspended with the SUSPEND routine and the wait still is not satisfied. Another process must issue the call to RESUME.
Suspended	The process cannot reenter the Ready state until another process in the same job reactivates the suspended process with a call to the RESUME routine. A process can put itself or any other process in the same job into the Suspended state with a call to the SUSPEND routine.

Figure 6–3 Process State Transitions



The rules for process state transitions are as follows:

- Ready is the initial state for a process.
- When a process's wait conditions are satisfied, it enters the Ready state. If the scheduling state of the system is such that the process can run immediately, the process enters the Run state.
- The scheduler selects a ready process to enter the Run state based on the system's jobs and process priorities.
- A process in the Run state enters the Ready state when the process is preempted by a higher-priority process.
- A process in the Run state enters the Wait state when the process issues a call to WAIT ANY, WAIT ALL, WAIT ANY EXPEDITED, or WAIT ALL EXPEDITED that blocks due to the wait conditions not being satisfied.
- If a process is in the Run or Ready state when it is suspended, it enters the Ready state when it is resumed. If the scheduling state of the system is such that the process can run immediately, the process enters the Run state immediately.
- If a process is in the Wait state when it is suspended and the wait conditions still are not satisfied when the process is resumed, it reenters the Wait state.
- If a process was in the Wait state when it was suspended and all the wait conditions are satisfied when the process is resumed, it enters the Ready state. If the scheduling state of the system is such that the process can run immediately, the process enters the Run state immediately.

Scheduling

The order in which processes enter the Run state depends on job and process scheduling. The VAXELN kernel selects a process to run on the basis of a preemptive, priority-scheduling scheme.

To accommodate preemptive priority scheduling, you must assign a priority to each job and process in a VAXELN system. You set initial priorities when you build the system, on System Builder menus, and you can change priorities dynamically at runtime with the routines SET JOB PRIORITY, SET JOB PRIORITY ANY, and SET PROCESS PRIORITY. Job priorities can range from 0 to 31 (with 0 being the highest and 16 the default). Process priorities can range from 0 to 15 (with 0 being the highest and 8 the default). Therefore, within a job, processes can have 16 levels of priority, independent of the job's priority.

The kernel scheduler considers a job ready to execute if one or more processes in that job are in the Ready state. Of the jobs and processes that are ready to execute, the kernel scheduler gives preference to those with the highest priorities. The scheduler identifies the job with the highest priority and then selects that job's highest-priority ready process for execution. If multiple jobs have the same highest priority, the scheduler selects the job that has the highest-priority ready process. The jobs in a system, whether executing or idle, are rescheduled when one or more of a job's processes enters the Ready state.

Since process rescheduling is automatic and predictable, you can design systems that execute without noticeable delays, even though some programs may sit idly while others execute.

In loosely or closely coupled multiprocessing configurations, each processor executes its own copy of a VAXELN system image, with its own kernel, system components, and program images. Single-processor scheduling rules are applied separately on each processor participating in these configurations to schedule jobs and processes.

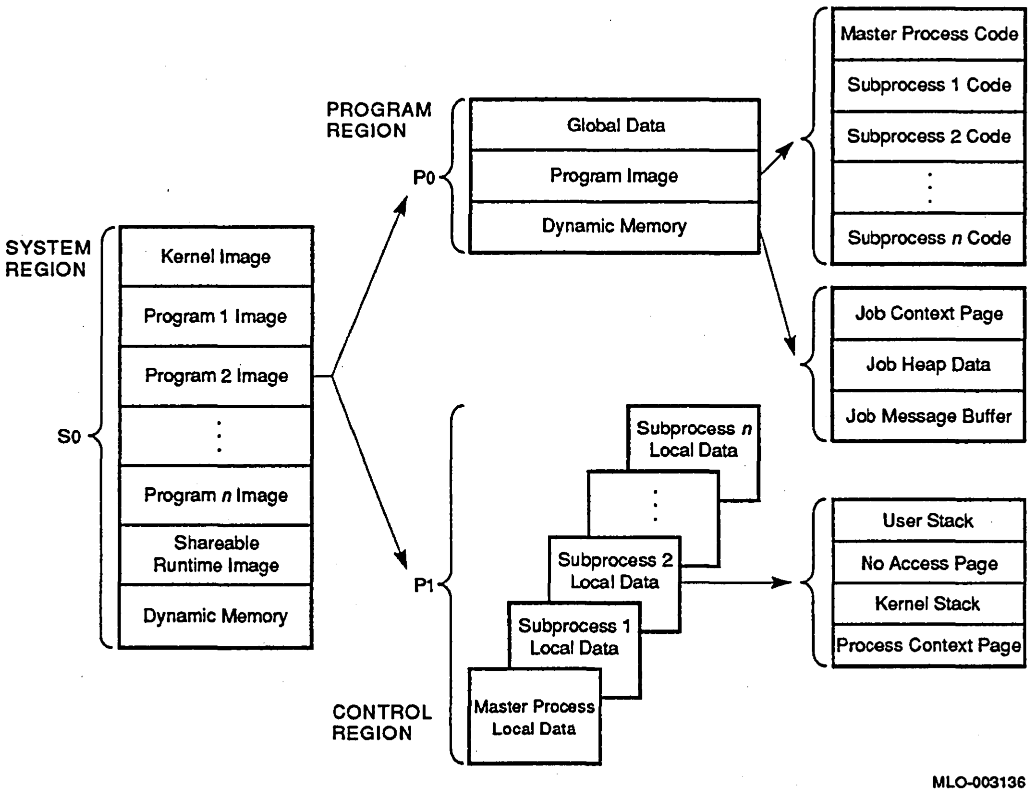
However, in a tightly coupled symmetric multiprocessing configuration, application components running on different processors share a single copy of the VAXELN system image, including the kernel, system components, and program images. The kernel schedules all processors participating in the configuration, selecting a ready job to run on each available processor. Once a job begins to run on a processor, all its processes run on that processor also; later, the job can run on another processor. If a job is eligible to run on only a subset of the processors configured for the system, the kernel ensures that the job and its processes run on those processors only. The scheduling of a job for a particular processor may preempt the processor's execution of a lower-priority job.

Initially, a job is eligible to run on any processor configured for the system. A job can alter its eligibility while executing by calling the SET JOB ELIGIBILITY routine. This mechanism lets you fine tune applications to gain maximum performance.

Memory Management

The VAXELN kernel uses VAX memory management hardware to map a VAXELN system into a processor's virtual address space. Figure 6–4 illustrates a typical mapping of virtual address space for a single activation of a program image.

Figure 6–4 Memory Allocation



When you load and boot a VAXELN system onto a target processor, the kernel maps the system image (including kernel, program, and shareable runtime images) into S0 virtual address space (the system region).

When the kernel creates a job, it creates a P0 page table and maps the job's program image, global data, and message buffers into P0 virtual address space (the program region). If multiple jobs in a system use the same program image, the kernel maps the image's global read/write data to each job's P0 address space and lets all jobs share the same read-only code and data.

The kernel uses P0 virtual address space for static variables and message text. Also, the kernel maps the context of open file variables into P0 address space so that the runtime libraries can use the variables for their data structures on a jobwide basis.

A job's processes share its P0 page table and P0 address space. Thus the processes can access the same job-level data. The processes can coordinate their access to this data by using synchronization techniques.

In addition to setting up static memory mapping, the kernel manages the data associated with dynamically created processes. When the kernel creates a process, it creates a P1 page table and maps a kernel stack into P1 virtual address space (the control region). If a job is running in user mode, the kernel maps a user stack into that address space also. Each process in a job, including the master process, has its own stacks that store process-specific data, such as local variables and procedure call frames.

The kernel uses P1 virtual address space exclusively for dynamic memory; P1 address space does not map any part of the job's program image. Kernel routines and kernel mode programs use the fixed-sized kernel stack. The kernel expands the user stack as necessary, enabling programs to start out with minimal stack space. This feature saves space that might be wasted if memory were preallocated.

Your application programs can control memory allocation directly by calling language-specific routines, such as the Pascal NEW and DISPOSE procedures; memory allocating utility routines ALLOCATE CLEAR HEAP, ALLOCATE HEAP, and FREE HEAP; and kernel routines that allocate and free P0, P1, and S0 address space.

Process Synchronization

In addition to performing scheduling and memory management tasks, the kernel coordinates operations on kernel data structures, including process synchronization. Process synchronization is a mechanism for coordinating the concurrent execution of two or more processes within a job or in different jobs. Processes must be synchronized when they share a resource or when they depend on the completion of another process's execution. The ability of a process to gain sole access to a shared resource is called mutual exclusion. The ability of a process to coordinate its activities with completion or other actions of other processes is called event response.

To attain mutual exclusion or coordinate event response, processes wait for one or more conditions to exist. Conditions for which a process might wait include:

- A specific date and time
- A period of time to expire
- Another process to terminate
- A semaphore or mutex to be signaled

- An event to occur

See Table 6–3 for a list of process synchronization operations.

Communication Between Processes and Between Jobs

Processes and jobs exchange information by applying interprocess and interjob communication techniques.

Different communication mechanisms are available for processes and jobs. Processes in the same job communicate by using module-level data (global data) and queues. Jobs communicate by passing messages and sharing areas of memory. Message-passing allows jobs to communicate whether or not they execute on the same node; sharing memory areas restricts communication to jobs executing on the same node.

Sharing Module-Level Data A job's processes can communicate by sharing module-level data: constants and variables that you declare outside routines. Module-level data is static; the kernel makes it available to all processes in a job by mapping the data in the job's P0 virtual address space. Since concurrently executing processes compete for the module-level data, you control access by using semaphores and mutexes.

Processes can also share local data (data declared within a routine) by passing the data or data pointers as arguments to routines and other processes. The kernel stores this data in process-specific P1 virtual address space.

Sharing Packets of Data Using Queues In addition to sharing module-level data, a job's processes can communicate by using queues. Queues provide an efficient, highly structured means for a job's processes to exchange large packets of information.

VAXELN provides the predeclared data types and routines you need to create and maintain queue structures. The routines for inserting and removing queue entries use VAX machine instructions specifically designed to synchronize queue operations automatically. Thus two processes can access a queue simultaneously: One can insert an entry while the other removes an entry. An application can use semaphores or mutexes to indicate the queue's state transitions (empty to nonempty and full to nonfull).

Passing Messages A message is a block of contiguous bytes of memory that is transmitted between processes in the same or different jobs. The kernel maps message data into a job's unique, protected P0 virtual address space, making the data available to all processes in that job. Within a single system, the kernel uses VAX memory management hardware to distinguish the virtual address space for each job. Within a local area network (LAN), the virtual address space for each job resides in the memory of its system, which might be a different target system. By passing messages, the jobs in a VAXELN system can use the same mechanism to share data efficiently and transparently in both single-processor and multiprocessor configurations.

Processes send messages to and receive messages from system-maintained queues called ports. The ports in a system store messages that are waiting to be received. Calls to the CREATE PORT routine create ports dynamically and associate them with unique port identifiers that can be used throughout the application: within the creating job, within other jobs on the same node, or within jobs on other network nodes.

To facilitate interjob communication and to make the distribution of applications across LAN nodes transparent, you can use the CREATE NAME routine to associate port identifiers with port names. When ports are associated with names, a process can call the TRANSLATE NAME routine to look up a name in a table. Then the process can use the returned port identifier to communicate with other processes and jobs. Port names can be up to 31 characters long and can be either local or universal. Local port names are known only to processes and jobs on the node on which they are created. Universal port names are known to processes and jobs on all nodes in the LAN.

VAXELN systems can pass messages by using datagrams or virtual circuits. The datagram method, which uses the DECnet–VAX datagram, requires no explicit connection sequence and provides fast communication with low overhead. The datagram method cannot guarantee message delivery or sequence, but the probability of received messages being correct is extremely high. Using datagrams, a process can obtain the identifier of any named port in the system, whether the port is on the same node or on a different node on the Ethernet.

The virtual circuit method, which uses the network services protocol within the VAXELN DECnet Service, is the preferred method for VAXELN systems to pass messages. Virtual circuits require two ports, usually in different jobs, to be connected as a pair. Despite the overhead of setting up and handling a virtual circuit connection, circuits offer the following advantages:

- Guaranteed delivery and sequence
- Flow control
- Message size not limited by the underlying physical media characteristics due to automatic message segmentation and reconstruction

When a job sends a message to another job on the same node, the kernel simply unmaps the message buffer's address from the sending job's virtual address space and maps the address to the receiving job's address space. If the jobs are on different Ethernet nodes, the VAXELN DECnet Service transports the data across the network and places it in the receiving job's virtual address space. (Network configurations limit datagram message size to the maximum imposed by relevant network devices.)

Sharing Memory Areas Jobs executing on the same node can communicate by sharing an area, a common region of physically contiguous memory. Each job that shares an area must identify it by declaring an area object and specifying the same area identifier in a call to one of the following routines:

CREATE AREA
CREATE AREA EVENT
CREATE AREA SEMAPHORE

The kernel maps the associated physical memory to each job's virtual address space. The first job that calls one of these routines creates the area and an associated event or semaphore that controls access. Subsequent calls to the routine let other jobs gain access to the area. You synchronize access to the area by specifying the area in calls to the WAIT, SIGNAL, and CLEAR EVENT routines. Using the LOCK AREA and UNLOCK AREA routines, jobs synchronize access to the area without issuing a wait. A job executes a wait operation for an area only if the area is already locked.

Device Handling

The kernel also provides routines for handling devices and device interrupts. Using the device kernel object and the associated kernel routines, you can write sophisticated device drivers for external hardware devices, such as sensors, analog-to-digital converters, array processors, communication lines, and robots. Section 6.2.5 discusses device drivers in more detail.

Exception Handling

To achieve realtime performance, VAXELN systems must respond to exceptions without significant delay. An exception is a hardware or software event that changes the normal flow of a program's execution synchronously or asynchronously. Synchronous exceptions occur at predictable points during a program's execution (for example, dividing by zero). In contrast, asynchronous exceptions result from unpredictable events, such as power failures. Table 6–5 lists examples of exception types according to their hardware or software origin.

Table 6–5 Types of Exceptions

Exception Source	Exception Type
Hardware	Division by zero Integer overflow Nonexistent memory addressing Power failure
Software	Signal sent to a process Pascal range violation Error opening a disk file Runtime error that the kernel detects

VAXELN application programs can handle a variety of exceptions by checking status codes or by using exception handlers. You have the option of specifying a status argument in kernel and utility routine calls. By checking whether the argument's return value is odd (success) or even (failure), a program can determine whether an operation succeeds or fails.

If you include VAXELN message object modules when you compile and link a program, you can also check for specific exception conditions. The message modules associate status codes with symbols. In your program, you can compare these symbols with the status codes that routines return to their status arguments.

When an exception condition occurs, the action taken depends on whether you establish an exception handler or include a debugger in the system.

- If you establish an exception handler, it takes control and appropriate action.
- If you do not establish an exception handler and you include debugger support in your system, the debugger takes control when an exception occurs.
- If you do not establish a handler and you do not include debugger support, the kernel deletes the process that caused the exception.

The VAXELN Toolkit provides routines that you can use to create exception handlers for conditions that require more extensive handling than status code checking. For example, before you create an exception handler in a Pascal program, you declare the handler as a function of type EXCEPTION_HANDLER. Once you declare the handler, you can use exception-handling routines to perform the operations listed in Table 6–6.

Table 6–6 Exception-Handler Operations

Operation	Routine
Establish a function as a block's exception handler	ESTABLISH
Disable exception handlers	REVERT
Prevent the delivery of asynchronous exceptions to a calling process	DISABLE ASYNCH EXCEPTION
Allow the delivery of asynchronous exceptions to a calling process	ENABLE ASYNCH EXCEPTION
Raise an exception in the calling process	RAISE EXCEPTION
Raise the asynchronous exception KER$_PROCESS_ATTENTION in a process	RAISE PROCESS EXCEPTION
Unwind an exception handler's call stack	UNWIND
Get text associated with a status code	GET STATUS TEXT

6.2.2 VAXELN POSIX Runtime Library

The VAXELN POSIX runtime library lets you develop and run portable applications in a POSIX environment. The POSIX standards and drafts support the concept of open systems.

The VAXELN Toolkit includes limited support for the following POSIX standards and draft standards:

- POSIX 1003.1–1990
- P1003.4/D11
- P1003.4a/D4

The POSIX Characteristics Menu on the System Builder menu lets you select some POSIX characteristics, such as the round-robin interval.

VAXELN POSIX applications that call VAXELN routines must be linked with both the VAXELN POSIX and VAXELN C runtime libraries. The PSXSHARE and CRTLSHARE runtime libraries contain pointers to shareable images and should be linked with your VAXELN application.

POSIX 1003.1–1990

The POSIX 1003.1 (interface) standard defines a set of functions for use in application programs that use the C programming language. The use of the POSIX 1003.1 standard in an application promotes application portability among systems that support the POSIX 1003.1 standard. POSIX 1003.1 includes: some functions that are identical to ANSI C functions; some functions that use the same syntax as ANSI C functions, but operate slightly differently in the POSIX environment; some functions that are unique to POSIX.

POSIX 1003.4/D11

The P1003.4/D11 (realtime) standard defines a set of functions that can be used in the design and creation of realtime applications in the VAXELN POSIX environment. These functions facilitate realtime application development in the following areas:

- Process synchronization
- Communication between processes
- Process memory locking

Process Synchronization Using VAXELN POSIX

Using synchronization techniques and restrictions on resource access ensures that critical and noncritical activities execute at appropriate times with the necessary resources available. The following table lists the synchronization functions available in the VAXELN POSIX runtime library.

Synchronization	Header File	Purpose
Binary semaphores	**<binsem.h>**	Restricts access to resources
Clocks and timers	**<time.h>**	Arms and disables timers
Priority scheduling	**<sched.h>**	Sets process priority and the scheduling policy

Binary Semaphores A binary semaphore is a synchronization mechanism used to control access to systemwide resources. With VAXELN POSIX binary semaphores, you can create and remove persistent and nonpersistent binary semaphores. You can also release, wait for, and lock binary semaphores. Each action on a semaphore requires an explicit function call and is under the control of the application programmer.

Semaphores are used by cooperating processes to synchronize access to resources, such as shared memory. Semaphores can protect resources such as global variables, hardware resources, and the kernel from uncontrolled access.

Semaphore protection works only if all communicating processes using the shared resource cooperate by waiting for the semaphore when it is unavailable and resetting the semaphore count when relinquishing the resource. For cooperating tasks, semaphores are mutual exclusion flags that lock and unlock a resource.

Clocks and Timers Realtime clocks and timers allow an application developer to synchronize and coordinate activities according to a predefined schedule. The systemwide clock (CLOCK_REALTIME) provides the timing base for per-process timers. VAXELN POSIX clock and timer functions allow you to retrieve and set the systemwide clock, suspend execution for a period of time, provide high-resolution timers, and use asynchronous event notification. Realtime timers are created, armed, and removed by the application programmer.

Realtime timers allow the application to set timers based on either absolute or relative time. Furthermore, VAXELN POSIX timers can fire as either a one-shot or periodic timer. The application creates timers in advance, but the timers can be manipulated according to the needs of the realtime application. Some applications may require only one or two timers; others may require multiple timers within a single process.

Priority Scheduling The scheduler determines how CPU resources are allocated to executing processes. Each process has a priority that associates the process with a run queue. Although each process starts out with an initial priority, that priority can change as the application executes.

Priority scheduling gives an application programmer control over the execution sequence of processes that comprise an application. Priority scheduling also addresses the need for a realtime process to execute when it needs to and for as long as it needs to.

The system maintains a list of runnable processes at each priority level. Each process list has a priority value ranging from PRIO_MIN to PRIO_MAX. A process in a list with a high priority value executes before a process in a list with a low priority value.

A scheduling policy is the algorithm that determines how processes are placed on the process list and when processes execute. Whatever the scheduling policy, the process at the head of the process list with the highest priority level executes first.

VAXELN POSIX supports two scheduling policies: first-in, first-out (FIFO) and round-robin (RR). These two scheduling policies determine how the process lists are managed, the order of the process lists, and when processes of different priorities can execute.

Table 6–7 categorizes synchronization functions and lists the corresponding VAXELN POSIX functions.

Table 6–7 VAXELN POSIX Process Synchronization Functions

Operation	Functions
Make a binary semaphore special file or determine the number of semaphores in a set	**sem_mksem** **sem_getnsems**
Open, close, and destroy a binary semaphore	**sem_open** **sem_close** **sem_destroy**
Wait (or conditionally wait) for a binary semaphore	**sem_wait** **sem_ifwait**
Release a binary semaphore	**sem_post**
Get or set the value of the systemwide clock, get the resolution of the clock	**clock_gettime** **clock_settime** **clock_getres**
Get or set the clock drift rate	**clock_getdrift** **clock_setdrift**
Allocate or free a per-process timer, get the value or overrun of the per-process timer	**timer_create** **timer_delete** **timer_gettime**
Arm a per-process timer absolutely or relatively	**timer_settime**
Get or set the priority of a process, get the maximum or minimum priority allowed	**sched_getparam** **sched_setparam** **sched_get_priority_max** **sched_get_priority_min**
Get or set the scheduling policy	**sched_getscheduler** **sched_setscheduler**
Get the quantum allowed for round robin scheduling	**sched_get_rr_interval**
Yield to another process	**sched_yield**

Communication Between Processes Using VAXELN POSIX

Communication is a way of isolating or controlling different levels of functionality within an application. Using communication between processes, you can synchronize independently executing processes by passing data within an application. Processes can pursue their own tasks until they must synchronize with other processes at some predetermined point. When they reach that point, they wait for some form of communication to occur. The following table lists the types of communication functions available in VAXELN POSIX.

Communication	Header File	Purpose
Signals	**<signal.h>**	Allows two or more processes to communicate through signals
Shared memory	**<mman.h>**	Allows two or more processes to share the same address space

Signals The signal interface is a traditional form of communication and is generally used to notify processes that something has happened in one process that affects another process. Signals are a way of passing data within an application when the application must respond to an application-defined occurrence. Signals are often sent asynchronously; that is, the receiving process cannot predict when a signal will arrive. The application must contain code to take action once the signal is received. The action can be to terminate the process, ignore the signal, or catch the signal and respond appropriately.

Signals provide a means to communicate among a large number of processes, but communication is limited to a signal number. Signals use a data structure, making signal delivery asynchronous, fast, and reliable.

Signals do not pass data, do not identify the sending process, and are not prioritized. Nevertheless, signals are used by realtime timers and other events to trigger the start of a signal handler once the signal is received.

VAXELN POSIX signals use the POSIX 1003.1 signals functions as well as the P1003.4/D11 realtime signal functions.

Shared Memory The use of shared memory allows multiple processes to share data because a region of memory is mapped into each process's address space. Use of shared memory among processes provides faster data access.

You can allocate as little memory as possible for each process or allocate one large region and manage it within the application. Shared memory can be created as persistent or nonpersistent, mapped and unmapped, linked and unlinked, or closed.

Table 6–8 lists communication operations and the corresponding VAXELN POSIX functions.

Table 6–8 VAXELN POSIX Communication Functions

Operation	Functions
Send a signal	**kill** **sigsend**
Suspend the calling process until a signal is delivered	**pause** **sleep** **nanosleep** **wait** **waitpid**
Store a set of pending signals, poll for queued events	**sigpending** **sigpoll**
Queue a signal to a process	**sigqueue**
Wait for queued realtime signals	**sigwaitrt** **sigtimedwait**
Examine or specify the action of a signal	**sigaction**
Open and unlink a shared memory object	**shm_open** **shm_unlink**

Locking Memory

VAXELN POSIX applications do not require explicit use of memory-locking functions because applications are automatically locked in memory. There is no paging or swapping in VAXELN POSIX applications.

However, to make your application portable to other environments, you may need to use the POSIX memory-locking functions. Although the calls to these functions do not affect the performance of an application, the software checks the syntax of the calls.

Table 6–9 lists the VAXELN POSIX memory-locking functions.

Table 6–9 VAXELN POSIX Memory-Locking Functions

Operation	Functions
Lock and unlock a specified memory region	**mlock** **munlock**
Lock and unlock all memory mapped by the address space	**mlockall** **munlockall**

POSIX 1003.4a/D4

The P1003.4a/D4 (threads) standard defines a set of thread functions that can be used in the design and creation of multithreaded realtime applications in the VAXELN POSIX environment.

A thread is a single, sequential flow of control within a program. Within a single thread, there is a single point of execution. In a multithreaded program, threads execute concurrently with multiple points of execution. Since threads execute within a single address space, threads can read and write to the same shared memory locations. Therefore, thread usage must be synchronized through such elements as mutexes and condition variables to ensure that shared memory is accessed appropriately.

On single-processor systems multiple threads can improve application performance by overlapping I/O operations with computational operations.

Threads can improve the performance of slow devices such as disks, networks, terminals, and printers. A multithreaded program can perform useful work while waiting for the device to complete its next event (such as a disk transfer or the receipt of a packet from the network).

VAXELN POSIX provides a full complement of threads functions. These functions perform the following types of tasks:

- Create, delete, and determine thread attributes
- Create, delete, and determine thread mutex attributes
- Set or retrieve the scheduling policy or priority of a thread
- Create, initialize, wait for, or delete condition variables
- Perform specific or global locks
- Cancel or delay thread execution
- Notify the scheduler of a yield
- Perform cleanup operations

6.2.3 KAV30 Runtime Library

With the KAV30, you develop applications on a VMS host system using the VAXELN Toolkit and high-level languages and runtime libraries such as Ada, VAX C, VAX FORTRAN, and VAXELN Pascal. The application is built into a VAXELN runtime system, downline loaded into a KAV30 target via Ethernet, or booted from SCSI disk or EPROM. The VAXELN Toolkit runtime libraries contain the language bindings to use these languages with the KAV30.

The KAV30 module allows you to have a common distributed VAX architecture extending from the mainframe to the VMEbus. Being on the VMEbus makes the KAV30 well suited for industrial front end solutions where a broad selection of readily available sensor interfaces and I/O options is required. Over 4000 VME modules are available from more than 400 suppliers.

The VAX architecture, when combined with the powerful software development and runtime environment provided by the VMS host system and the VAXELN Toolkit, significantly reduces product development times. Applications, including device drivers, are developed in high level languages with compilers common between host and target. Extensive networking and integration capabilities are provided using DECnet and TCP/IP protocols.

The VAXELN kernel for the KAV30 has the following capabilities:

- **Asynchronous system trap processing.** VAXELN applications can include device drivers for the devices on the VMEbus or VSB that interact with the KAV30. These device drivers contain code to call AST routines to handle interrupts.
- **Timers.** There are five 32-bit timers and two 16-bit timers on the KAV30. One 16-bit timer is a watchdog timer while the other one is the local bus timeout timer.
- **Calendar/clock.** The calendar/clock maintains the time and date in units as small as one-hundredths of a second. This programmable calendar/clock can be used to set alarms and timers in several different formats. The timesave RAM stores the contents on the clock in the event of a power failure.
- **FIFO buffers.** Four independently operating FIFO buffers enable intelligent devices on the VMEbus to exchange data with the KAV30.
- **Battery backed-up RAM.** The battery backed-up RAM allows you to store critical information in the event of a system or power failure.
- **Scatter-gather map.** The scatter-gather map (SGM) allows the KAV30 to use a master/slave model to access devices on the VSB or VMEbus.
- **Ability to communicate with VMEbus devices.** A KAV30 can communicate with VMEbus devices by using shared memory or by exchanging data through the FIFO buffers.
- **Error logging support.** The KAV30 kernel logs errors for timeouts, bus errors, invalid SGM entries, or SGM access violations.

The VAXELN KAV30 system services allow you to perform the following tasks:

- Initialize the KAV30

- Establish and control access to the devices on the VSB and VMEbus
- Establish and control the KAV30 FIFO buffers
- Establish and control the calendar/clock and counter/timers
- Exchange data with the VSB and VMEbus devices
- Read and write RAM on the KAV30
- Collect error information from the RAM
- Use ASTs in user-written device drivers
- Interrupt another VMEbus device

The VAXELN system service names for the KAV30 begin with KAV$. The exception is services in VAX Ada. Because the dollar sign ($) is not part of the VAX Ada character set, KAV30 kernel services in VAX Ada have the facility prefix KAV_.

Table 6–10 summarizes the VAXELN system service routines for the KAV30.

Table 6–10 KAV30 Routine Summary

System Service	Description
KAV$BUS_BITCLR	Clears the bits at a VSB or VMEbus address
KAV$BUS_BITSET	Sets the bits at a VSB or VMEbus address
KAV$BUS_READ	Reads the contents of a VSB or VMEbus address
KAV$BUS_WRITE	Writes data to a VSB or VMEbus address
KAV$CHECK_BATTERY	Checks the power supply to the RAM and the calendar/clock
KAV$CLR_AST	Clears a device's AST queue
KAV$DEF_AST	Creates an AST control block for an event
KAV$FIFO_READ	Reads data from a KAV30 FIFO buffer
KAV$FIFO_WRITE	Writes data to a KAV30 FIFO buffer
KAV$GATHER_KAV_ERRORLOG	Reads error log information from the KAV30 RAM
KAV$IN_MAP	Maps a 64 Kbyte page of VMEbus address space to the KAV30 process address space
KAV$INT_VME	Generates vectored VMEbus interrupts
KAV$LIFO_WRITE	Writes data to a KAV30 LIFO buffer

(continued on next page)

Table 6–10 (Cont.) KAV30 Routine Summary

System Service	Description
KAV$NOTIFY_FIFO	Delivers an AST when a specified event occurs in a KAV30 FIFO buffer
KAV$OUT_MAP	Maps KAV30 system I/O space to the VSB or VMEbus address space, in 64 Kbyte pages
KAV$QUE_AST	Queues an AST for delivery to a process
KAV$RTC	Performs realtime clock functions, using the KAV30 calendar/clock
KAV$RW_BBRAM	Reads or writes the KAV30 RAM
KAV$SET_AST	Places an entry in the AST queue for a device
KAV$SET_CLOCK	Sets the KAV30 system clock and the calendar/clock
KAV$TIMERS	Sets a counter/timer and delivers and AST when the timer interval expires
KAV$UNMAP	Unmaps VMEbus address space from KAV30 system RAM or unmaps the KAV30 system RAM from the VMEbus address space
KAV$VME_SETUP	Configures the VSB and VMEbus interrupt delivery mechanism

6.2.4 VAXELN Runtime Libraries

The VAXELN Toolkit supplies runtime libraries for linking with VAXELN Pascal, VAX C, and VAX FORTRAN object modules. The libraries are organized into object module and shareable image libraries so you can customize the runtime support for your system images. Depending on the libraries you link with your program object modules, a system image can include:

- Program images that share one copy of runtime routine code
- Program images that use local copies of runtime routine code
- A combination of images that share routine code and images that use local routine code

6.2.5 Device Drivers

Realtime applications require fast, predictable response to service external devices, such as sensors, communication lines, and robots. VAXELN systems meet this requirement by using device drivers that minimize overhead and maximize the VAX processor's speed and responsiveness.

Device drivers are programs that control communication between application programs and external devices. In the case of realtime applications, most external devices are interrupt driven; they communicate with the application only when they need service. A device requests service by sending an interrupt signal to the processor. The processor recognizes the signal, stops what it is doing, and services the request by executing a block of the application's code called an interrupt service routine (ISR).

Once you decide on your application's device requirements, you build the relevant devices and drivers into your VAXELN system by specifying device characteristics on the System Builder's Device Description Menu. (The System Builder is discussed in Section 6.1.3.)

The VAXELN Toolkit simplifies VAX device support with the following facilities:

- Pregenerated drivers for commonly used devices
- Virtual-memory driver for RAM disk support
- Support for writing drivers, including the ISRs, in high-level languages
- High-level language device driver templates you can use for writing your own device drivers

The next sections provide information about the following:

- VAXELN device support
- Small Computer System Interface (SCSI) driver
- Virtual-memory driver
- Software for developing application-specific drivers

VAXELN Device Support

The VAXELN Toolkit supplies device drivers that you can include in your VAXELN systems. These drivers provide support for a variety of disk, tape, printer, terminal, Ethernet, and realtime devices.

Realtime device support includes a collection of object modules that reside in the RTLOBJECT module library and corresponding source code. The object modules define routines for interfacing with the supported realtime devices. You add realtime device support to a VAXELN system by including the appropriate module in one of the system's application programs and linking that program with the RTLOBJECT library.

The toolkit supplies the source code for a number of the supplied realtime device drivers. You can use the supplied source code as templates while writing your own drivers.

For lists of the bus devices currently supported by the VAXELN Toolkit, see the VAXELN Toolkit *System Support Addendum (SSA)* or *Software Product Description (SPD)*.

Small Computer System Interface Driver

The VAXELN Toolkit provides a driver image that supports the American National Standards Institute, Small Computer System Interface (SCSI) devices on realtime systems. The image includes a disk class driver and a generic class driver.

For lists of the class drivers currently supported by the VAXELN Toolkit, see the VAXELN Toolkit *System Support Addendum (SSA)* or *Software Product Description (SPD)*.

The generic class driver provides an interface for all other types of SCSI devices, such as scanners, optical devices, test equipment, and medical devices.

VAXELN application programs can use a supplied message interface to communicate with the generic class driver.

You can use the VAXELN SCSI driver image for third-party SCSI devices that attach to realtime systems. The disk class driver supports direct-access and CD–ROM devices, whereas the generic class driver provides an interface for all other devices.

You can also combine a user-written SCSI class driver with the supplied VAXELN SCSI port driver to produce a vendor-specific VAXELN SCSI driver image. Then you can build that image into a VAXELN system.

Note

The *American National Standard for Information Systems—Small Computer System Interface-2 (SCSI-2)* specification allows flexibility for some device implementation details and omits other details. Thus implementations of the SCSI standard may differ from manufacturer to manufacturer and from device to device. Although you can use third-party devices with the VAXELN SCSI disk class driver, the VAXELN Toolkit does not necessarily support such devices.

Digital does not guarantee that third-party devices that currently run with the supplied class driver will continue to run under subsequent releases of the VAXELN Toolkit.

The *American National Standard for Information Systems—Small Computer System Interface-2 (SCSI-2)* specification should be the official guide to what a third-party device implements.

To ensure that your third-party device will work properly in a VAXELN environment, Digital encourages the use of an established and supported VAXELN interface.

You can use the following facilities to build third-party SCSI device support into VAXELN systems:

- VAXELN SCSI disk class driver
- VAXELN SCSI generic class driver message interface
- A user-defined class driver that communicates with the VAXELN SCSI port driver

The application designer must decide which method to use for a particular SCSI device application. The designer should consider the SCSI device's capabilities, user needs, and available programming resources.

For more information about Digital's implementation of the *American National Standard for Information Systems—Small Computer System Interface-2 (SCSI-2)* specification and how to use the implementation to develop SCSI peripheral devices that are currently available through Digital, see *Small Computer System Interface: An Overview* and *Small Computer System Interface: A Developer's Guide*.

Virtual-Memory Driver

The VAXELN Toolkit's virtual-memory driver (VMDRIVER) lets you create a virtual disk structure in system memory and use the memory as you would an actual disk drive. You can use the virtual disk structure as a scratch disk for the life of the system. Multiple readers and writers can share the disk, and it can participate in network file operations.

The VMDRIVER runs as a job in a VAXELN system. You build the driver into a system by entering the driver's characteristics on the System Builder's Program Description Menu.

The memory pages used for the VM disk are allocated from contiguous addresses in system virtual address space. Therefore, once the disk is initialized, you cannot extend it.

Customized, Application-Specific Device Drivers

The VAXELN Toolkit provides a highly productive environment for developing application-specific device drivers. You can implement drivers in VAX Ada, VAX C, VAX FORTRAN, or VAXELN Pascal; you have the option of customizing existing driver code.

You can design a device driver so that it executes as an independent job or as part of a user job. As an independent job, a device driver is a shareable resource available to all program images in a system. Such a driver might support a single unit, such as a line printer, or it might function as a server supporting multiple units.

A driver that is part of a user job might be a process within the job or a collection of subroutines. Such a driver reduces overhead by eliminating job context switching, but only processes in the job that includes the driver can use it.

A VAXELN device driver executes concurrently with the jobs that use the related device. The driver's activity depends on the characteristics and actions of the device it controls. However, you program a driver's general interface by declaring a variable of type DEVICE (which represents the hardware device) and an ISR. You then specify these constructs in routine calls to perform the operations listed in Table 6–11.

A driver's ISR provides a fast, customized interface for handling device interrupts and power recovery. When an interrupt occurs, the kernel executes a maximum of two machine instructions on a single processor system; then it calls the ISR to service the device. While servicing the device, the ISR communicates with the driver code by sharing an area of memory called the communication region. The driver establishes this region when it creates the device object. When the ISR finishes servicing the device, it unblocks a waiting driver process by calling the SIGNAL DEVICE kernel routine.

Table 6–11 Device-Handling Operations

Operation	Routine
Associate a device with an ISR and a driver program	CREATE DEVICE
Read data from a device's control status register (CSR) or data buffer	READ REGISTER

(continued on next page)

Table 6–11 (Cont.) Device-Handling Operations

Operation	Routine
Write data to a device's CSR or data buffer	WRITE REGISTER
Cause a process to wait for an ISR to service a device interrupt	WAIT ALL WAIT ALL EXPEDITED WAIT ANY WAIT ANY EXPEDITED
Indicate that the ISR has finished servicing an interrupt	SIGNAL DEVICE

Depending on your target configuration, a driver program can use either the DISABLE INTERRUPT and ENABLE INTERRUPT routines or the LOCK DEVICE and UNLOCK DEVICE routines to synchronize access to the communication region. You use DISABLE INTERRUPT and ENABLE INTERRUPT if your target is a single processor or if it is a VAX 8800 multiprocessor. You use LOCK DEVICE and UNLOCK DEVICE if your target configuration includes a multiprocessor that lets devices interrupt on any processor, such as a VAX 6000 series multiprocessor.

For a driver job to use DISABLE INTERRUPT on a VAX 8800 multiprocessor, the job must be running on the processor that handles the device's interrupts. The driver sets this up automatically when an application creates the device object. If a driver does not use DISABLE INTERRUPT, it can use the SET JOB ELIGIBILITY routine to make the job eligible to run on other processors. If necessary, an application can request specific processor eligibility at runtime by issuing a call to the SET JOB ELIGIBILITY routine.

6.2.6 Network Communications Services

The VAXELN Toolkit includes Ethernet/IEEE 802 datalink drivers for supported network devices. Each of the datalink drivers supports the VAXELN Ethernet/IEEE 802 Datagram Service, VAXELN DECnet Service, and VAXELN Internet Services.

- The Datagram Service provides an interface that VAXELN systems can use to communicate with other types of systems using system-independent communications protocols.
- The DECnet Service routes messages sent between two DECnet LAN nodes, manages VAXELN universal names for the LAN, and furnishes a runtime interface for managing DECnet on the local node.

- The Internet Services provide an Ethernet network interface that VAXELN systems can use to communicate with other applications in an Internet network.

The datalink drivers also provide security and local area transport (LAT) support. Security is furnished through the Authorization Service, which maintains a database of authorized users and identifies users that issue network requests. LAT host services let VAXELN systems communicate with devices attached to terminal servers.

The VAXELN datalink drivers support one or more Ethernet controllers in a system. VAXELN systems can include multiple Ethernet controllers of the same type and can participate in homogeneous or heterogeneous networking environments. Although DECnet software can run on only one controller at a time, you can implement other private Ethernet/IEEE 802 protocols that can run on all available controllers. For example, if your system is configured with two Ethernet controllers, DECnet can operate over one of the controllers while both controllers are used for private Ethernet/IEEE 802 protocols.

Ethernet/IEEE 802 Datagram Service

The VAXELN Ethernet/IEEE 802 Datagram Service enables VAXELN systems to communicate with other types of systems over a Carrier Sense Multiple Access/Collision Detect (CSMA/CD) LAN. Using the service's network interface routines, you can program system-independent communications protocols that your VAXELN application can use to communicate in such environments. VAXELN application programs can use the routines to do the following:

- Retrieve the CSMA/CD LAN configuration
- Retrieve Ethernet controller attributes
- Connect and disconnect an Ethernet/IEEE 802 protocol
- Transmit and receive messages over the CSMA/CD LAN

DECnet Service

The DECnet Service controls message transmission between DECnet network nodes, manages a network name table, and provides an interface for managing DECnet on the local node. You configure a DECnet Service for each target node using DECnet software in a multinode application. The DECnet Service preserves the methods for sending and receiving messages, whether jobs communicate on the same node or between nodes; data transmission across network nodes is transparent to your programs.

DECnet Service Protocols The DECnet Service employs the following Phase IV DECnet protocols:

- Routing protocol
- Network services protocol (NSP)
- Session control protocol (SCP)
- Data access protocol (DAP)

The routing protocol routes system-level datagrams between VAXELN nodes and other DECnet nodes. The NSP and SCP support transparent application-level circuits that are connected to remote nodes.

The VAXELN runtime software uses DAP in most communication tasks within an application. For example, console and disk I/O use DAP as their highest-level interface. All VAXELN disk, tape, and terminal drivers have DAP front ends to facilitate transparent multiprocessing in LAN configurations.

In addition, VAXELN uses direct device access (DDA) to perform I/O functions that the DAP architecture does not define. DDA provides an interface for disk and serial-line read and write operations. This protocol also provides an interface for dynamically setting serial-line characteristics, setting serial lines to the spacing state, monitoring the use of out-of-band characters, and controlling modem signals.

Message Transmission The DECnet Service uses Phase IV DECnet protocols to add transparent network extensions to the message-passing kernel routines ACCEPT CIRCUIT, CONNECT CIRCUIT, DISCONNECT CIRCUIT, RECEIVE, and SEND. When an application uses these routines to pass messages between two network nodes, the kernel and DECnet Service on each node cooperate to ensure message delivery. When a process sends a message, the kernel checks to see if the specified port value is local to the executing node. If it is not, the kernel and DECnet Service route the message through the receiving node's DECnet Service to the destination port. The receiving process receives and replies to the message as though executing on the same node as the sending process.

Name Service The DECnet Service also provides a name service, which adds network extensions to the CREATE NAME, TRANSLATE NAME, and DELETE kernel routines. These extensions let jobs access and maintain a table of universal port names, which are known to all VAXELN nodes in a VAXELN LAN.

Universal port names are the key to distributed applications. Universal port names allow a VAXELN system to move a job or disk file to another node without your having to modify code. The DECnet Service ensures the validity of the communication path. Thus, a job running on one node can open, read, and write files located on another node, while the use of multiple nodes remains transparent to the user. Also, jobs executing on different nodes can communicate over a network transparently by establishing circuits and exchanging messages between named ports.

Each target system in a VAXELN network application retains a list of the universal names it creates and sends a copy of those names to the network's universal name table. One of these target systems serves as an acting name server and manages this table. If the acting name server shuts down, another node in the LAN that has universal name service capabilities is elected as the name server.

Network Management The DECnet Service supports a subset of the Phase IV network management protocol and VAXELN network management services. To manage VAXELN DECnet nodes from your VMS development system, you use the DECnet–VAX network control program (NCP). You can use the NCP functions to invoke the following facilities:

- **Network management listener (NML)** Monitors the network and controls DECnet systems.
- **Loopback Mirror.** Tests the DECnet Service and its ability to communicate with other nodes on the network.

The DECnet Service also provides the following services for managing VAXELN DECnet nodes from VAXELN target systems:

- **Network Management Service.** Furnishes a routine interface for dynamically starting and stopping DECnet software at runtime. A VAXELN application can use the routines to initialize DECnet addresses at runtime, start and stop DECnet to temporarily reduce network overhead, or switch the Ethernet controller on which DECnet is to run.
- **Downline Load Service.** Handles VAXELN system load requests and provides a runtime routine interface. An application can use the interface routines to configure, manage, and monitor a memory-resident downline load database. Applications can also use routines to trigger boot VAXELN systems to remote VAXELN target nodes.

Communication with VMS and ULTRIX Nodes The DECnet Service also provides for communication with VMS and ULTRIX network nodes. Jobs running on VAXELN nodes can communicate with jobs running on VMS and ULTRIX nodes transparently by using standard I/O statements. Alternatively, jobs running on VMS and ULTRIX nodes can use network-specific features, such as mailboxes, to exchange messages with VAXELN jobs.

Remote Terminal Utility The Network Service provides a remote terminal utility that lets you connect to a remote computer system from a terminal on another computer system by using a SET HOST command. For example, you can connect to a VAXELN system from a VMS system terminal by using the DCL SET HOST command, or you can connect to a VMS system from a VAXELN system terminal by using the ECL SET HOST command. Once connected to a remote system, you can log in, use operating system commands (such as DCL and ECL commands), receive messages, and interact with programs that run on that system.

To use the VAXELN Remote Terminal Utility, you must build it into your VAXELN system with outbound, inbound, or outbound/inbound capability. The outbound capability lets you connect to computer systems from your VAXELN system. The inbound capability lets you connect to your VAXELN system from other systems.

Internet Services

VAXELN applications can use the VAXELN Internet Services to communicate between two computer hosts that reside on the same or on different networks. The hosts are the sources and destinations of transferred data. The Internet Services provide the protocols necessary for VAXELN applications to transfer data over an Internet.

An Internet is a set of networks that are connected by hosts called gateways. A network is a collection of hosts that are physically connected by a communications medium, such as an Ethernet. Gateways physically connect and transfer messages between networks. Higher-level software hides the underlying Internet architecture and makes a collection of networks appear as a single large network. Applications can communicate across intermediate networks even though the networks are not connected to the source or destination host.

Note

Although VAXELN systems can use gateways for Internet communication, they cannot function as gateways.

The VAXELN Internet Services provide the following features:

- Connectionless packet delivery service or end-to-end connection-oriented stream delivery service
- Packet delivery service that is independent of the communications medium over which data is transmitted
- Communications environment that supports a variety of computer platforms
- Communications protocol standards support

The Internet architecture consists of four layers of protocol that allow two-way interprocess data flow between hosts, gateways, and networks. The architecture includes an application layer, host-to-host protocol layer, Internet protocol (IP) layer, and network protocol layer. The host-to-host layer supports two protocols: the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP).

Processes on a host transmit data by passing it to the lower protocol layers. A process at the application layer passes the data to the host-to-host protocol layer. The host-to-host protocol layer then packages the data according to protocol functions. For example, TCP adds a header that ensures reliable communication. Then the protocol sends the packaged data to the IP layer. IP also adds a header and sends the data to the local datalink driver.

Table 6–12 summarizes the protocols that the VAXELN Internet Services use.

Table 6–12 VAXELN Internet Services Protocols

Protocol	Description
Internet protocol (IP)	Implements mechanisms for connecting networks and gateways into a system that can deliver network packets from source to destination. This protocol routes packets to destinations through networks, keeps track of routes for hosts and networks, and accounts for incompatibilities. IP is limited to delivering datagrams, without provisions for reliability, flow control, sequencing, or other services provided by host-to-host protocols.

(continued on next page)

Table 6–12 (Cont.) VAXELN Internet Services Protocols

Protocol	Description
User datagram protocol (UDP)	Provides host-to-host datagram communication for applications that do not require streamed communication. UDP adds multiplexing to IP, letting multiple processes use the protocol to send and receive data independently. UDP is transaction oriented, and it does not guarantee delivery or duplicate protection.
Transmission control protocol (TCP)	Provides host-to-host, connection-oriented communication in a network environment. TCP adds multiplexing, checksum computations, connectivity, and reliability to IP. For data transmission, TCP uses virtual circuits. The circuits provide automatic sequencing, error control, and flow control.
Address resolution protocol (ARP)	Dynamically maps Internet addresses to physical Ethernet addresses and stores the address pairs in an ARP cache. Using this protocol, an application can determine a target host's physical (built-in) Ethernet address.
Internet control message protocol (ICMP)	Transmits error and control messages to a destination host's IP when an IP datagram delivery fails. ICMP provides routing information and notifies hosts when a datagram cannot reach its destination or does not reach its destination because a time limit expires.
Reverse Address Resolution Protocol (RARP)	Determines a diskless host's Internet address at startup so that the host can operate in an Internet network. A host can broadcast a message that specifies its physical Ethernet address to all hosts in a LAN. A host running an RARP server searches its address database and responds by returning the appropriate Internet address.
Boot protocol (BOOTP)	Determines a diskless host's Internet address at startup so that the host can operate in an Internet network. A host can broadcast a message that specifies its physical Ethernet address to all hosts in a LAN. A host running a BOOTP server searches its address database and responds by returning the appropriate Internet address. BOOTP is based on UDP.

In addition to providing the services of the Internet protocols, the VAXELN Internet Services provide runtime routines that applications can use to:

- Manage an ARP cache
- Show statistics
- Convert byte-order
- Manage an Internet routing table
- Manage Internet network interfaces
- Retrieve Internet performance and error data
- Retrieve TCP connection data
- Convert the byte order of Internet and host physical addresses
- Manipulate Internet addresses
- Communicate over an Internet using sockets
- Retrieve and set socket characteristics
- Transfer files from one host to another using FTP protocol in a heterogeneous operating system environment [1]

TELNET Server

The VAXELN Toolkit includes a TELNET server that is implemented as a terminal driver. The TELNET server lets you gain access to VAXELN systems from remote systems using TCP/IP. Once you gain access to a VAXELN system, you can type commands as if you were logged in to that system locally.

LAT Host Services

The VAXELN Toolkit includes local area transport (LAT) host services that VAXELN systems can use to communicate with devices attached to terminal servers, such as the DECserver 500. LAT is a communications protocol that lets system nodes running LAT host services communicate with dedicated terminal server nodes running LAT server services. The collection of system nodes and terminal server nodes in a LAN constitutes a LAT network.

The VAXELN LAT host services support the following:

- Terminal server communication
- Terminal I/O
- A control interface that LAT application programs can use to manage and monitor the LAT environment on a VAXELN system

[1] Supported for clients only

- An interactive utility you can use to manage and monitor the LAT environment on a VAXELN system

A VAXELN system that includes the LAT host services is a VAXELN service node. A service node can offer services to or request access to services offered by a terminal server. By default, a service node offers ECL as a service. You can access that service from an interactive terminal attached to a terminal server.

The LAT host services let application programs:

- Manage and monitor a VAXELN service node's characteristics and activities by calling VAXELN LAT utility routines
- Set up dedicated service environments
- Set up application device environments

You can initiate communication between a service node and terminal server from an interactive terminal attached to the terminal server or from an application program running on the service node. From an interactive terminal, you establish a session with a service offered by the service node. The service can be ECL or a user-created dedicated service that is built into your VAXELN system and executes as a job.

An application program running on a service node can establish a session with a remote application device or service attached to a terminal server. An application device offers a service to VAXELN service nodes in a LAT network. For example, a printer would offer printing services; a terminal device might offer display services.

A VAXELN application program can manage and monitor a LAT service node environment by calling LAT utility routines.

The LAT driver also supports a LAT Control Program (LATCP) Utility that lets you manage and monitor LAT service node characteristics and activities interactively by entering LATCP commands.

6.2.7 Authorization Service

The VAXELN Toolkit includes an optional Authorization Service that you can include in network applications. The Authorization Service protects system resources and data by maintaining a database of a system's authorized users and identifying users that issue network requests.

A target system can include local or network authorization services. When a system includes the network authorization services, it handles authorization for the nodes in a LAN that need security but do not have their own service. To use the network authorization services, at least one node in a LAN must include this service. If multiple nodes include the network authorization service and all nodes in the LAN use the same database file, one target system acts as an authorization server and manages the database while the other nodes stand by as backups. By designating multiple authorization servers, you can preserve the application's security if the acting server shuts down.

The DECnet Service, File Service, and Command Language Utility (ECL) use the Authorization Service to protect the resources and data that they control. The DECnet Service running on a particular node accepts circuit connections only from users who are listed in the Authorization Service's database. The File Service provides read, write, and delete protection for files on disks that it controls. ECL lets authorized users log into VAXELN systems. Your application programs can use the service to protect their resources and data.

6.2.8 File Service

The File Service is a set of system disk and tape driver services that lets VAXELN application programs perform file-oriented disk and tape I/O operations. The File Service consists of a disk File Service and a tape File Service and provides for access to local and remote data.

Disk File Service

The disk File Service uses FILES–11 On-Disk Structure Level 2 services and is compatible with the VMS file system and the VMS record management services (RMS). Files are organized sequentially, and programs can use sequential or random access for creating, reading, and writing sequential disk files. VAXELN does not support indexed, multikeyed, and relative file access.

The disk File Service provides a direct device access (DDA) interface for local file access. Applications can use this interface to read data from and write data to local disks block by block, avoiding the overhead incurred by the data access protocol (DAP). This interface maximizes disk performance by allowing very large data transfers. The DDA disk interface also provides for physical memory transfers by allowing applications to transfer data to and from an allocated region of virtual address space.

Tape File Service

The tape File Service is based on ANSI Standard X3.27–1978. You can use this service to transport files to and from other operating systems that implement this standard.

Remote File Access

The File Service uses DECnet to implement transparent file access across the network. Using this service, VAXELN systems can run without a local disk or tape and can gain access to files on other DECnet nodes, as if the disks or tapes were physically located on the same system.

The VAXELN file access listener (FAL) handles file access requests from remote systems. You can include the FAL in a VAXELN system that needs to provide remote DECnet file access to files on that system.

6.3 VAXELN Utilities

The VAXELN Toolkit provides development, command language, and network utility programs that you can build into your VAXELN system image and use when the system executes. (Many utilities additionally have cooperating components that execute on the VMS development system and interact with executing VAXELN applications.)

The VAXELN utilities allow you to perform the following types of operations:

- Debug and fine tune VAXELN applications, Section 6.3.1
- Issue commands to the system at a VAXELN terminal, Section 6.3.2
- Perform network functions such as LAT control and SET_HOST, Section 6.3.3

6.3.1 Development Utilities

The VAXELN Toolkit provides the following tools for debugging and fine tuning your VAXELN application:

- VAXELN Debugger (EDEBUG), supporting remote debugging over Ethernet and local debugging from the target system console
- VAXELN Performance Utility (EPC and EPA), which collects and analyzes data about application program performance
- VAXELN Display Utility (EDISPLAY), which displays system and job resource information on a video terminal
- Error logging tools, for logging hardware errors, volume changes, and system events

You might also use the following VMS and VAX tools when developing VAXELN applications:

- VMS Librarian
- VMS Error Log Utility

- Language-Sensitive Editor (LSE)
- Source Code Analyzer (SCA)
- Code Management System (CMS)
- Module Management System (MMS)
- Test Manager

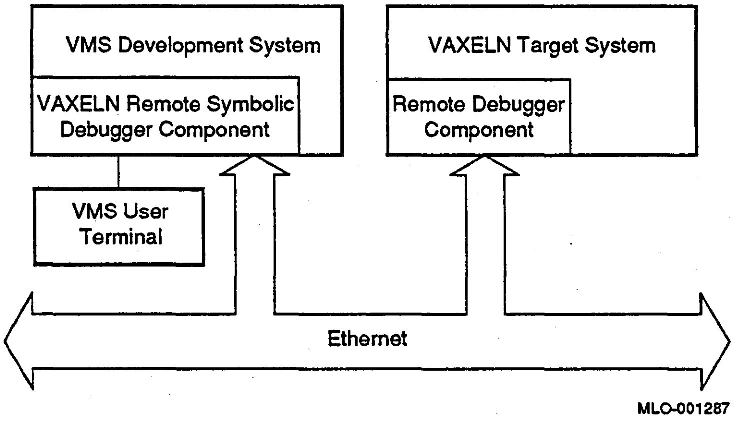
For more information about these tools, see Part VI.

VAXELN Debugger (EDEBUG)

The VAXELN Debugger (EDEBUG) supports two environments: a remote symbolic environment that you use from a development system terminal and a local environment that you use from the target system's console.

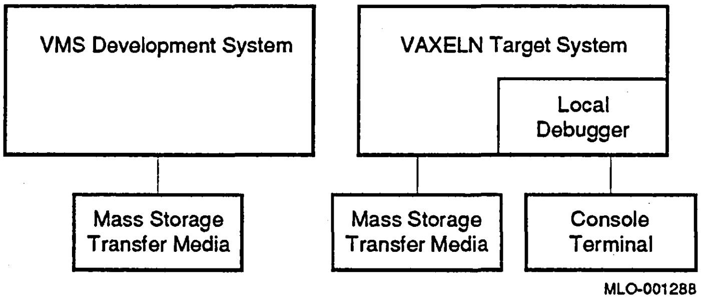
If you have a DECnet–VAX license and the appropriate Ethernet hardware, you can use the remote debugger environment. This environment lets you downline load and run your VAXELN system on a target VAX system from the VMS development system. In addition, you can access multiple VAXELN target systems simultaneously and use debug symbol table information provided by VAX compilers. Figure 6–5 shows a remote debugger environment.

Figure 6–5 VAXELN Remote Debugger Environment



If your application cannot take advantage of the remote debugging environment, you can build a self-contained local environment into your VAXELN system and debug from your target system console. The local environment does not allow symbolic debugging but provides traditional debugging functions, such as setting breakpoints and examining and depositing values in memory and registers. Figure 6–6 shows a local debugger environment.

Figure 6–6 VAXELN Local Debugger Environment



You must use the local debugger environment to debug or inspect code executing at an elevated VAX hardware interrupt priority level (IPL) (greater than 0).[1] Examples of such code include the kernel and ISRs.

The VAXELN Debugger provides complete support for modular programming. You can debug an application's components (programs or parts of programs) independently. Rather than halt the entire system when a breakpoint is encountered, the VAXELN Debugger lets all but the affected process continue running while you debug the suspected module.

The VAXELN Debugger provides a rich set of commands that you can use from a remote debugger environment, complete with command line recall. The local debugger environment supports a subset of these commands, which encompasses essentially all but the commands that access development system data.

[1] A hardware priority is not the same as a VAXELN job or process priority.

VAXELN Performance Utility (EPC and EPA)

The VAXELN Performance Utility is a tool for tuning a VAXELN system's performance. This utility helps you measure and analyze a VAXELN system's runtime behavior by isolating bottlenecks and other performance problems that affect the system's throughput.

The VAXELN Performance Utility has two components: a Collector (EPC) that you include in a VAXELN system for collecting performance data as the target system executes; and an Analyzer (EPA) that you run on a VMS development system to process the data and display tabular or histogram performance reports.

If your development and target systems are connected by DECnet–VAX, you can use the VAXELN Performance Utility to collect and analyze the following types of performance data:

- Samples of a job's program counter (PC)
- The number of times each job enters the Run state and the elapsed CPU times
- The number of times a job's processes enter the Run state and the elapsed CPU times
- The number of times a line of source code in a job was executed in a test
- The number of times a job calls each VAXELN kernel routine

You control the VAXELN Performance Utility by issuing Collector and Analyzer commands from the VMS development system.

VAXELN Display Utility (EDISPLAY)

The VAXELN Display Utility (EDISPLAY) displays VAXELN system resource information on a VAXELN or remote VMS video terminal. The utility lets you monitor system performance without including debugger support in the system. Three displays are available:

- **Setup.** Provides access to memory and job displays and a help menu. Also allows you to change the rate at which the utility updates information.
- **Memory.** Displays system-level information (such as memory, pool, and slot table usage) and identifies the system's jobs.
- **Job.** Displays statistics about a job and its subprocesses.

When you build EDISPLAY into your VAXELN system, you can specify whether the utility is to run when the system begins executing. If the utility does not start immediately, you can invoke it at runtime with an ECL command, a debugger command, or a CREATE JOB routine call from program code.

VAXELN Error Logging Tools

The VAXELN Toolkit provides the Error Logging Service, which you can build into your VAXELN system image to log errors, and the Error Logging Server (ELSE) utility, which you can run under VMS to create a remote error log file for the VAXELN system. These tools provide input to the VMS Error Log Utility, which generates error log reports.

Error Logging Service The Error Logging Service writes data that identifies hardware errors, volume changes, and system events to an error log file that exists on the local VAXELN target system or on a remote VMS system. You select local or remote error logging by editing a System Builder menu. If you select local error logging, you must transfer the log file to a VMS system for processing, using a VMS DCL COPY command or a VAXELN ECL COPY command. If you select remote error logging support, the Error Logging Service transmits error log data over the Ethernet to the VAXELN Error Log Server.

Error Log Server The VAXELN Error Log Server is invoked on the VMS development system and records error log records generated by a VAXELN target system. The server writes the records to a file on the VMS system.

You start and stop the Error Log Server by invoking supplied command procedures. By changing runtime parameters and command qualifiers in the startup command procedure, you can tailor the server to meet your application needs.

Once the error log file resides on the VMS system, you can use the VMS Error Log Utility to generate an error log report. You invoke the VMS Error Log Utility by issuing the DCL command ANALYZE/ERROR_LOG and using command qualifiers to specify various forms of error log output.

6.3.2 Command Language Utility

The VAXELN Command Language Utility (ECL) provides an interactive command-language interface to executing VAXELN target systems. ECL provides a basic set of commands and a command-line editor that you can use to display system characteristics, load and execute programs, and maintain files, disks, and magnetic tapes.

You can issue commands interactively from a target system terminal or from a remote VMS system terminal by first using the VMS command SET HOST. As part of your ECL session, you can invoke command files (including LOGIN.COM) and define symbols equivalent to often-used commands.

The ECL Utility does not furnish an environment for program development, such as that provided by the VMS Digital Command Language (DCL); you develop VAXELN systems in a VMS environment.

6.3.3 Network Utilities

The VAXELN Toolkit provides the following network utilities:

- LAT Control Program (LATCP), which allows you to interactively manage and monitor LAT service node characteristics and activities
- Outbound Remote Terminal Utility (SET_HOST), which lets you connect to a remote Digital system from a VAXELN terminal session

LAT Control Program (LATCP)

The LAT Control Program (LATCP) provides a set of commands that you can use to control and monitor VAXELN LAT service node characteristics and activities at runtime. You can issue commands interactively from a terminal attached to a terminal server.

When you build the LATCP Utility into your VAXELN system, you can specify whether the utility is to run when the system begins executing. If the utility does not start immediately, you can invoke it at runtime with an ECL command, a debugger command, or a CREATE JOB routine call from program code.

Outbound Remote Terminal Utility (SET_HOST)

The Outbound Remote Terminal Utility (SET_HOST) allows you to connect to a remote Digital system, such as VMS, from a VAXELN terminal session. For example, you can set host to a VMS terminal, using the ECL command SET HOST, and enter commands as if you were logged into VMS locally. Both the target system and the remote system must be running DECnet, and the remote node must support inbound remote access.

VAXELN systems can also be configured to accept inbound remote access; for example, you can SET HOST from a VMS terminal session to a VAXELN system and enter commands as if you were a VAXELN terminal user.

Both forms of remote access, outbound and inbound, can be included in your VAXELN system using the System Builder.

7 VAXELN DECwindows

VAXELN DECwindows (which is based on Release 4 of MIT's X Window System) incorporates the Open Software Foundation (OSF) Motif interface into the DECwindows product, providing powerful software to simplify the development of graphics-oriented applications. DECwindows Motif contains several programming layers:

- **Xlib.** Xlib, the lowest level interface to the X Window System, is a graphics library used to manipulate screen displays and to handle events.
- **X Toolkit.** The X Toolkit (also called Intrinsics) is a library of routines that creates and manipulates interface objects called widgets.
- **Motif Toolkit.** The Motif Toolkit provides a library of routines that create the Motif widget set. A Motif Window Manager (MWM) and Motif User Interface Language (UIL) compiler are part of Motif.
- **Digital extensions.** Digital extensions to Motif include widgets and convenience routines to manipulate widgets. Digital also provides non-C bindings for Ada, C, and FORTRAN programming.

The VAXELN Toolkit increases its versatility and enhances distributed processing by supplying DECwindows support. Using VAXELN and DECwindows software in concert, you can develop dedicated, realtime applications that feature network-transparent, graphics-oriented user interfaces. DECwindows also lets you apply graphics, colors, and fonts to increase application usability.

Like the VAXELN Toolkit, DECwindows lets you take advantage of the power and resources of other systems by distributing applications and application tasks among nodes in a network. You can run an application on a remote node, while the application displays output and you enter input on a local workstation. Network activity is transparent, and the application appears to run locally.

VAXELN applications that can benefit from DECwindows support are those that include a human interface, such as a monitor/control station. For example, a DECwindows interface for a process control application might display windows from which you can start various systems, display schematics of processes being controlled, and show graphs representing acquired and computed data. The system may also display alarms and warning messages when human intervention is necessary.

This chapter discusses the following:

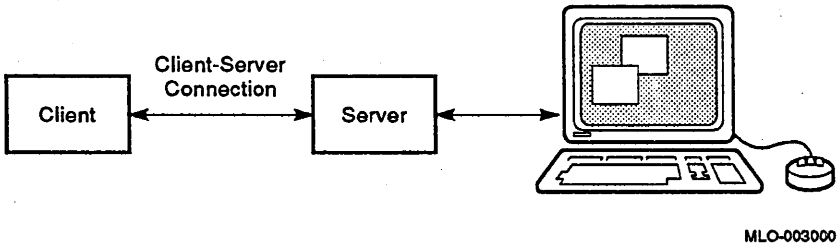
- DECwindows Architecture, Section 7.1
- DECwindows User Environment, Section 7.2
- VAXELN DECwindows Server, Section 7.3
- VAXELN DECwindows Applications, Section 7.4
- VAXELN DECwindows User Environment Components, Section 7.5

For additional sources of information on VAXELN DECwindows, refer to Appendix A.

7.1 DECwindows Architecture

The DECwindows architecture consists of one or more application programs, called clients, and a server. A client and the server communicate by sending data over a connection that the client establishes. Clients use an application programming interface to initiate, control, and monitor client-server communication. Figure 7–1 shows the DECwindows architecture.

Figure 7–1 DECwindows Architecture



The server provides a common means for clients to interact with graphics workstations. A server can handle multiple client-server connections; the server displays the output for each client in a separate application window.

The server receives graphics output requests from the client and sends replies, events, and errors to the client. Replies inform the client of the results of an output request. The server sends events to the client in response to keyboard or mouse input, side effects of client requests, and declarations from other clients that need to share data. Also, the server reports errors that occur during request processing.

A client can reside on the same node as the server or on a remote network node. In both cases, communication is transparent, and the local workstation on which the server runs displays the client's output. Thus clients that run on remote nodes appear to run on the local workstation.

The VAXELN Toolkit supplies the following DECwindows software:

- A DECwindows server that you can build into VAXELN systems
- The DECwindows runtime libraries and tools you need to develop VAXELN DECwindows clients that you can build into VAXELN systems
- DECwindows user environment components that you can build into VAXELN systems

Note

To use some VAXELN DECwindows features, your host system must be running VMS Version 5.4 or later and VMS DECwindows Motif must be installed. Also, it is assumed that you have access to the VMS DECwindows Motif documentation.

7.2 DECwindows User Environment

The graphics-oriented user interface that characterizes DECwindows environments lets you do the following:

- Tailor your own user environment
- Perform tasks by manipulating objects on the workstation screen using a pointer (mouse or tablet)
- Interact with applications in a consistent, transparent manner from workstations running any server that supports the DECnet or Internet protocol and the X Window System
- Use accelerators for interacting with applications more quickly
- Interact with an application that is distributed among network nodes from a single workstation screen

- Interact with multiple applications (local and distributed) from a single workstation screen

User interfaces for DECwindows applications look alike and respond to user input similarly. The interfaces consist of objects such as menu bars, scroll bars, pop-up menus, pull-down menus, and buttons. You use a pointer (mouse or tablet) to manipulate these objects and select text to perform various tasks. For example, you might select a button to open an application window that displays a graph based on data that the application is collecting.

Using DECwindows, you can communicate with applications that are distributed among network nodes from a single workstation screen. For example, you can interact with a compute-intensive application that runs on a remote VAX 6000 processor from one window, while you edit a file that resides on the local workstation in another window.

Similarly, you can distribute a single multiprocessing application among network nodes and communicate with the application components through different windows on a workstation screen. Consider a VAXELN loosely coupled multiprocessing application in which jobs are distributed across three VAXELN systems. A subset of the application's jobs might be distributed as follows:

System	Processor	Job Task
A, B, and C	VAX 4000 VLC	Acquire status data
D	VAX 6000	Compute the data
E	VAXstation 3100	Report the computation results

You could interact with each of the VAXELN systems from windows displayed on the workstation. Table 7–1 shows what the preceding jobs might display in various windows and the type of user interface they might provide.

Table 7–1 Sample DECwindows User Interface

Job Task	User Interface	Output	User Interface
Acquire data.	Select system A, B, or C from a pull-down menu.	Displays a layout of the device configuration from which the job is acquiring the status data.	Not applicable.
Compute data.	Not applicable.	Displays alarms and warning messages when computations identify conditions that require human intervention.	Correct the condition; then inform the system that the condition is corrected by selecting a toggle button.
Report results.	Select a graph type and the system whose results you want to view by clicking on option buttons.	Displays computation results in the form of graphs.	Not applicable.

7.3 VAXELN DECwindows Server

The VAXELN Toolkit provides a DECwindows server that runs as a job on the hardware where the graphics display, keyboard, and pointer are located. You can build the server into VAXELN systems that run on a subset of the VAXELN supported target workstations.

The DECwindows server supports one display screen, one keyboard, and one pointer per workstation for performing graphics, windowing, and input functions.

The VAXELN DECwindows server consists of components that handle client-server connections and provide interfaces for workstation devices. The server also manages fonts (memory-resident or on-disk), keymap files, and security access files.

7.4 VAXELN DECwindows Applications

The VAXELN Toolkit supplies the runtime libraries and development tools you need to develop VAXELN applications that employ the capabilities of graphics workstations. Applications can perform windowing and graphics functions or get input from the keyboard or pointer by calling routines from Xlib and Motif Toolkit programming libraries.

Xlib is the low-level X Window System interface that DECwindows applications can use to create windows, manage windows, and perform graphics functions.

The Motif Toolkit is layered on top of Xlib and consists of high-level library routines that applications can use to create and manage a DECwindows user interface. Using Motif Toolkit routines and development tools, applications can create, modify, and control interface objects, such as buttons, menus, and scroll bars. VAXELN DECwindows Motif includes Digital's implementation of the X Toolkit for the X Window System and provides the mechanisms you need to create a DECwindows user interface that conforms to the Motif style. The Motif Toolkit consists of the following components:

- User interface objects and runtime routines that create those objects
- X Toolkit routines that manipulate the user interface objects
- Cut and paste routines that you can use to copy data between applications
- Application development tools, including the User Interface Language (UIL) and Motif Resource Manager (MRM)

Applications written in Ada, FORTRAN, or VAXELN Pascal can call Xlib and Motif Toolkit routines by using the standard VAX calling format. Applications written in C can call the routines by using the VAX or MIT C calling format.

Note

VAXELN DECwindows Motif has limited support for programming in VAXELN Pascal. VAXELN Pascal language bindings are provided for Xlib and the XUI Toolkit routines. The VAXELN Pascal language bindings are not supported for the Motif Toolkit.

7.5 VAXELN DECwindows User Environment Components

The Motif Window Manager and terminal emulators are part of VAXELN DECwindows support. The Motif Window Manager manages the location and size of an application's main windows. You can use the Motif Window Manager to control the appearance of windows on your workstation screen.

The terminal emulators display a window that looks and functions like a terminal. That is, a terminal emulator window can perform application terminal I/O and can provide a command line interface. If your system is part of a network, you can use a terminal emulator window to communicate with other systems on that network.

The VAXELN Toolkit supplies terminal emulators (a console emulator and a VT300-series terminal emulator) that can handle terminal I/O for VAXELN DECwindows applications running on the local workstation or on remote systems in the local area network (LAN). The console emulator displays a window that provides minimal terminal I/O functionality. The VT300-series terminal emulator displays a window that looks and functions like a VT300 series terminal.

8 VAXELN Window Server

The VAXELN Window Server (EWS) software is a product for setting up a DECwindows environment on Digital's workstations, VT1300 terminals, and VAXELN Window Stations. EWS software offers a DECwindows solution for diskless workstations and, depending on your configuration, can provide improved windowing performance.

Digital's VAXELN Window Station is the price-performance leader in the high-resolution color X Window terminal market. The VAXELN Window Server software provides the DECwindows functionality.

This chapter discusses the following:

- VAXELN Window Server Overview, Section 8.1
- VAXELN Window Server Features, Section 8.2
- VAXELN Window Station, Section 8.3
- VAXELN Window Server Development Software, Section 8.4
- VAXELN Window Server Hardware and Software Requirements, Section 8.5

8.1 VAXELN Window Server Overview

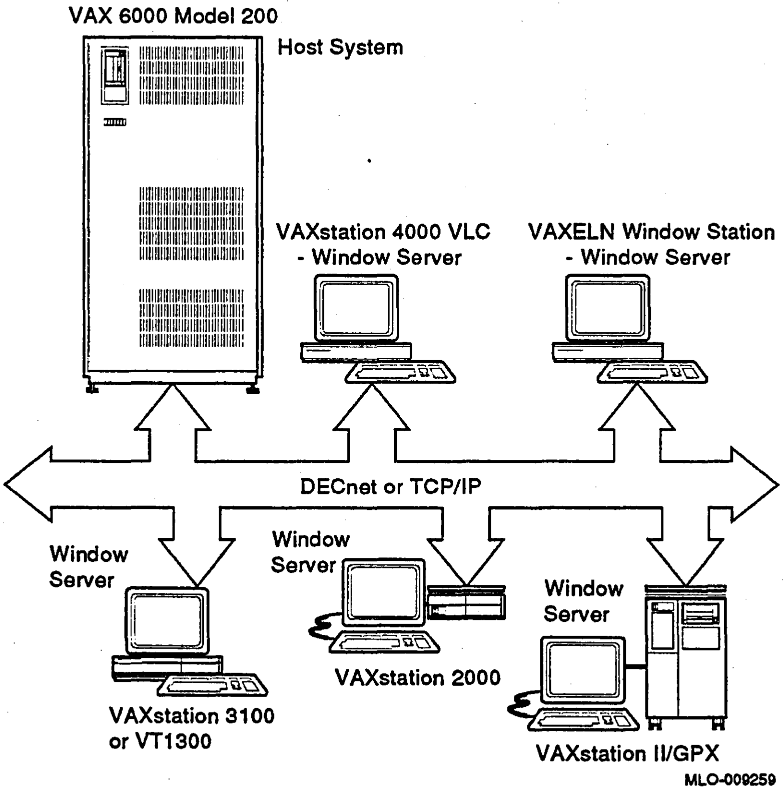
The VAXELN Window Server software is an ULTRIX and VMS layered product that loads a VAXELN system image containing a DECwindows server to a target VAXstation. These images, downline loaded, also set up an environment in which a host VMS or ULTRIX system provides DECwindows applications back to the workstation. Since VAXELN supports the full range of Digital workstations, EWS can be used by both older and newer VAXstations.

The VMS or ULTRIX host system in a VAXELN Window Server configuration downline loads the appropriate VAXELN Window Server system images to the VAXstation via Ethernet. Once the image is loaded, that device becomes an X Window Terminal and acts as the display device for DECwindows applications that run on a central host system.

A single host system can support multiple window servers. Figure 8–1 shows a sample VAXELN Window Server configuration consisting of a VAX 6000 Model 200 host system and the following window servers:

- A VAXstation 4000 VLC
- A VAXELN Window Station
- A VAXstation 2000 workstation
- A VAXstation 11/GPX workstation
- A VT1300 terminal

Figure 8–1 Sample VAXELN Window Server Configuration



DECwindows is an implementation of the industry-standard X11 protocol which provides powerful graphics software that can simplify many typical tasks. For further information on DECwindows and the X Window System, refer to Chapter 7.

8.2 VAXELN Window Server Features

The VAXELN Window Server software has these key features:

- **Customizable environment.** Users may customize (rebuild) the EWS configuration by rebuilding the EWS system images using the VAXELN Toolkit. These customized images, which can include local DECwindows clients or custom software, are then downline loaded to the target workstation or X Window terminal. Customized software may be dedicated to a specific task, such as process control, or it may be used in interactive sessions to a host computer.
- **DECnet and TCP/IP support.** The communication protocol between the host system and the workstation or VT1300 terminal can be DECnet, TCP/IP (transmission control protocol/internet protocol), or both.
- **Supports a full range of Digital VAXstations.** EWS allows users to include many older, minimum-memory platforms, such as the VAXstation 2000, to be brought online as Windowing Stations.
- **User selectable login.** EWS allows users to select a node on a network from which the session will run. This menu list can include any mix of VMS DECnet, ULTRIX DECnet, and ULTRIX TCP/IP nodes.

8.3 VAXELN Window Station

The VAXELN Window Station combines the existing VAXELN Window Server Software (EWS) and the high-performance, low-cost VAXstation 4000 VLC workstation. The customer chooses an appropriate monitor, keyboard, and mouse. This packaging offers flexibility which can meet the requirements of most any situation and allows for use of third party products which have been developed for EWS (such as a Touch Screen monitor).

The VAXELN Window Station has these key features:

- The EWS software downline loads a VAXELN system image providing X Window Terminal functionality as an alternative to a fully functional workstation.

  System management becomes a centralized function, local to the host computer. Basckups, for example, are not necessary on an X Window Terminal.

- The VAXELN Toolkit allows you to customize the windowing environment.
- Since the VAXELN Window Station can also run VMS, it can be easily upgraded to a workstation.
- The VAXstation 4000 VLC is diskless and comes with a standard 8 Mbytes of memory, optionally expandable to 24 Mbytes.
- Both TCP/IP and DECnet communication protocols are supported, making it easy to access multiple operating systems supporting the X protocol.

The VAXELN Window Station gives you access to applications running on different operating systems and hardware platforms from anywhere across the network. Multiple window applications can be displayed simultaneously whether running under VMS, ULTRIX, or any other operating system suporting the X Window Protocol.

8.4 VAXELN Window Server Development Software

The VAXELN Window Server is an example of a product that was developed using the VAXELN Toolkit. While another operating system could have been chosen, the characteristics of VAXELN were a factor in selecting the VAXELN Toolkit.

Typically, X Window terminals boot quickly, are easily upgraded, and may cost less per seat than full workstations. Since customers have grown to expect these characteristics in X Window terminals, the selection of suitable software was critical to the success of the VAXELN Window Server product.

VAXELN was selected as the software of choice for developing the VAXELN Window Server product because it gives EWS these key capabilities and characteristics:

- **Boot quickly off the network.** The Maintenance Operation Protocol (MOP) allows VAXELN systems to boot quickly off a network. The operating system is booted from a host node and DECwindows applications can be operational in a matter of moments.
- **Software easily updated.** VAXELN's downline load capability affords easy software updates. The server node is updated with nwe VAXELN system images and the EWS workstations simply reboot off the network to take advantage of new software. There is no ROM software for EWS.
- **Software easily customized.** The VAXELN Toolkit allows you to customize the system image, tune memory, or specialize the software. The VAXELN Toolkit is a sophisticated, mature, full-featured toolkit which can be used to adapt EWS to almost any environment. This feature is unique to the X Window Terminal industry.

- **Low memory requirements.** The minimum memory required for running the VAXELN Window Server is 6 Mbytes, which reduces the cost per seat for VAXELN Window Server operation.
- **Memory-resident fonts.** The ability of VAXELN to store fonts in physical memory rather than on a disk drive is important for an X Window Terminal.
- **Support the full range of workstations.** VAXELN is a full implementation of the VAX architecture, which allows support for the entire range of Digital VAXstations, not just X Window Terminal products. Older VAXstations with limited memory may be added to the network as EWS workstations. For example, while a VAXstation 2000 with only 6 Mbytes of memory may not have enough power or memory to serve as a standalone system, it can comfortably run EWS.
- **Use VAXELN transport mechanisms.** Data is moved using standard VAXELN messaging and shared memory, making VAXELN Window Server software fast and dependable.

8.5 VAXELN Window Server Hardware and Software Requirements

This section identifies the VAXELN Window Server hardware and software requirements.

For a detailed list of host and window server processors (VAX and MIPS-based) supported by the VAXELN Window Server software for VMS and ULTRIX and the software required to down-line load and run the VAXELN Window Server software, refer to the *System Support Addendum (SSA)* for the VAXELN Window Server for VMS and ULTRIX, respectively. You may obtain these from your Digital sales representative.

VAXELN Window Server Hardware Requirements

A system configuration for installing and running the VAXELN Window Server software requires the following hardware:

- A system to serve as the EWS host system that runs DECwindows client applications
- A Digital workstation, VAXELN Window Station, or VT1300 terminal
- An Ethernet connection between the VAXELN Window Server host system and a target workstation or terminal

VAXELN Window Server Software Requirements

This section outlines the software required to down-line load and run the VAXELN Window Server software for VMS and ULTRIX.

VMS Requirements A VMS host system requires the following software:

- VMS operating system with DECwindows application support
- For the DECnet protocol environment: DECnet–VAX networking software
- For the TCP/IP protocol environment: VMS/ULTRIX Connection (UCX)

If you choose to rebuild your VAXELN Window Server system images, you will also need the VAXELN Toolkit.

ULTRIX Requirements An ULTRIX host system requires the following software:

- ULTRIX operating system
- ULTRIX X11 DECwindows user environment
- Maintenance Operation Protocol (MOP)
- DECnet—ULTRIX *or* TCP/IP networking software

Part III
VMS Systems in Realtime Applications

Part III surveys the VMS operating system as a development and runtime environment for realtime applications and introduces the product VMS POSIX, which can aid in realtime application development on VMS systems. This part contains the following chapters:

- Chapter 9, Survey of VMS Realtime Capabilities, outlines the VMS operating system realtime services and some VMS realtime application programming techniques.
- Chapter 10, outlines the functionality provided with the VMS POSIX product. VMS POSIX allows application developers to write applications that contain functions that contain POSIX 1003.1, P1003.2/D10, and P1003.4/D9 functions.

9 Survey of VMS Realtime Capabilities

The VMS operating system is a full-service operating system widely regarded as the premier software development environment in the industry. It also features all the services required of a realtime operating system. A VMS system can be tuned to meet the performance requirements of many of the most demanding realtime applications at runtime. Tuning involves shutting down unnecessary system processes (for example, accounting, error logger, batch queues, network servers), locking the realtime process into physical memory to prevent paging, and other techniques.

This chapter covers the following topics:

- VMS Realtime Services, Section 9.1
- Programming Techniques for VMS Realtime Applications, Section 9.2
- VMS POSIX for VMS Realtime Applications, Section 9.3
- Software Tools for VMS Realtime Applications, Section 9.4

For additional information on VMS realtime capabilities, refer to the *Realtime User's Guide*. Additional sources of information are listed in Appendix A.

9.1 VMS Realtime Services

The VMS operating system offers these realtime services:

- **Realtime scheduler.** The VMS operating system allows for 32 levels of process priority. The higher 16 priorities are reserved for realtime processes. At these higher priority levels, the scheduler is a true preemptive, priority-based scheduler. Processes that are assigned these realtime priorities undergo neither time-slice scheduling nor automatic priority adjustment, unlike timesharing processes. System services are available to alter process priority dynamically.

- **Memory management services.** Critical sections of code or data can be made resident in physical memory to guarantee availability and eliminate the device overhead associated with paging. Also, the privileged user has complete control of the allocation of physical memory resources through the management of process quotas.
- **Interprocess communication.** The VMS operating system offers several methods of communication between processes, including:
  - **Common event flags.** Common event flags provide a simple, convenient means for event notification. Cooperating processes can set, clear, and wait for flags in a common event flag cluster.
  - **Global sections.** A global section is an area of physical memory that is contained within the virtual-address space of multiple processes. The VMS operating system furnishes services for dynamically creating, mapping, and deleting shared global sections. These services allow several processes to use shared memory for common data pools or interprocess message passing. Common event flags can be used to synchronize access to shared memory.

    In many realtime applications (such as data acquisition or industrial process control), response time is so critical that control variables and data readings must remain in memory. Frequently, many processes must use this data simultaneously. Global sections supply a convenient mechanism for fast data access and for rapid data passing from one process to another.
  - **Mailboxes.** The VMS operating system provides mailboxes, which are memory pseudodevices (virtual devices) for interprocess communication. The VMS operating system supports drivers for pseudodevices, including the null device (NL:), network device (NET:), remote terminal device (RT:), and mailbox (MB:). You can assign channels to these devices and issue I/O requests just as though they were real devices.

    Processes can send messages or other data using mailboxes that cannot be conveyed by the simpler and faster operations of setting and clearing event flags. Mailboxes can hold multiple messages that are read on a first-in-first-out (FIFO) basis. In contrast, the current status of an event flag does not indicate how many times it has been set or cleared.

    Mailboxes entail some overhead. To pass and read messages faster, you can use a global section (discussed previously) to hold the messages and common event flags to notify processes that messages are ready to be read.

— **DECnet.** DECnet, Digital's networking software that runs on nodes in local and wide area networks (LANs and WANs), enables task-to-task communication. Using DECnet, a logical communication link can be created between two programs for the purpose of interprocess communication. The two programs can be running under the same or different operating systems.

— **Asynchronous System Trap Service Routines.** An asynchronous system trap (AST) is a software-simulated interrupt used for event notification within a process. An AST service routine is a user-written routine that receives control when an AST is delivered after being queued to the process.

The AST interrupts the process execution flow as soon as no higher priority process is executable, unless specific conditions temporarily prevent delivery. When the AST service routine completes, the current image continues executing from the point at which it was interrupted. ASTs are thus a mechanism to allow asynchronous operations.

— **Hibernation and Suspension.** Hibernation and suspension (accessible through separate VMS system service calls) are two synchronization mechanisms that allow a process to control when it or another process becomes active. Both hibernation and suspension temporarily halt process execution. A process can place itself in the hibernate state, or place itself or other processes in the suspend state.

— **Shareable Images.** Shareable images can be used to share frequently used code or data among multiple processes. A shareable image might contain routines common to several programs. If a shareable image is installed in the system as a permanent global section (as is normally the case), other programs can link with it and share its contents.

— **Lock Manager.** Realtime applications often run as a group of processes that share a common database. The VMS lock management services supply a mechanism that lets cooperating processes synchronize their access to shared resources.

The lock management services allow a group of processes to associate a name with a resource and then individually request access to the resources using that name. If the resource is locked, the request is placed in a queue. When the resource is free, the request is granted. A process that requests a lock must specify a lock mode; the lock mode designates how the process wants to share the resource with the rest of the group. Processes can change the lock mode assigned to a resource by requesting a lock conversion.

  - **Privileged Shareable Images.** A privileged shareable image is a shareable image containing one or more routines that nonprivileged users can call to perform privileged functions. The creator of the privileged shareable image codes, compiles or assembles, links, and installs the routine. Other users can then call this routine in their programs, provided they have linked their object module(s) with the privileged shareable image.

    Privileged shareable images thus offer a vehicle for you to write and use your own system services. These shareable images furnish a suitable vehicle for special-purpose routines that nonprivileged processes in realtime applications can use.

- **Interprocess synchronization.** As mentioned earlier, the VMS operating system supplies event flags and locks for interprocess synchronization and signaling. Common event flags can be used to synchronize access to shared physical memory and to notify a process of event completion (for example, notification that an I/O operation is complete). Locks allow shared multiprocessor access to shared devices or memory.
- **Symmetric multiprocessing.** The VMS operating system offers true symmetric multiprocessing (SMP) support for multiprocessor VAX systems. VMS SMP configurations consist of multiple central processing units (CPUs) executing code from a single memory address space. All processors share a single copy of the VMS operating system. Any processor can request I/O as well as execute computational tasks. This allows you to make maximum use of multiprocessor configurations.
- **Low overhead I/O.** A process can map a portion of its address space to the I/O page and directly access I/O module control/status registers (CSRs). This access to I/O module CSRs can be accomplished by using high-level language assignment operations or assembler-level instructions.

  Access to devices by drivers is accomplished by using the queued input/output (QIO) system service. QIO, in conjunction with a device driver, allows for asynchronous direct memory access (DMA) or interrupt-driven I/O.

9.2 Programming Techniques for VMS Realtime Applications

The following VMS programming techniques optimize realtime application development and performance:

- **Read and write to device registers directly.** If you are a privileged user, you can read from and write to device registers directly from a program, bypassing the I/O subsystem altogether. For example, during polled I/O (also known as memory-mapped I/O), a synchronous routine call maps directly to the I/O page of the VMS operating system and reads from or writes to the CSR of the device. This provides the least software overhead between your program and the I/O device.
- **Use QIO system service to access device drivers directly.** You can access device drivers directly by using the QIO system service to perform synchronous or asynchronous I/O. The QIO system service is most suited to continuous I/O using asynchronous I/O routine calls and multiple buffers, or to I/O through DMA-driven devices.
- **Use file creation methods that optimize disk I/O.** You can optimize disk I/O for realtime access by creating files in contiguous blocks or in specific regions of a disk volume. This will minimize head-movement overhead.
- **Use connect-to-interrupt facility to service device interrupts.** You can use the connect-to-interrupt facility to service device interrupts without writing a full device driver. A process with suitable privileges can connect to a device interrupt vector using the VMS connect-to-interrupt facility. A process normally uses this facility for devices that do not have VMS drivers. Such devices cannot be used in a DMA mode and must be attached to the Q22–bus or UNIBUS.

  Connecting to a device interrupt vector allows you to:

  - Respond to an interrupt within a very short time
  - Preempt other system processing to handle a realtime event (for example, a clock interrupt)
  - Buffer data from a device in realtime and return the data to the process at a later time
  - Set an event flag or queue an AST (or both) to notify your process that an interrupt has occurred

9.3 VMS POSIX for VMS Realtime Applications

VMS POSIX is a software product that allows you to develop and run portable applications in a POSIX environment. The VMS POSIX product includes an interactive interface and a programming interface, based on the POSIX standards and draft standards. The POSIX standards and drafts support the concept of open systems.

The VMS POSIX product includes support for the following POSIX standards and draft standards:

- POSIX 1003.1–1990
- P1003.2/D10
- P1003.4/D9
- P1003.1a/D4
- P1003.2a/D5 (UPE)

VMS POSIX also conforms to the XPG3 BASE specifications. VMS POSIX has passed the Verfication Suite (VSX3) test suite and has been branded by X/Open as conforming to the BASE specifications. The Verfication Suite consists of over 5000 tests that test the implementation of the XPG Common Applications Environment (CAE), which consists of internationalization system calls and functions, commands and utilities, and the C language as implemented on a specific hardware platform.

Standard	Description
POSIX 1003.1	A set of functions and calls for programs using the C programming language. Using the POSIX 1003.1 standard in an application promotes application portability among systems supporting the POSIX 1003.1 standard. The functions and calls in POSIX 1003.1 include some functions that are identical to ANSI C functions, some functions that use the same syntax as ANSI C functions but operate slightly differently in the POSIX environment, and some functions that are unique to POSIX.
P1003.2/D10	A set of commands and utilities that provide functions at an interactive level. For this version of VMS POSIX, Draft 10 of P1003.2 has been used. The P1003.2/D10 utilities provide an interface based on the most popular UNIX commands and utilities. A VMS user may notice that some commands are similar to DCL-level commands in the VMS operating system, and other commands and utilities are unique to the POSIX environment.

Standard	Description
P1003.4/D9	A set of functions and calls for designing and creating realtime applications in the VMS POSIX environment.

The VMS POSIX realtime environment offers these POSIX features:

- **Process synchronization.** VMS POSIX functions offer several methods for synchronizing processes, including:
  - **Binary semaphores.** A binary semaphore is a synchronization mechanism used to control access to systemwide resources.
  - **Realtime clocks and timers.** Realtime clocks and timers allow an application developer to synchronize and coordinate activities according to a predefined schedule.
  - **Priority scheduling.** Priority scheduling gives an application programmer control over the execution sequence of the processes comprising an application. Priority scheduling also addresses the need for a realtime process to execute when it needs to and for as long as it needs to. VMS POSIX supports three scheduling policies: first-in first-out (FIFO), round-robin (RR), and timesharing (OTHER).
- **Interprocess communication.** VMS POSIX functions offering interprocess communication include:
  - **Event notification.** Event notification is a way of passing data within an application when the application must respond to an application-defined occurrence. Events are associated with event classes to define groups of events so that you have more control over operations. In addition, event masks block or unblock event notification.
  - **Message queues.** A message queue is a systemwide special file accessible by a pathname. Message queues remain after a child process terminates and can be reused by other processes. VMS POSIX message functions allow you to define the size of the message, receive messages selectively, and send or receive messages asynchronously.
  - **Shared memory.** Shared memory allows multiple processes to share data by mapping a region of memory into each process's address space. Use of shared memory among processes provides faster data access.

    You can allocate as little memory as possible for each process or allocate one large region and manage it within the application. Shared memory can be created as persistent or nonpersistent, mapped or unmapped, linked or unlinked, or closed.

- **Performance.** Access to files and memory are major factors affecting the speed of program execution. A realtime application designer should consider the importance of execution speed versus file and data integrity. VMS POSIX provides several methods of controlling application performance, including:
  - **Process memory locking.** In a non-realtime environment, the system swaps processes in and out of memory depending on system load and resources. However, realtime processes require an upper limit on the amount of time required to fetch data. Locking a process's address space in memory eliminates paging and swapping by the target process. VMS POSIX memory-locking functions allow an application to lock the entire address space, and the stack segment, and to perform region and lock stacking.
  - **Asynchronous I/O.** Realtime applications require input and output operations that do not block the initiating process. With VMS POSIX, asynchronous I/O read and write operations are queued and the initiating process continues execution.
  - **Synchronized I/O.** Synchronized input and output ensures data and file integrity during read and write operations. A synchronized input operation ensures that a read operation from a device transfers the current image of the data from the device. A synchronized output operation blocks the initiating process from executing and does not return until the write operation is complete.
  - **Realtime Files.** A realtime application relies on the file management system to provide bounded response times to whatever external demands are placed on the application. The file system must ensure a high level of performance, access to resources, and the delivery of data to the media. VMS POSIX realtime files allow for this capability. Note, however, that VMS POSIX does not support direct I/O.

9.4 Software Tools for VMS Realtime Applications

You can use the VMS operating system for many realtime applications requiring the full range of available services found in a general-purpose computing environment. A number of available software development products complement the VMS operating system.

9.4.1 VMS Error Log Utility

The VMS Error Log Utility, bundled with the VMS operating system, is a system-management tool that selectively reports the contents of one or more error log files. This utility supports most VMS system-supported hardware, including disks, tapes, CPUs, and memory.

9.4.2 VMS Librarian

With the VMS Librarian, you can maintain libraries of object modules and shareable images. This tool is also packaged with the VMS operating system and can:

- Create new libraries
- Insert or replace modules in an existing library
- List a library's contents
- Extract modules from a library
- Delete modules from a library
- Compress a library

9.4.3 VMS Linker

Another tool that is bundled with the VMS operating system is the VMS Linker. With this tool, you can combine object modules and shareable images into one of three types of images:

- Executable image
- Shareable image
- System image

You can control the linkage of your images by specifying qualifiers with the Digital Command Language (DCL) LINK command. In turn, the qualifiers instruct the VMS Linker to perform operations such as:

- **Resolve symbolic references.** The VMS Linker maintains a global symbol table (GST). Here, it stores the name and definition of every global symbol. To resolve a symbolic reference, the linker searches its GST for a definition of the symbol.
- **Limit the search scope in resolving symbolic references.** The VMS Linker can be directed to suppress the search for undefined symbolic references through default system libraries and the default shareable image library.

- **Allocate virtual memory.** The VMS Linker can place program segments in memory locations that best meet program and memory-management requirements.
- **Initialize an image.** After the VMS Linker resolves references and allocates virtual memory, it initializes the image by:
  - Filling it with the compiled binary data and code
  - Inserting addresses into instructions that refer to externally defined fields
  - Computing values that depend on externally defined fields
- **Generate debugger information.** The VMS Linker can generate a debug symbol table and give control to the VMS Debugger when the image is run.

10 VMS POSIX Realtime Programming

VMS POSIX is a software product that allows you to develop and run portable applications in a POSIX environment. The VMS POSIX product includes an interactive interface and a programming interface, based on the POSIX standards and draft standards.

An application is considered to have portable code when the same source code can be successfully compiled, linked, and run on more than one system. In the VMS POSIX environment, a program is considered portable when it conforms to the same POSIX standards and drafts supported by VMS POSIX. The advantages of portability are clear; however, tradeoffs may be required.

This chapter describes the following aspects of the VMS POSIX environment:

- VMS POSIX Programming Environment, Section 10.1
- VMS POSIX Commands and Utilities, Section 10.2
- VMS POSIX Realtime Environment, Section 10.3

For additional information on the VMS POSIX environment, refer to the *Guide to Using VMS POSIX*. Additional sources of information are listed in Appendix A.

10.1 VMS POSIX Programming Environment

On a VMS system where VMS POSIX is installed, you have the choice of logging in to either the VMS environment or the VMS POSIX environment. You may also switch between these two environments within a single login session. In the VMS POSIX environment, you must use only those commands supported by the POSIX environment. This environment uses the POSIX 1003.1 standard to promote and allow for portability between systems that support the POSIX 1003.1 standard.

The POSIX 1003.1 standard defines an operating system interface and standardized services through the C programming language. The following major categories are covered by the POSIX 1003.1 standard.

— **Process primitives.** Process primitives define process creation, execution, and termination. Signals and timers are defined as a means of synchronizing process execution and interprocess communication.

— **Process environment.** Process environment commands allow you to set and retrieve user and group IDs. In addition, commands are used for system identification, setting time, accessing environment variables, identifying terminals, and configuring system variables.

— **Files and directories.** File commands allow you to create, open, link, set a mask, rename, or remove files and special files. You can set file characteristics, such as file accessibility and status. Directory commands allow you to perform directory operations, such as creating or removing a directory.

— **Input and output primitives.** Input and output primitives include standard read and write operations as well as pipes and control operations on files.

— **Device-specific and class-specific functions.** Device-specific and class-specific functions include functions that allow you to communicate with devices and device files.

— **Language-specific services for the C programming language.** Language-specific services define extensions to C language functions.

— **System databases.** System database functions create and establish access to system databases.

— **Data interchange format.** Data interchange format commands specify archive and interchange file formats.

The VMS POSIX environment is similar to a UNIX environment. Many commands are identical to those of UNIX. Users accustomed to a VMS environment will find that many of the commands are similar, but will want to consult *Guide to Using VMS POSIX* for a comparison of the commands and how they function.

A VMS POSIX application can use all features available in the VMS operating system, with some restrictions. For example, the following VMS features can be used freely and do not affect your application's portability:

- VAXcluster environments
- Symmetric Multiprocessing (SMP)

- Disk shadowing

The following programming practices can be used but may affect the portability of your application:

- Linking a VMS POSIX application with modules written in languages other than C
- Using VMS file specifications
- Calling VMS system services

The following features are restricted in VMS POSIX applications:

- RMS journaling
- Recovery unit journaling

VMS POSIX provides a VMS POSIX runtime library and POSIX header files so that you can compile and link your application in the VMS POSIX environment.

You can use VMS HELP from the POSIX shell, which provides the full range of online help for VMS. If your system provides access to the DECwindows Bookreader, you can also obtain information about VMS POSIX in the DECwindows Bookreader.

10.2 VMS POSIX Commands and Utilities

The P1003.2/D10 draft standard is a set of commands and utilities that provide functions at an interactive level. The VMS POSIX command language interpreter (shell) has many of the functions and features of the Korn Shell. For this version of VMS POSIX, Draft 10 of P1003.2 has been used.

The P1003.2/D10 utilities provide an interface based on the most popular UNIX commands and utilities. The POSIX interactive user can use appropriate P1003.2/D10 commands to manipulate files, while POSIX-compliant applications can interact with the file system using POSIX 1003.1 and P1003.4/D9 functions. The VMS POSIX set of commands and utilities includes the following:

- All utilities presented on P1003.2/D10
- The **make** and **ar** utilities for software development
- The **c89**, **yacc**, and **lex** utilities for developing applications using the C language
- The **asa** utility for developing applications using FORTRAN

- Over 30 new utilities designed to increase application portability

The following file system and shell functionality supports VMS POSIX commands and utilities:

- Container file system
- VMS POSIX shell

10.2.1 VMS POSIX File System

The VMS POSIX file system more closely resembles a UNIX file system than the VMS file system. Like a UNIX file system, VMS POSIX uses pathnames instead of VMS file specifications, but you can use the VMS file system to increase interoperability with other components of the VMS operating system. You can also use the container file system, which provides a means of translating a VMS file name to a file that fully supports the POSIX standards. Files in the container file system have the following features:

- Full conformance to the POSIX standard
- POSIX standard naming conventions
- POSIX file time-stamping
- POSIX user IDs (UID) and group IDs (GID)

Symbolic links, hard links, special files, and first-in first-out (FIFO) files must reside in the container file system.

Within the VMS POSIX system, you can work both in the VMS file system and in the VMS POSIX container file system. Within the VMS POSIX container, P1003.2/D10 utilities can be used to create an additional entry (link) to a file or directory, remove a link, or perform other useful file functions.

10.2.2 VMS POSIX Shell

The VMS POSIX shell is a command language interpreter (CLI) and is equivalent to the VMS Digital Command Language (DCL). You can instruct the shell, either interactively or within a VMS POSIX command file (shell script), to perform a number of tasks. The shell is parsed by the *sh* utility and the functions listed in IEEE P1003.2/D10. The VMS POSIX shell (based on the Korn shell) is an application that runs as a separate process.

Shell scripts are files containing shell commands, much as a VMS command file contains DCL commands. From a shell script, you can perform the following functions:

- Manipulate and run commands in the foreground or background
- Redirect input, output, and error messages

- Use pipelines
- Set and pass variables
- Test conditions, use conditional statements, and loop
- Perform arithmetic operations
- Create shell functions
- Define environment and export variables

You can execute DCL commands from within the VMS POSIX environment by using the VMS POSIX *dcl* command, which creates a VMS POSIX subprocess running DCL. The process can run in the background or foreground, which allows you to create a multitasking environment. During the DCL session, the terminal is changed from VMS POSIX to VMS mode; when the program returns, the mode is reset to VMS POSIX.

10.3 VMS POSIX Realtime Environment

The VMS POSIX product includes support for the POSIX 1003.4 Draft 9 standard (P1003.4/D9), a set of functions and calls that can be used in the design and creation of realtime applications in the VMS POSIX environment in the following areas:

- Process synchronization
- Interprocess communication
- Realtime performance

Other POSIX features support interprocess communication through event notification, message queues, and shared memory. Performance on a realtime system is enhanced through the use of VMS POSIX features such as process memory locking, synchronous and asynchronous I/O, and realtime files.

10.3.1 Process Synchronization Using VMS POSIX

Synchronization techniques and access-control of resources ensure that critical and noncritical activities execute at appropriate times with the necessary resources available. The following table lists the synchronization functions available with VMS POSIX.

Function	Header File	Purpose
Binary semaphores	**<sys/sem.h>**	Restricts access to resources

Function	Header File	Purpose
Clocks and timers	**<sys/timers.h>**	Arms and disables timers
Priority scheduling	**<sys/sched.h>**	Sets process priority and the scheduling policy

- **Binary semaphores.** Semaphores are used by cooperating processes to synchronize access to resources such as shared memory. Semaphores can protect resources such as global variables, hardware resources, and the kernel from uncontrolled access.

  Semaphore protection works only if all of the communicating processes using the shared resource cooperate by waiting for the semaphore when it is unavailable and resetting the semaphore count when relinquishing the resource. For cooperating tasks, semaphores are mutual exclusion flags that lock and unlock a resource.

  With VMS POSIX semaphores, you can create and remove persistent and nonpersistent binary semaphores, post and wait for semaphores, and perform P and V operations. Each action on a semaphore requires an explicit function call.

- **Clocks and timers.** The systemwide clock provides the timing basis for per-process timers and is the primary source for timer synchronization. VMS POSIX clock and timer functions allow you to retrieve and set the systemwide clock, suspend execution for a period of time, provide high-resolution timers, and use asynchronous event notification.

  Realtime timers allow the application to set timers based on either absolute or relative time. Furthermore, VMS POSIX timers can fire as either a one-shot or periodic timer. The application creates timers in advance, but the timers can be manipulated based on the needs of the realtime application. Some applications may require only one or two timers; others may require multiple timers within a single process.

- **Priority scheduling.** The scheduler determines how CPU resources are allocated to executing processes. Each process has a priority that associates the process with a run queue. Although each process starts out with an initial priority, the priority can change as the application executes depending on the algorithm used by the scheduler or application requirements.

  The system maintains a list of runnable processes at each priority level. Each process list has a priority level value ranging from PRIO_MIN to PRIO_MAX. A process in a list with a higher priority value executes before a process in a list with a lower priority value.

VMS POSIX supports three scheduling policies; FIFO (first-in first-out), RR (round-robin), and OTHER (VMS timesharing). A scheduling policy is the algorithm that determines how processes are placed on the process list and when processes execute. Whatever the scheduling policy, the process at the top of the process list with the highest priority level executes first.

Table 10–1 categorizes the VMS POSIX synchronization functions and lists the corresponding functions.

Table 10–1 VMS POSIX Process Synchronization Functions

Operation	Functions
Open, close, and control a special file	**open** **close** **fcntl**
Make a binary semaphore special file	**mksem**
Wait (or conditionally wait) for a binary semaphore	**semwait** **semifwait**
Post (or conditionally post) to a binary semaphore	**sempost** **semifpost**
Get or set the value of the systemwide clock, get the resolution of the clock	**getclock** **setclock** **resclock**
Allocate or free a per-process timer, get the value of the per-process timer	**mktimer** **rmtimer** **gettimer**
Arm a per-process timer absolutely or relatively	**reltimer** **abstimer**
Get the resolution supported by relative or absolute timers and high resolution sleep	**resrel** **resabs** **ressleep**
High resolution sleep	**nanosleep**
Get or set the priority of a process	**getprio** **setprio**
Get or set the scheduling policy	**getscheduler** **setscheduler**
Yield to another process	**yield**

10.3.2 Interprocess Communication Using VMS POSIX

Using interprocess communication, you can synchronize independently executing processes by passing data within an application. Processes can pursue their own tasks until they must synchronize with other processes at some predetermined point. When they reach that point, they wait for some form of communication to occur. The following table lists the types of communication functions available in VMS POSIX.

Function	Header File	Purpose
Event notification	**<sys/events.h>**	Notifies the calling process of an instance of an event
Message queue	**<sys/mqueue.h>**	Passes messages between processes
Shared memory	**<sys/shmmap.h>**	Allows two or more processes to share the same address space

- **Asynchronous event notification.** Event notification is a way of passing data within an application when the application must respond to an application-defined occurrence. Events are associated with event classes to define groups of events so that you have more control over operations. In addition, event masks are used to block or unblock event notification.

  When an event is raised, the event is queued and the calling process is notified if the calling process is currently executing or the event becomes unblocked. A user-defined event handler is activated to handle the event. An event can be the arrival of a message, timer expiration, arrival of data, or asynchronous I/O request completion.

- **IPC message passing.** Interprocess communication through message queues allows cooperating processes within an application to send data to each other. A message queue is a systemwide special file accessible by a pathname. Message queues remain after a child process terminates and can be reused by other processes.

  VMS POSIX message functions allow you to define the size of the message, receive messages selectively, send or receive messages asynchronously, and perform other operations.

- **Shared memory.** Shared memory allows multiple processes to share data by mapping a region of memory into each process's address space. Use of shared memory among processes provides faster data access.

You can allocate as little memory as possible for each process or allocate one large region and manage it within the application. Shared memory can be created as persistent or nonpersistent. Once created, shared memory can be mapped and unmapped, linked and unlinked, or closed.

Table 10–2 lists the VMS POSIX interprocess communication operations and the corresponding functions.

Table 10–2 VMS POSIX Interprocess Communication Functions

Operation	Functions
Get or set the mask of blocked event classes, suspend processes, block specified event classes	**evtprocmask** **evtsuspend**
Poll for event notification, generate an application-defined event	**evtpoll** **evtraise**
Associate signals with an event class	**evtsigclass**
Create a message queue special file, send or receive a message from a queue	**mkmq** **mqsend** **mqreceive**
Get and set message queue attributes, purge messages	**mqsetattr** **mqgetattr** **mqpurge**
Get process identifier, allocate and free a message data buffer	**mqgetpid** **msgalloc** **msgfree**
Get and put event data from or on a message queue	**mqgetevt** **mqputevt**
Make a shared memory special file, map the shared file into a process's address space, unmap previously mapped shared memory	**mkshm** **shmmap** **shmunmap**

10.3.3 Realtime Performance Using VMS POSIX

Access to files and memory represent major factors affecting the speed of program execution. A realtime application designer should consider the importance of execution speed versus file and data integrity. The following table lists the types of performance functions available in VMS POSIX.

Function	Header File	Purpose
Process memory locking	**<sys/memlk.h>**	Locks process address space in memory

Function	Header File	Purpose
Synchronized I/O	**<sys/fcntl.h>**	Performs synchronized I/O operations
Asynchronous I/O	**<sys/aio.h>**	Performs asynchronous I/O operations
Real-time Files	**<sys/files.h>**	Performs file management

- **Process memory locking.** A realtime application cannot afford long latencies in the execution of critical code. In a virtual memory system, a process may have part of its address space paged in and out of memory in response to system demands for critical space. In some cases the entire process may be swapped out to disk in order to free up resources. Memory locking is one of the primary tools for guaranteeing that time-critical processes are locked into memory and subject to memory management appropriate only for timesharing applications.

  However, realtime processes require an upper limit on the amount of time required to fetch data. VMS POSIX memory-locking functions allow an application to lock the entire address space, and the stack segment, and to perform region and lock stacking.

- **Synchronized input and output.** Synchronized input and output ensures data and file integrity during read and write operations. A synchronized input operation ensures that a read operation from a device transfers the current image of the data from the device. A synchronized output operation blocks the initiating process from executing and does not return until the write operation is complete.

  Synchronized I/O file integrity is a superset of data integrity that also ensures that all other file system information relevant to the data is successfully transferred.

- **Asynchronous input and output.** Asynchronous I/O allows the calling process to regain control of execution immediately once an I/O operation is queued. Without asynchronous I/O, the process waits while I/O completes before continuing execution. Realtime applications require input and output operations that do not block the initiating process. With VMS POSIX, asynchronous I/O read and write operations are queued and control is immediately returned to the calling process.

- **Realtime files.** A realtime application relies on the file management system to provide bounded response times to whatever external demands are placed on the application. The file system must ensure a high level of performance, access to resources, and delivery of the data to the media.

VMS POSIX realtime files allow for this capability. Note, however, that VMS POSIX does not support direct I/O.

Table 10–3 categorizes the VMS POSIX performance functions and lists the corresponding functions.

Table 10–3 VMS POSIX Realtime Performance Functions

Operation	Functions
Lock and unlock a memory region	**memlk** **memunlk**
Asynchronous read and write, list-directed I/O	**aread** **awrite** **listio**
Cancel or wait for asynchronous I/O request	**acancel** **iosuspend**
Asynchronous and synchronous file synchronization	**afsync** **rtsync**
Create a realtime file, get and set attributes of a realtime file	**rtcreate** **getattr** **fgetattr** **setattr** **fsetattr**
Get the capabilities of a realtime file, get the increment list	**getcap** **fgetcap** **getincr** **fgetincr** **fsgetincr**
Read and write from a file	**read** **write**

Part IV

DEC OSF/1 Systems in Realtime Applications

Part IV surveys DEC OSF/1 as a development and runtime environment for realtime applications and introduces the POSIX functionality for realtime application development on DEC OSF/1 systems. This part contains the following chapters:

- Chapter 11, Survey of Programming on DEC OSF/1 Systems, outlines the DEC OSF/1 operating systems and some realtime kernel application programming techniques.
- Chapter 12, DEC OSF/1 Realtime Programming Environment, outlines the functionality provided with DEC OSF/1 POSIX realtime (P1003.4/D11) kernel.

11 Survey of Programming on DEC OSF/1 Systems

Digital offers two UNIX operating systems, ULTRIX and DEC OSF/1. Both operating systems are full-service systems that are highly regarded within the industry. Both ULTRIX and DEC OSF/1 offer an environment for developing realtime applications. As a layered product, DECelx Realtime Tools for ULTRIX (DECelx) includes a runtime executive, powerful testing and debugging facilties, and an unparalleled ULTRIX cross-development package. As a component of the DEC OSF/1 operating system, the realtime kernel provides suppport for many POSIX 1003.4 Draft 11 realtime functions.

The DEC OSF/1 operating system is Digital's implementation of the Open Software Foundation (OSF) operating system component V1.01 including OSF's Motif V1.1.3.

This chapter discusses the following DEC OSF/1 topics:

- DEC OSF/1 Operating System Overview, Section 11.1
- DEC OSF/1 Programming Support Tools, Section 11.2
- DEC OSF/1 Realtime Environment, Section 11.3

For additional information on DECelx Realtime Tools, refer to Part I. For additional sources of information on DEC OSF/1 and DECelx, refer to Appendix A.

11.1 DEC OSF/1 Operating System Overview

The DEC OSF/1 operating system is an advanced kernel design. It is an advanced kernel architecture based on the Mach V2.5 kernel design from Carnegie Mellon University. The operating system complies with standards and industry specifications, including FIPS 151-1, POSIX 1003.1–1990, XPG3 base branding, XTI, X11R5 support, and AT&T System V Interface Definition (SVID) Issue 3 (base plus kernel extensions). The DEC OSF/1 operating system is compatible with the Berkeley 4.3 BSD programming interfaces, which ensures a level of compatibility with Digital's ULTRIX operating system.

The base operating system kit contains media for installing the Realtime Options. When installed, the realtime kernel and P1003.4/D11 functions are available to users on that system.

The following DEC OSF/1 features assist programmers in developing realtime applications:

- Preemptive kernel, Section 11.1.1
- Realtime priorities, Section 11.1.2
- DECthreads, Section 11.1.3
- Memory mapped files, Section 11.1.4
- Interprocess Communication, Section 11.1.5
- Networking, Section 11.1.6

One of the goals of OSF is to provide an interface for developing portable applications that run on a variety of hardware platforms. DEC OSF/1 is compliant with the OSF Application Environment Specification (AES), which specifies the interface to support portable applications.

11.1.1 Preemptive Kernel

When you install the realtime options kit, the kernel is replaced with a realtime (preemptive) kernel. The preemptive kernel allows a higher-priority process to preempt a lower-priority process regardless of whether it is running in kernel mode or user mode. With a preemptive kernel, the Process Preemption Latency (the amount of time it takes to preempt a lower-priority process) is small and bounded.

A realtime environment must be able to respond to an event within a bounded (generally quite short) period of time. A preemptive kernel guarantees that a higher-priority process can quickly interrupt a lower-priority process, regardless of whether the low-priority process is in user or kernel mode. Whenever a higher-priority process becomes runnable, a preemption is requested, which causes the higher-priority process to displace the running, lower-priority process. A preemptive kernel guarantees a deterministic response to realtime events by providing the ability to respond to realtime requests.

Every realtime application interacts with the operating system in two modes: user mode and kernel mode. User mode processes allow the application user to interact with the application. User mode processes call utilities, library functions, and other user applications.

In kernel mode, the application accesses and interacts with the operating system. During execution, a user process often calls system functions, switching the context from user to kernel mode.

The amount of time it takes for a higher-priority process to displace a lower-priority process is referred to as Process Preemption Latency. In a realtime environment, the primary concern of application designers is the Maximum Process Preemption Latency that can occur at runtime. Designers must understand system timing contraints before designing time-critical applications.

A preemptive kernel, such as the DEC OSF/1 realtime kernel, allows the operating system to respond as quickly as possible to a process preemption request. The DEC OSF/1 realtime kernel can break out of kernel mode to honor the preemption request.

A preemptive kernel supports the concept of process synchronization, while maintaining data integrity, with the ability to respond quickly to interrupts. The kernel employs mechanisms to protect the integrity of kernel data structures and defines the restrictions on where the kernel cannot preempt execution.

The Maximum Process Preemption Latency for a preemptive kernel is exactly the amount of time required to preserve system and data integrity and preempt the running process. Under these conditions it is not unusual for worst-case preemption to take milliseconds.

11.1.2 Realtime Priorities

DEC OSF/1 user and system priorities are designed to handle the needs of timesharing users and the operating system. Priority ranges are divided equally between the nonprivileged user and the system, with the total number of priorities equal to 40.

The DEC OSF/1 realtime kernel supports three priority ranges and is designed to handle the needs of nonprivileged, timesharing users, the operating system, and realtime users. The total number of priorities available is 64, with 32 reserved for realtime processes.

Every process begins execution with a default initial priority inside the range for timesharing processes. During execution, realtime processes dynamically adjust their priorities to deal with realtime tasks. Process priority is closely tied to the P1003.4/D11 scheduling policies and must be used in conjunction with those functions.

11.1.3 DECthreads

DEC OSF/1 provides software developers with the ability to write multithreaded programs using DECthreads. DECthreads provides two interfaces to its threading engine, one which is defined by the former Concert Multithreaded Architecture (CMA) and a second draft which is defined by the P1003.4a/D4 draft specification.

A thread is a single sequential flow of control within a process. It is the active execution of a designated routine, including any nested routine invocations. Within a single thread, there is a single point of execution. Applications can be designed to be multithreaded so that each process can have multiple flows of execution.

Because threads execute independently and simultaneously, they provide a useful model for structuring applications to exploit parallelism. An application can be designed to have multiple single-threaded processes carrying out processing of different parts of an application. For example, different processes can do read and write operations from different files, resulting in non-blocking I/O.

Threads can be scheduled based on the priority of the thread and are subject to the same priority scheduling policies used by realtime functions. Each thread has an associated priority and scheduling policy: FIFO, round-robin, or timesharing.

Binary semaphores are used to synchronize thread access to shared resources. Special thread functions create these semaphores and track locking activities.

11.1.4 Memory-Mapped Files

DEC OSF/1 supports the Berkeley `mmap` function which allows an application to access files with memory operations rather than file I/O operations.

Memory-mapped files allow a process to access files by directly incorporating file data into the process's address space. Once a file is mapped into process's address space, the data can be easily manipulated and I/O data movement is reduced. Data no longer has to be copied into process data buffers as it is with a `read` or `write` function call. If multiple processes concurrently map a file, its contents may be shared among them, thus creating a low-overhead means of communication and synchronization.

11.1.5 Interprocess Communication

On DEC OSF/1 Version 1.2, interprocess communication is done using both P1003.4/D11 and System V IPC constructs as well as pipes, named pipes, and POSIX 1003.1 signals. DEC OSF/1 interprocess communication includes:

- **System V messages.** A message queue is a systemwide special file accessible by a pathname. Message queues remain after a child process terminates and can be reused by other processes. System V message functions allow you to define the size of the message, receive messages selectively, send or receive messages asynchronously, and other operations that make message queues desireable for realtime applications.
- **Pipes**. Pipes are used to transfer small amounts of data among related processes.
- **Named pipes**. Named pipes are like pipes, except that named pipes use file descriptors which provide communication for unrelated processes.
- **POSIX 1003.1 signals.** POSIX 1003.1 signals provide a means to communicate to a large number of processes, but communication is limited to a signal number. Some signals used in DEC OSF/1 realtime timer and asynchronous I/O functions use a data structure, making signal delivery asynchronous, fast, and reliable.

11.1.6 Networking

On DEC OSF/1 Version 1.2, networking is facilitated using traditional networking methods as well as industry-standard networking facilities. DEC OSF/1 networking facilities include the following functionality:

- **INTERNET.** The DEC OSF/1 operating system supports a number of INTERNET RFC (Request for Comment) and non-RFC standards.
- **TCP/IP.** TCP/IP supports network communications over supported network devices. The TCP/IP protocol suite is implemented in the socket framework.
- **Sockets.** Sockets are similar to pipes. Each process has a single, two-way channel. A socket is created with a domain and a type, which defines communication semantics.
- **Streams.** Streams provide dynamic loading and linking of kernel level protocols. Like sockets, streams provide a framework for character I/O to and from user space to kernel networking protocols.
- **XTI.** X/Open Transport Interface (XTI) is an extension to the AT&T System V Streams user space interface called Transport Level Interface (TLI).

- **FDDI support.** DEC OSF/1 provides FDDI fiber optic support for the DECStation and DECsystem 5000 Series.
- **Name Services.** DEC OSF/1 supports the BIND V4.8 name service, which provides a host name and address lookup service for the INTERNET network. You can use BIND to replace or supplement the hosts, aliases, auth, group, networks, passwd, protocols, rpc, and services databases.
- **Network Time Protocol.** DEC OSF/1 provides the Network Time Protocol (NTP) to synchronize and distribute time for all machines in a network environment. The time synchronization daemon, `timed`, is used to distribute time to all networked machines.

11.2 DEC OSF/1 Programming Support Tools

The DEC OSF/1 base system software kit provides a number of programming support tools, including facilities for text manipulation, macro and application generation, source file management, and software kit installation and creation. These programming support tools are independent of COHESION environment products, which combine much of the functionality of these tools into an integrated development environment.

The following utilities and commands are of specific interest to realtime application developers:

- **The compiler system.** The C language is the primary language supported by DEC OSF/1. Tools that make up the DEC OSF/1 compiler system include compiler commands, preprocessors, compilation options, the link editor, and the archiver.
- **Optimizer.** The global optimizer improves the performance of DEC OSF/1 compiler object programs by transforming existing code into more efficient coding sequences. The compiler system performs both machine-independent and machine-dependent optimizations.
- **dbx.** The primary debugging tool on DEC OSF/1 is the `dbx` debugger. This debugger allows you to trace problems at the source code level, control a program's execution, and monitor program control flow, variables, and memory locations.
- **make.** The `make` utility builds up-to-date versions of the application. It is most useful in large programming projects in which multiple source files are combined to form a single application.

- **prof**. The `prof` program helps find areas of code where most of the execution time is spent. In the typical application, most of the execution time is consumed in a few sections of code; it is profitable to concentrate on improving code efficiency in those sections.
- **SCCS**. The Source Code Control System, which is also in integral part of the DECset and FUSE products, is a code management system that allows you to keep source files in a common library and maintain control over them.
- **setld**. The `setld` software installation and management facility allows you to create, install, manage, and remove software kits. All DEC OSF/1 software kits supplied by Digital are compatible with `setld` and many other software vendors also supply their kits in this form.

These and other programming support tools are included with the base system software kit. You can also separately purchase CASE tools, which provide a common interface and database for managing software development projects. For additional information on UNIX CASE tools, refer to Chapter 18.

11.3 DEC OSF/1 Realtime Environment

The DEC OSF/1 realtime kernel and environment provides you with the capability of developing and running portable applications in a POSIX environment. DEC OSF/1 includes support for the following POSIX standards and draft standards:

- POSIX 1003.1–1990
- P1003.4/D11

11.3.1 POSIX 1003.1 on DEC OSF/1

The POSIX 1003.1 standard is a set of functions and calls for use in application programs using the C programming language. The purpose of using the POSIX 1003.1 standard in an application is to promote and allow the portability of such an application between systems that support the POSIX 1003.1 standard. The functions and calls in POSIX 1003.1 include some functions that are identical to ANSI C functions, some functions that use the same syntax as ANSI C functions, but operate slightly differently in the POSIX environment, and some functions that are unique to POSIX.

11.3.2 P1003.4/D11 on DEC OSF/1

The POSIX 1003.4/D11 draft standard is a set of functions and calls that can be used in the design and creation of applications in the DEC OSF/1 realtime environment.

The DEC OSF/1 realtime environment offers these POSIX features:

- **Process synchronization.** DEC OSF/1 functions offer several methods for synchronizing processes, including:
  - **Realtime clocks and timers.** Realtime clocks and timers allow an application developer to synchronize and coordinate activities according to a predefined schedule. The systemwide clock (CLOCK_REALTIME) provides the timing base for per-process timers. Realtime timers are created, armed, and removed by the application programmer. You can use an absolute or relative timer on a one-shot or periodic basis.

    Realtime timers use the nanosecond as the smallest unit of time, which makes them better suited to realtime applications.
  - **Priority scheduling.** Priority scheduling gives an application programmer control over the execution sequence of processes that comprise an application. Priority scheduling also addresses the need to ensure that a realtime process to can execute when it needs to and for as long as necessary to satisfy the realtime requirement.

    The system maintains a list of runnable processes at each priority level. Each process list has a priority level value in the range from PRIO_MIN to PRIO_MAX. A process in a list with a higher priority value executes before a process in a list with a lower priority value.

    The DEC OSF/1 realtime kernel supports three scheduling policies; first-in first-out (FIFO), round-robin (RR), and timesharing (OTHER). These three scheduling policies determine how the process lists are managed, the order of the process lists, and when processes of different priorities can execute.
  - **Semaphores.** As a synchronization mechanism, a semaphore is used to control access to systemwide resources. With P1003.4/D11 semaphores, you can create, reserve, release, and remove binary semaphores. Each action on a semaphore requires an explicit function call and is under the control of the application programmer.
- **Shared memory.** Shared memory allows multiple processes to share data by mapping a region of memory into each process's address space. Use of shared memory among processes provides faster data access.

You can allocate a little memory as possible for each process or allocate one large region and manage it within the application. Shared memory can be opened and unlinked.

- **Process memory locking.** Access to files and memory represent major factors affecting the speed of program execution. The DEC OSF/1 realtime kernel provides process memory locking as a means to control application performance.

  Locking a process's address space in memory eliminates paging by the target process. DEC OSF/1 memory-locking functions allow an application to lock and unlock both the current and future address space.

- **Asynchronous I/O.** Realtime applications require input and output operations that do not block the initiating process. With the DEC OSF/1 realtime kernel asynchronous I/O read and write operations are queued and the initiating process continues execution.

12 DEC OSF/1 Realtime Programming Environment

The DEC OSF/1 operating system provides you with the capability of developing and running applications in a POSIX environment. DEC OSF/1 includes a regular programming interface and a realtime programming interface, based on the POSIX standards and draft standards. The POSIX standards and drafts support the concept of open systems.

This chapter describes the following aspects of the DEC OSF/1 POSIX environment:

- DEC OSF/1 POSIX Environment, Section 12.1
- DEC OSF/1 Realtime Functions, Section 12.2

For additional information on the DEC OSF/1 realtime environment, refer to the *DEC OSF/1 Guide to Realtime Programming*. Additional sources of information are listed in Appendix A.

12.1 DEC OSF/1 POSIX Environment

DEC OSF/1 supports the POSIX 1003.1–1990 standard to promote and allow for portability between systems that support the POSIX 1003.1 standard. The POSIX 1003.1 standard defines a standard operating system interface and provides standardized services through the C programming language.

The following major categories are covered by the POSIX 1003.1 standard.

— **Process primitives.** Process primitives define process creation, execution, and termination. Signals and timers are defined as a means of synchronizing process execution and interprocess communication.

— **Process environment.** Process environment commands allow you to set and retrieve user and group IDs. In addition, commands are used for system identification, setting time, accessing environment variables, identifying terminals, and configuring system variables.

— **Files and directories.** File commands allow you to create, open, link, set a mask, rename, or remove files and special files. You can set file characteristics, such as file accessibility and status. Directory commands allow you to perform directory operations, such as creating or removing a directory.

— **Input and output primitives.** Input and output primitives include standard read and write operations as well as pipes and control operations on files.

— **Device-specific and class-specific functions.** Device-specific and class-specific functions include functions that allow you to communicate with devices and device files.

— **Language-specific services for the C programming language.** Language-specific services define extensions to C language functions.

— **System databases.** System database functions create and establish access to system databases.

— **Data interchange format.** Data interchange format commands specify archive and interchange file formats.

DEC OSF/1 provides a POSIX runtime library and header files so that you can compile and link your application in the POSIX environment.

The `man` utility is available for obtaining a full range of online help for commands and functions. If your system provides you with access to the Bookreader, then you can also obtain information through that means.

12.2 DEC OSF/1 Realtime Functions

The DEC OSF/1 realtime kernel includes support for portions of the P1003.4/D11 draft standard, a set of functions and calls that can be used in the design and creation of realtime applications in the DEC OSF/1 realtime environment in the following areas:

- Clocks and Timers
- Process Priority Scheduling
- Memory Locking
- Asynchronous I/O
- Semaphores
- Shared Memory and Memory Mapped Files

12.2.1 Clocks and Timers

The DEC OSF/1 systemwide clock, CLOCK_REALTIME[1], provides the timing base for per-process timers and is the primary source for timer synchronization. This clock maintains user and system time as well as the current time and date. The resolution of the CLOCK_REALTIME clock is such that it provides the basic mechanism to support realtime per-process timers and high resolution sleep.

Clock and timer functions allow you to retrieve and set the systemwide clock, retrieve and correct for clock drift rate, suspend execution for a period of time, provide high resolution timers, and use asynchronous signal notification.

Timers in realtime applications must be able to respond quickly to asynchronous external events. Timers often schedule tasks and events in time increments considerably smaller than the traditional one-second timeframe. Because the CLOCK_REALTIME clock and realtime timers use seconds and nanoseconds as the basis for time intervals, the resolution for the system clock, realtime timers, and the `nanosleep` function has a fine granularity. For example, in a robotic data acquisition application, information retrieval and recalculation operations may need to be completed within a 4-milliseconds timeframe. Timers are created to fire every 4 milliseconds to trigger the collection of another round of data. On expiration, a timer sends a signal to the calling process.

Realtime timers must be flexible enough to allow the application to set timers based on either absolute or relative time. Furthermore, timers must be able to fire as a one-shot or periodic timer. The application creates timers in advance, but specifies timer characteristics when the timer is set.

Realtime applications must be able to establish and manipulate timers based on the needs of the application. Some applications may require only one or two timers; others may require multiple timers within a single process. The P1003.4/D11 timing facilities support multiple per-process timers up to a system-defined limit. Each timer is created and armed independently, which means that the application designer controls the action of each and every timer.

Table 12–1 categorizes the DEC OSF/1 clock and timer functions.

[1] CLOCK_REALTIME is the TIME-OF-DAY clock for DEC OSF/1.

Table 12–1 DEC OSF/1 Clock and Timer Functions

Operation	Functions

Get or set the value of the systemwide clock, get the resolution of the clock	`clock_gettime`
	`clock_settime`
	`clock_getres`

Get or set the clock drift rate	`clock_gettimedrift` `clock_settimedrift`

Allocate or free a per-process timer, get the value of the per-process timer	`timer_create`
	`timer_delete`
	`timer_gettime`

Arm a per-process timer absolutely or relatively	`timer_settime`
High resolution sleep	`nanosleep`

12.2.2 Process Priority Scheduling

The scheduler determines how CPU resources are allocated to executing processes. Each process has a priority that associates the process with a run queue. Although each process starts out with an initial priority, that priority can change as the application executes, depending on the algorithm used by the scheduler or application requirements.

A scheduling policy is the algorithm that determines how processes are placed on the process list and when processes execute. Whatever the scheduling policy, the process at the top of the process list with the highest priority level executes first. DEC OSF/1 realtime supports three scheduling policies; first-in first-out (FIFO), round-robin (RR), and timesharing (OTHER). FIFO and RR are fixed-priority scheduling policies. That is, a process's priority and scheduling policy are controlled by the user, rather than the scheduler.

Under the FIFO scheduling policy, a running process continues to execute if there are no other higher-priority processes. Under the control of the user, a running process can raise its priority to avoid being preempted by another process. Therefore, a high-priority, realtime process running under the FIFO scheduling policy can use system resources as long as necessary to finish realtime tasks.

Under the RR scheduling policy, the highest-priority process runs until either its allotted time (quantum) is complete or the process is preempted by another, higher-priority process. A process continues to execute as long as the waiting processes are at a lower priority. Therefore, high priority processes running under the RR scheduling policy can share the processor with other time-critical processes.

Under the OTHER scheduling policy, a running process runs in the same manner as any other timesharing process on the system. Priorities can be adjusted by the system. The OTHER scheduling policy is designed for non-realtime tasks.

Table 12–2 categorizes the DEC OSF/1 process priority scheduling functions.

Table 12–2 DEC OSF/1 Process Priority Scheduling Functions

Operation	Functions
Get or set the priority of a process	`sched_getparam` `sched_setparam`
Get the maximum or minimum priority allowed	`sched_get_priority_max` `sched_get_priority_min`
Get or set the scheduling policy	`sched_getscheduler` `sched_setscheduler`
Get the time limit allowed for round robin scheduling	`sched_get_rr_interval`
Yield to another process	`sched_yield`

12.2.3 Process Memory Locking

A realtime application cannot afford long latencies in the execution of critical code. In a virtual memory system, a process may have part of its address space paged in and out of memory in response to system demands for critical space. Memory locking is one of the primary tools for guaranteeing that time-critical processes are locked into memory and not subject to memory management for timesharing applications. Unless time-critical processes are locked into memory, the latency introduced by paging may cause involuntary and unpredictable time delays at runtime.

Realtime application developers should consider memory locking as a required part of program initialization. Many realtime applications remain locked for the duration of execution, but some may want to lock and unlock memory as the application runs. You can use the `plock` function or you can use DEC OSF/1 P1003.4/D11 memory-locking functions. The P1003.4/D11 memory-locking functions let you lock the entire process at the time of the function call and to automatically lock all additional memory throughout the life of the application. Or, you can selectively lock and unlock memory as needed.

Two P1003.4/D11 functions allow you to lock memory, which makes the process memory or segments of the process immune to paging. The `mlock` function lets you lock an address range into memory, and the `mlockall` function lets you lock all of a process's memory (both current and future). You can remove memory locks with corresponding calls to the `munlock` and `munlockall` functions.

DEC OSF/1 P1003.4/D11 memory-locking functions are easier to use than many other memory-locking functions and offer the following advantages:

- Portability
- Ability to preallocate and lock memory space
- Ability to lock memory with fewer function calls

Realtime applications often need to lock the entire process for the life of the application. Memory-locking functions globally track which regions are locked and which are not. If the data or text segment of a process is shared, then locked data or text is locked for all sharing processes.

Table 12–3 categorizes the DEC OSF/1 realtime memory-locking functions.

Table 12–3 DEC OSF/1 Memory-Locking Functions

Operation	Functions
Lock and unlock a specified memory region	`mlock` `munlock`
Lock and unlock all memory mapped by the address space	`mlockall` `munlockall`

12.2.4 Asynchronous Input and Output

DEC OSF/1 asynchronous I/O allows the calling process to regain control of execution immediately after an I/O operation is queued. This capability is desirable in many different applications ranging from graphics and file servers to dedicated realtime data acquisition and control systems. Without asynchronous I/O, the process waits while I/O completes before continuing execution. With asynchronous I/O, once an I/O request is queued, control is immediately returned to the calling process. The process immediately continues execution, thus overlapping tasks.

Many realtime applications need this ability to overlap application processing and I/O operations. Often, one process simultaneously performs multiple I/O functions while other processes continue execution. For example, a process can queue data for output without blocking (waiting for I/O completion). Applications need to gather large quantities of data from multiple channels within a short, bounded period of time. In such a situation, blocking I/O may work at cross purposes with application timing constraints. Asynchronous I/O performs nonblocking I/O, allowing simultaneous reads and writes, which frees processes for additional processing.

Notification of asynchronous I/O completion is optional. If you choose to use signal notification, the signal is specified in the `aiocb` structure, thereby eliminating the need to call the `signal` function and providing faster interprocess communication.

Table 12–4 categorizes the DEC OSF/1 realtime asynchronous functions.

Table 12–4 DEC OSF/1 Asynchronous I/O Functions

Operation	Functions

Asynchronous read and write, list-directed I/O	`aio_read`
	`aio_write`
	`lio_listio`

Retrieve error or return status of list-directed I/O	`aio_error` `aio_return`
Cancel or wait for asynchronous I/O request	`aio_cancel` `aio_suspend`

12.2.5 Shared Memory and Memory Mapped Files

Shared memory and memory-mapped objects allow processes to communicate by incorporating data directly into process address space. Processes communicate by sharing portions of their address space. When one process writes to a location in the shared area, the data is immediately available to other processes sharing the area. Communication is fast because there is none of the overhead associated with system calls. Data movement is reduced because it is not copied into buffers.

A memory object is either a memory-mapped file or a shared-memory object that can be mapped concurrently into the address space of one or more processes. Memory-mapped files are files, while shared-memory objects may be either files or namespaces. In either case, shared memory can be accessed by all processes through the use of global names. Processes using shared memory map specific sections of the physical address space by using the name of the shared memory.

A process manipulates its address space by mapping or removing portions of shared memory into the process address space. When multiple processes map the same shared memory, they share access to the underlying data. Shared-memory functions allow you to open and unlink the shared memory. The functions discussed in this chapter allow you to map and unmap, protect, and synchronize shared memory.

Using a shared, mapped object means that the changes to the mapped memory are reflected back to the memory. Other processes using the same file descriptor and opening the connection to the shared-memory region have a shared mapping of the file. Thus, POSIX memory-mapped objects permit a single process or multiple processes to incorporate data directly into process address space. Once an object is mapped into a process address space, its data can be manipulated easily. If the mappings allow it, data written into memory through the address space of one process appears in the address space of all processes mapping the same portion of the memory.

Memory mapped objects are persistent; their names and contents remain until the application calls the `unlink` function.

Using semaphores in conjunction with shared memory, you can further synchronize access to shared memory. When using shared memory, processes map the same area of memory into their address space. This allows for fast interprocess communication because the data is immediately available to any other process using the same shared memory. If your application has multiple processes contending for the same shared-memory resource, you must coordinate access.

Binary semaphores provide an easy means of regulating access to shared memory and to determine if the memory resource is available. Typically, an application will begin execution at a nonrealtime priority level, then perform the following tasks when an application uses mapped or shared memory and semaphores:

- Create the shared memory
- Determine the address and map the region into memory
- Create a binary semaphore set
- Adjust the process priority and scheduling policy as needed
- Before a read or write operation, lock (reserve) the semaphore
- After a read or write operation, unlock (release) the semaphore

Table 12–5 categorizes the DEC OSF/1 realtime shared memory and memory mapped file functions.

Table 12–5 DEC OSF/1 Shared Memory and Memory Mapped File Functions

Operation	Functions
Maps and unmaps the memory object into memory	`mmap` `munmap`
Modifies protections of memory objects	`mprotect`
Synchronizes a memory-mapped object	`msync`
Returns the shared-memory file descriptor	`shm_open`
Removes the name of the shared memory	`shm_unlink`

12.2.6 Semaphores

Binary semaphores, as specified in P1003.4/D11, provide an efficient form of interprocess communication. Cooperating processes can use binary semaphores to synchronize access to resources, most commonly, shared memory. Semaphores can also protect the following resources available to multiple processes from uncontrolled access:

- Global variables, such as file variables, pointers, counters, and data structures. Protecting these variables means preventing simultaneous access by more than one process, such as reading information as it is being written by another process.
- Hardware resources, such as disk and tape drives. Hardware resources require controlled access because simultaneous access can result in corrupted data.

Semaphore protection works only if all the processes using the shared resource cooperate by waiting for the semaphore when it is unavailable and resetting the semaphore to an unlocked state when relinquishing the resource. Applications using binary semaphores must carefully detail cooperative tasks. All of the processes that share a resource must agree on which semaphore controls the resource.

POSIX P1003.4/D11 binary semaphores are persistent. The state of the individual binary semaphore is preserved after the semaphore set is no longer open. For example, a binary semaphore set containing 10 semaphores may contain 7 semaphores in the locked state and 3 in the unlocked state when the last process using the semaphore set closes it. The next time a process opens that semaphore set, it will find 7 locked semaphores and 3 unlocked ones. For this reason, cleanup operations are advised when using binary semaphores.

The functions relating to semaphores follow the same general logic as for P1003.4/D11 shared memory and memory mapped files. Table 12–6 categorizes the DEC OSF/1 realtime semaphore functions.

Table 12–6 DEC OSF/1 Semaphore Functions

Operation	Functions
Create and open a new binary semaphore set	`sem_mksem` `sem_open`
Unlocks a binary semaphore	`sem_post`
Deallocates or destroys the binary semaphore set	`sem_close` `sem_destroy`
Returns the number of semaphores in the set	`sem_getnsems`

As with other interprocess communication methods, you can set up a signal handler to remove the semaphore set as one of the tasks performed by the last process in your application.

Part V

High-Level Language Support for Realtime Applications

Part V surveys high-level programming language tools that you can use for realtime and scientific application development and execution. This part contains the following chapters:

- Chapter 13, High-Level Language Overview, identifies high-level programming language tools used to develop realtime and scientific applications.
- Chapter 14, VAXELN Ada, describes the programming environment and software tools used to develop and run VAXELN Ada applications.
- Chapter 15, XD Ada Cross-Development System, discusses a family of production-quality Ada cross-development tools for embedded realtime systems projects.

13 High-Level Language Overview

Digital furnishes high-level programming language products for realtime and scientific application development. The products include compilers and associated tools and options. You can use particular combinations of these products to code VAXELN, VAX Realtime Accelerator, VAXELN Ada, and XD Ada systems, as well as DEC Realtime Integrator functions.

The high-level languages and tools Digital supplies for realtime and scientific application development include:

- DEC Ada, Section 13.1
- VAX BASIC, Section 13.2
- DEC C, Section 13.3
- DEC C++, Section 13.4
- DEC Fortran, Section 13.5
- VAX FORTRAN High-Performance Option, Section 13.6
- DEC Pascal, Section 13.7
- VAX Pascal, Section 13.8

These language products have been tailored to work efficiently with the DECset CASE tools presented in Chapter 17. In addition, as native-mode VAX languages, they are integrated into the VAX common language environment. This integration provides support for the VAX interlanguage calling standard and VMS system services and utilities.

Other compilers and tools you can use for realtime application development include VAXELN Pascal, described in Chapter 6; VAXELN Ada, Chapter 14; and XD Ada, Chapter 15.

13.1 DEC Ada

DEC Ada for OpenVMS on VAX and Alpha systems are validated implementations of the full ANSI/MIL–STD–1815A–1983 Ada language, specified in the *Reference Manual for the Ada Programming Language*. The DEC Ada compiler runs on the OpenVMS operating system and generates optimized, shareable, position-independent code.

DEC Ada for ULTRIX on RISC systems and DEC Ada for DEC OSF/1 on MIPS systems implement the American National Standards Institute (ANSI) and International Standards Organization (ISO) standard Ada programming language on UNIX systems. Where allowed by the standard, DEC Ada for UNIX systems is implemented to make programming in a UNIX environment convenient and efficient.

Ada is a powerful, general-purpose programming language that emerged as the result of a competition sponsored by the U.S. Department of Defense. The competition was intended to define a language suitable for programming embedded computer systems. Among the language requirements were features to reduce software costs by increasing maintainability, evolvability, reliability, and portability. The Ada language meets these requirements through the following features:

- Modular structure for programs
- Data abstraction
- Separate compilation of program units
- Strong typing
- Generic definitions
- Exception handling

In addition, Ada's multitasking language features make it particularly suitable for realtime applications.

Ada Programming Support Environment

The environment for developing Ada programs consists of tools and utilities provided by DEC Ada and the operating system; the Ada compiler, Ada library of predefined units, Ada program library manager, Ada runtime library, a debugger, and development tools and utilities. You may also want to take advantage of the wide assortment of CASE tools, many of which are surveyed in Chapter 17.

DEC Ada Program Library Manager The DEC Ada program library manager furnishes management and utility functions for Ada program libraries. The program manager supports large-scale programming efforts through separate compilation. Separate compilation allows Ada programs to be composed of individual compilation units, all of which depend on each other in specific ways. These dependencies determine the order of compilation for the units. The DEC Ada program library manager keeps track of the individual compilation unit dependencies and can compile units in their proper order to keep the program library current.

Because the DEC Ada program library manager was designed for the development of both DEC Ada and VAXELN Ada systems, you can defer your choice of a target system until you link your Ada application. This makes it easier to build an Ada application that can run on either an OpenVMS or VAXELN target.

By deferring your choice of a target system until link time, you can create a DEC Ada program library that can then be modified for a VAXELN target. You can do this in one of two ways: either you can change the DEC Ada program library characteristics and then relink, or you can relink by invoking the program library manager with a special link qualifier. In the latter case, the DEC Ada program library is not modified. Likewise, a VAXELN Ada program library can be modified for a VAX target.

On a UNIX system, the program library manager is the user interface to the DEC Ada compiler and the `ld` linker. DEC Ada uses the `dbx` debugger and a choice of text editors.

For additional sources of information on DEC Ada, refer to Appendix A.

13.2 VAX BASIC

VAX BASIC is an interactive, shareable language processor for the VMS operating system. This compiler takes full advantage of the VAX floating point, decimal, and character instructions.

With the VAX BASIC compiler, you get a high-performance environment for both application development and timesharing through in-line VAX instructions. It combines the power of a structured programming language with the interactivity of the BASIC environment and the convenience of graphics statements. This compiler is also integrated with key components of the VAX Information Architecture.

13.3 DEC C

DEC C is the name of a family of ANSI C compilers that provides a single language implementation across all strategic Digital platforms (VMS, ULTRIX, and DEC OSF/1). The goal for DEC C is to provide an industry-leading compiler for use as a system and application compiler for Digital systems. DEC C provides support for the American National Standards Institute (ANSI) definition of the C programming language, ANSI X3J11/88–159. DEC C passed the Plum Hall test suite.

A most important feature of the DEC C compiler is its ability to accept different dialects of the C language. With use of command line options, DEC C is compatible with older dialects of C, including common C (Kernighan and Ritchie C) and VAX C. The compiler can be run in the following modes:

- ANSI C mode — compiles the C language as defined by the American National Standard for C along with any extensions not prohibited by that standard
- Relaxed ANSI C mode — follows the ANSI C standard, but also accepts additional Digital keywords and predefined macros that do not begin with an underscore
- VAX C mode — provides compatibility for programs that depend on old VAX C behavior
- Common, or pcc, mode — provides compatibility with the pcc compiler on ULTRIX

The DEC C compiler provides support for function inlining to eliminate call overhead, and source code checking features such as those found in the `lint` utility for assistance in identifying nonportable or unintentional coding practices.

DEC C is an extended implementation of the C programming language originally developed at Bell Laboratories. This compiler runs on the VMS operating system and generates optimized, position-independent code.

The DEC C compiler supports decomposition of **for** and **while** loops. Decomposed loops run in parallel in multiple processes, thereby reducing the total elapsed time required to run a program. This capability is most useful on multiprocessor machines.

In addition to the language, the ANSI C standard also defines the contents of the C library and header files.

13.4 DEC C++

DEC C++ for ULTRIX on RISC systems and DEC C++ for VAX/VMS systems are native compilers which implement the C++ programming language. DEC C++ includes the following components:

- C++ compiler
- DEC C compiler
- Symbolic debugger for both C++ and C programs
- DEC C++ class libraries
- Enhanced ANSI C and XPG3 header files

The DEC C++ compiler implements C++ as defined by *The Annotated C++ Reference Manual* by Ellis and Stroustrup. The DEC C++ implementation includes templates but excludes exception handling. The symbolic debugger, DECladebug, provides basic debugging capabilities as well as many enhancements to aid in debugging C++ applications. (DEC C++ for ULTRIX includes a dbx interface.)

DEC C++ can be used within the DECset or DEC FUSE programming environments.

13.5 DEC Fortran

DEC Fortran is the name of a family of Fortran compilers that provides a single language implementation across strategic Digital platforms (OpenVMS, ULTRIX, and DEC OSF/1). DEC Fortran is an implementation of full language FORTRAN–77 conforming to the ANSI–X3.9–1978 FORTRAN standard. DEC Fortran is highly compatible with VAX FORTRAN.

This compiler includes optional support for programs conforming to the previous standard, ANSI–X3.9–1966. DEC Fortran for ULTRIX meets the Federal Information Processing Standard Publication (FIPS–69–1) requirements by conforming to the ANSI Standard by including a flagger. The flagger optionally produces diagnostic messages for compile-time elements which do not conform to the Full-Level ANSI–X3.9–1978 FORTRAN standard. DEC Fortran for ULTRIX also conforms to the International Standard ISO 1539–1980(E). DEC Fortran conforms to MIL-STD–1753.

DEC Fortran supports extensions to the ANSI standard including a number of extensions defined by the VAX Fortran compiler that runs on VMS and ULTRIX systems.

DEC Fortran includes enhancements to the ANSI standard, including some attributes of Cray pointers, common blocks, additional data types, and composite data declarations using STRUCTURE, END STRUCTURE, and RECORD statements and access to record components through field references.

The DEC Fortran compiler includes a multi-phase optimizer that is capable of performing optimizations across entire programs. The highest level of optimization is on by default. Some specific optimizations include:

- Interprocedural analysis across all modules compiled on the same command line
- In-line expansion of statement functions and routines
- Value propogation
- Constant folding
- Loop unrolling
- Removal of invariant expressions from loops, redundant and unreachable code

13.6 VAX FORTRAN High-Performance Option

To use the new VAX Vector Processors effectively, you need a compiler that can easily take your existing application code and generate code that uses the vector instructions. The VAX FORTRAN compiler, combined with the High-Performance Option (HPO), is designed to do just that for applications on either VAX/VMS or VAX ULTRIX systems.

Digital has offered multiprocessor VAX systems for some time. To simplify programming, VAX FORTRAN supplied mechanisms to more easily exploit the power of VAX multiprocessors. Now, the VAX FORTRAN compiler, combined with the HPO, extends this capability for vector processor systems. It should be noted that there is still only one VAX FORTRAN compiler for VAX/VMS systems: whether scalar or vector and whether a single-processor or multiprocessor configuration.

The HPO capabilities enable the VAX FORTRAN compiler to automatically generate vector-processor instructions. The compiler automatically decomposes programs to improve performance on multiple-processor systems.

And, the VAX FORTRAN HPO on VAX/VMS systems is supported by the VAX Language-Sensitive Editor (VAX LSE) and the VAX Performance Coverage Analyzer (VAX PCA). With this support, Digital's CASE leadership position is extended into the vector processing arena.

The VAX FORTRAN HPO for VAX/VMS systems and for VAX ULTRIX systems includes these enhancements to the VAX FORTRAN compiler:

- Thorough dependence analysis
- Automatic transformations for vectorizing difficult constructs
- Multilevel vectorization and decomposition
- Compile-time performance analysis to choose the best vectorization and decomposition method
- Integration of vector and scalar optimizations
- Generation of VAX Language-Sensitive Editor (VAX LSE) diagnostics for vectorization inhibitors [1]
- Loop summary listing for viewing optimization results

VAX FORTRAN HPO on VAX/VMS systems supports vector processor systems as well as automatic parallel decomposition. VAX FORTRAN HPO on VAX ULTRIX systems has only vector support.

13.7 DEC Pascal

DEC Pascal for RISC and DEC Pascal for DEC OSF/1 are implementations of the Pascal language that complies with the ANSI/IEEE–770X3.97–1983 standard and ISO–7185–1983 (E) Level 0 and Level 1. DEC Pascal is a fully compliant, Class A FIPS 109 Certified Pascal compiler. ISO Level 0 is the ANSI implementation and ISO Level 1 includes support for conformant arrays.

DEC Pascal contains some Domain Pascal extensions, additional BSD Pascal compatibility, and shared libraries. The Pascal language contains control statements, data types, and predeclared procedures and functions. Containing extensions to the standard, DEC Pascal generates optimized code that takes advantage of RISC hardware.

DEC Pascal provides extensions to the standard language definition to provide application developers with a Pascal language that serves a wide range of programs and yet remains consistent with the C programming environment.

To enhance portability, DEC Pascal meets the X/Open Pascal language requirements and and will apply for XPG/3 Branding. DEC Pascal's conformance to these standards reinforces Digital's overall commitment to meet industry standards.

[1] Not available on VAX ULTRIX

13.8 VAX Pascal

VAX Pascal is an implementation of the Pascal language that accepts programs compatible with either level of the ISO specification for programming languages: ISO–7185–1983(E) and ANSI/IEEE–770X3.97–1983. VAX Pascal also meets the Federal Information Processing Standard publication FIPS–109 requirements by accepting programs conforming to the ANSI standard. Furthermore, VAX Pascal accepts many features from the draft Extended Pascal Standard: ANSI/IEEE–770X3.160–1989 and ISO–10206.

The VAX Pascal compiler has been validated for both levels of the ISO Unextended Pascal Standard and for conforming to FIPS–109. With extensions to the standards, the VAX Pascal compiler can generate optimized, shareable code that takes full advantage of the VAX hardware floating point and character instruction sets and the virtual memory capabilities of the VMS operating system.

14 VAXELN Ada

VAXELN Ada, a fully validated implementation of the Ada language, is a realtime software offering that enables you to include VAX Ada programs in a VAXELN system. The VAXELN Ada product is well integrated in the VMS and VAXELN Toolkit environments, where the VMS operating system serves as the development system and the VAXELN kernel serves as the target.

VAXELN Ada systems offer the flexibility, dedication, and reliability required by government agencies for managing radar control, communications, and navigational systems. Industrial uses of VAXELN Ada include the control of shop floor equipment, automation, and robots.

This chapter describes the following:

- VAXELN Ada Software Features, Section 14.1
- VAXELN Ada Programming Support Environment, Section 14.2
- VAXELN Ada Runtime Software, Section 14.3
- VAXELN Ada Development Cycle, Section 14.4
- VAXELN Ada Hardware and Software Requirements, Section 14.5

14.1 VAXELN Ada Software Features

Key features of the VAXELN Ada software include:

- **Compliance with U.S. Government requirements.** VAXELN Ada, combined with the VAX Ada compiler and VAXELN Toolkit, has been validated by the U.S. Government's Ada Validation Office and complies with ANSI/MIL–STD–1815A–1983 (as specified in the *Reference Manual for the Ada Programming Language* by the Ada Joint Program Office).
- **Target system flexibility.** You can defer the choice of a target system until you link your Ada applications, thus making it easier to build applications that run on either VAXELN or VMS systems.

- **Integration with the VAXELN runtime environment.** VAX Ada applications integrate well with the VAXELN runtime environment in the following ways:
  - — They conform to the VAX calling standard, which allows VAX Ada code to call and be called by code written in other VAX high-level languages. VAX Ada code can call VAXELN and VAX Rdb/ELN runtime facilities.
  - — They adhere to standard Ada exception handling within the scope of an Ada routine; they follow standard VAX exception handling outside the scope of an Ada routine.
  - — They can be linked with shared images and access shared memory areas between applications on the same target system.
  - — They can control custom devices. You can write device drivers entirely in Ada high-level language code; no macro code is required.
  - — VAXELN Ada systems run as separate VAXELN processes under control of the VAXELN scheduler. This differs from the VMS environment, in which Ada tasks are multithreaded through the runtime library and the entire program is scheduled as a single process by the VMS scheduler.
- **VAX Ada software development tools foundation.** VAXELN Ada system development complements and builds on the VAX Ada development tools by using the VAX Ada compiler and program library manager.
- **VAX Ada development and runtime support.** VAXELN Ada system development and runtime support includes all of the VAX Ada development and runtime features with the following restrictions:
  - — No access to indexed sequential access method (ISAM) or relative files.
  - — No support for VMS logical names.
  - — No support for the VMS asynchronous system trap (AST)[1] mechanism. Instead, VAXELN system service calls supply this functionality with the additional flexibility required in a realtime environment.
  - — No time-slicing. Instead, VAXELN provides the strictly preemptive, priority-driven scheduler needed in a realtime environment.

[1] An AST is a VMS mechanism for furnishing a software interrupt when an external event occurs.

14.2 VAXELN Ada Programming Support Environment

The VAXELN Ada programming support environment (APSE) offers efficient, controlled, VMS system development of VAXELN Ada systems. APSE consists of the VAX Ada compiler, VAX Ada program library manager, VAXELN Toolkit, and VAXELN Ada Remote Debugger. Sections 14.2.1 through 14.2.4 summarize these tools.

Two additional tools that you will find useful for debugging VAXELN Ada systems include the VAXELN Display Utility and VAXELN Performance Utility. Section 6.3.1 discusses these tools.

Furthermore, you can use Digital CASE tools such as the VAX Language-Sensitive Editor (VAX LSE), VAX DEC/Code Management System (VAX DEC/CMS), VAX DEC/Test Manager, and VAX Source Code Analyzer (VAX SCA) to enhance your productivity. Chapter 17 surveys these and other VMS CASE offerings.

14.2.1 VAX Ada Compiler

You use the VAX Ada compiler to compile VAX Ada programs which subsequently can be included in a VAXELN Ada system image. This compiler features comprehensive diagnostic measures, including automatic syntax error correction (when used with the VAX Language-Sensitive Editor [VAX LSE]), which can facilitate application development. Chapter 13 presents the VAX Ada compiler.

14.2.2 VAX Ada Program Library Manager

You also use the VAX Ada program library manager to develop VAXELN Ada systems. The VAX Ada program library manager furnishes management and utility functions for Ada program libraries.

14.2.3 VAXELN Toolkit

Chapter 4 introduces the various components of the VAXELN Toolkit that you can use to develop VAXELN Ada systems.

14.2.4 VAXELN Ada Remote Debugger

The VAXELN Ada Remote Debugger (which is part of VAXELN Ada) enables VMS system debugging of VAXELN Ada systems running on a VAXELN target. This fully symbolic remote debugger is a superset of the VMS and VAXELN debuggers. To use the VAXELN Ada Remote Debugger, you need a DECnet–VAX license and an Ethernet local area network (LAN) link between the development and target systems.

The VAXELN Ada Remote Debugger recognizes most VAX Debugger commands, including those designed specifically for debugging Ada tasks. In addition, this debugger has VAXELN system-specific commands, which are highlighted in the next sections.

The VAXELN Ada Remote Debugger supports:

- Mixed VAX Ada code and code written in other VAX high-level languages
- Various Digital-supplied VAX Ada and VAXELN library packages that can be used in VAXELN Ada systems
- Ada tasking programs, described next
- VAXELN Ada system monitoring and control commands, also described next

Debug Features for Ada Tasks

When you debug programs involving Ada tasks, you need debugging features beyond those applicable to most programming languages. In addition to general debugging features, the VAXELN Ada Remote Debugger allows you to:

- Display the state of given tasks or all tasks in a program.
- Display more detailed substates of a main state.
- Place a task on hold so that it is not scheduled for execution.
- Perform a task switch.
- Change the priority of a task.
- Abort a task.
- Display tasking runtime statistics, such as number of entry calls, tasks activated, and **accept** and **select** operations.
- Set breakpoints for various tasking events.

Debug Support for VAXELN Ada System Monitoring and Control Commands

The VAXELN Ada Remote Debugger supplies debug support for VAXELN Ada system monitoring and control commands that let you do the following:

- Debug from a VMS development system. Remote console input/output (I/O) allows the VMS system's debugging terminal to act as the target system's console device.
- Display the status of jobs running on the target system and the use and availability of system resources.
- Create new VAXELN jobs from programs that are either resident on the target system or that you add dynamically at runtime.

- Switch from debugging one VAXELN job to another.

14.3 VAXELN Ada Runtime Software

The VAXELN Ada runtime software includes the VAXELN runtime facilities and VAX Ada predefined library packages. Sections 14.3.1 and 14.3.2 offer brief descriptions of this runtime software.

14.3.1 VAXELN Runtime Facilities

The VAXELN runtime facilities can be accessed by an Ada application that is part of a VAXELN Ada system image. Chapters 4 and 6 describe the VAXELN runtime facilities.

14.3.2 VAX Ada Predefined Library Packages

The VAX Ada software is packaged with many standard and supplementary library packages, most of which you can use to develop VAXELN Ada systems. These library packages supply definitions for types, subtypes, constants, bindings, and procedures that you can invoke to interface with the VAXELN runtime facilities.

14.4 VAXELN Ada Development Cycle

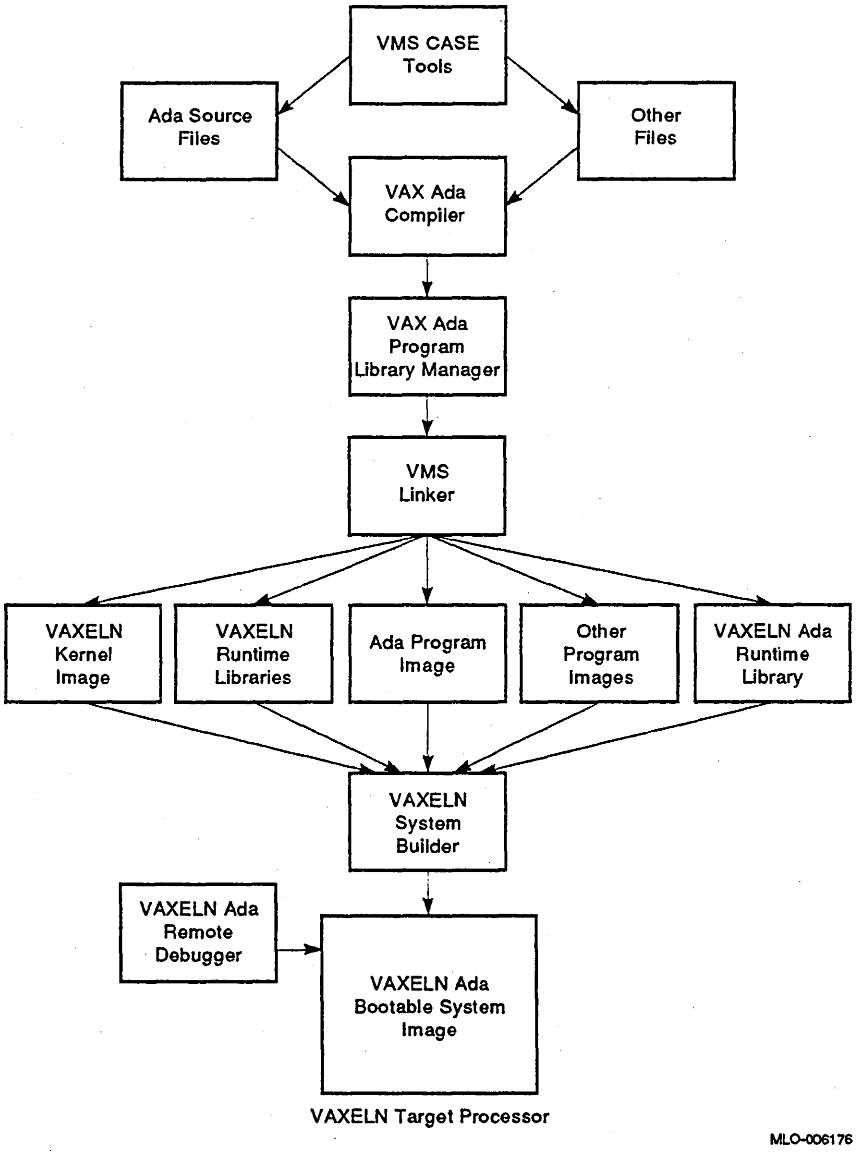
You can use these steps to create a VAXELN Ada system:

1. Write your Ada program on a VMS development system using VAX Ada code.
2. Compile your Ada program using the VAX Ada compiler.
3. Link your Ada program image using the VAX Ada program library manager interface to the VMS Linker.
4. Build your VAXELN Ada system image using the VAXELN System Builder to combine one or more Ada program images, any other program images, runtime libraries, and a VAXELN kernel image.
5. Load and boot your VAXELN Ada system image onto a target processor by a downline system load, disk or tape device, or read-only memory (ROM).

   To downline load your system image, you must have a DECnet–VAX license and Ethernet hardware configured into your system.

Figure 14–1 illustrates how to build a typical VAXELN Ada system using the VAX Ada compiler, VAXELN Toolkit, VMS CASE tools, and VMS development tools in conjunction with VAXELN Ada.

Figure 14–1 Developing a Typical VAXELN Ada System



14.5 VAXELN Ada Hardware and Software Requirements

VAXELN Ada hardware and software requirements and options are specified next.

Development Hardware Requirements

The VAXELN Ada *System Support Addendum (SSA)* includes a list of supported VAXELN Ada development system processors. Contact your Digital sales representative to obtain the latest VAXELN Ada *SSA*.

Target Hardware Requirements

VAXELN Ada target processors include any VAX or MicroVAX target supported by the VAXELN software. For a list of supported VAXELN target processors, see the latest VAXELN Toolkit *System Support Addendum (SSA)*, also available from your Digital sales representative.

Development Software Requirements

To develop a VAXELN Ada system, the following software is required:

- VAX Ada compiler
- VAXELN Ada
- VAXELN Toolkit
- VMS operating system
- DECnet–VAX

Optional Hardware and Software

You may also want to acquire a DECnet–VAX license and Ethernet hardware. When configured into your system, these ingredients will enable you to downline load your VAXELN Ada system image from the development system to the target and to use the VAXELN Ada Remote Debugger.

15 XD Ada Cross-Development System

The XD Ada product set is a family of production-quality Ada cross-development tools for embedded realtime systems projects. This software generates optimized, relocatable code for Motorola MC68*xxx* and MIL–STD–1750A microprocessor targets. XD Ada is presently targeted to the MC68000, MC68020, MC68030, MC68040, MC68332, MC68340, and MIL-STD_1750A bare microprocessor. XD Ada has been developed jointly by Digital Equipment Corporation and SD (part of the SD–Scicon group) to address the needs of the rapidly increasing number of military and commercial organizations committed to using the Ada language in embedded realtime systems projects.

This chapter describes the following:

- XD Ada Software Features, Section 15.1
- XD Ada Programming Support Environment, Section 15.2
- XD Ada Runtime Software, Section 15.3
- XD Ada Development Cycle, Section 15.4
- XD Ada Hardware and Software Requirements, Section 15.5

For additional sources of information on XD Ada, refer to Appendix A.

15.1 XD Ada Software Features

Key features of the XD Ada software include the following:

- **Complete validation.** The XD Ada software is a validated implementation of the full ANSI/MIL–STD–1815A–1983 Ada language, specified in the *Reference Manual for the Ada Programming Language*.
- **Extension to the VAX Ada compiler.** The XD Ada cross-development system is the natural choice for current VAX Ada users. The XD Ada toolset, in conjunction with the VAX Ada compiler, extends this unrivaled development environment to embedded realtime systems.

- **VMS development, testing, and debugging environment compatibility.** XD Ada system development allows VMS host development, testing, and debugging of your Ada applications using the VAX Ada compiler, VAX Ada program library manager, VAX Debugger, VMS development tools, and VMS CASE tools. Then, you use the XD Ada toolset to compile your application for a different target system. This approach enables the development tools to exploit the host's effectiveness while optimizing the target computer for the embedded application.
- **Application development without target hardware.** Because applications that can be included in XD Ada systems can be developed completely on a VMS host, software development can start before the target hardware is available. XD Ada also furnishes support for emulators. Emulators are devices or programs that allow application execution on a different type of computer than the one on which an application was developed or for which it was written.
- **Reduced development and runtime costs.** Two features of the XD Ada compiler can reduce overhead on the host and target:
  - The cross-compiler's compilation rate approaches that of the VAX Ada compiler. It is based on Digital's use of efficient file and memory management combined with memory-resident data structures.
  - The cross-compiler's compact code generation is based on global optimization techniques, pattern-matching instruction selection, and a global register allocator.

  In addition, compatible host and target development and runtime environments enable efficient use of staff and equipment resources.
- **Comprehensive diagnostic messages.** When used with the VAX Language-Sensitive Editor (VAX LSE), the XD Ada compiler features comprehensive diagnostic messages, including automatic syntax error correction.
- **Target system-dependent application development support.** Since different target systems vary in characteristics such as the size of storage units, memory size, and the smallest and largest integers supported, XD Ada supplies target-specific application development support. This software support consists of a target system-dependent, predefined library package as well as length, data representation, enumeration representation, record representation, and address clauses.

- **Reconfigurable target runtime system.** The XD Ada software is packaged with a reconfigurable, modular runtime system featuring optimized rendezvous performance. Rendezvous refers to the interaction that occurs between two parallel Ada tasks. During this interaction, one task calls an entry of the other task, and a corresponding **accept** statement is executed by the other task on behalf of the calling task. Rendezvous performance is a measure of the time it takes for a rendezvous between two Ada tasks to complete.

15.2 XD Ada Programming Support Environment

The XD Ada programming support environment consists of a set of software tools that provide efficient, controlled, VMS host development of XD Ada systems. The XD Ada toolset currently supports the Motorola MC68*xxx* and MIL–STD–1750A target microprocessor environments. Each toolset contains a target-specific version of the XD Ada cross-compilers, library of predefined compilation units, program library manager, macro assemblers, librarian, builder, loader, formatter, **run** command, and XD Ada Debugger. Sections 15.2.1 through 15.2.10 introduce these components.

You can also use a variety of VMS tools and utilities in conjunction with VMS CASE tools to facilitate XD Ada program development and testing. Such CASE tools include the VAX Debugger, VAX DEC/Code Management System (VAX DEC/CMS), VAX DEC/Test Manager, VAX Language-Sensitive Editor (VAX LSE), VAX Source Code Analyzer (VAX SCA), and various VAX Information Architecture products. Chapter 17 presents these and other VMS CASE products.

15.2.1 XD Ada Cross-Compilers

XD Ada cross-compilers are full ANSI/MIL–STD–1815A cross-compilers targeted to the Motorola MC68*xxx* and MIL–STD–1750A microprocessors. The XD Ada cross-compilers generate optimized, relocatable object code from applications written in Ada. These compilers also flag implementation-dependent features that could result in nonportable code.

15.2.2 XD Ada Library of Predefined Compilation Units

This library contains standard program library units for use in Ada applications, including those specified in the *VAX Ada Language Reference Manual.*

15.2.3 XD Ada Program Library Manager

This program library manager provides the following features:

- An extended version of the VAX Ada program library manager.
- A consistent control mechanism for both host and target XD Ada components.
- Coordination of target-independent Ada source code. This is accomplished by maintaining target-dependent libraries that contain separate object code versions for each different target computer.
- User interface to the XD Ada cross-compiler and builder.

15.2.4 XD Ada Macro Assemblers

These assemblers are targeted to the Motorola MC68*xxx* and MIL–STD–1750A microprocessors. The XD Ada macro assemblers translate assembly language into object code for the destined target processor.

15.2.5 XD Ada Librarian

The librarian creates and maintains assembly language object code libraries of files assembled with the XD Ada macro assembler. It also creates and maintains XD Ada macro libraries, which include macro definitions used by the XD Ada macro assembler.

15.2.6 XD Ada Builder

The builder links object files produced by both the XD Ada cross-compiler and macro assembler, and automatically links in the XD Ada target runtime system (XD Ada RTS). The result is a loadable, executable program image of the complete XD Ada system. The XD Ada builder requires a target definition and a mapping definition.

15.2.7 XD Ada Loader

The loader downline loads an Ada executable program image onto the target by means of one of the host-to-target communication links described in Section 15.4.

15.2.8 XD Ada Formatter

The formatter reformats an Ada program image so that it can be transferred to the target without the loader.

15.2.9 XD Ada Run Command

The run command, **xdrun**, starts execution of an Ada program image on the target.

15.2.10 XD Ada Debugger

This debugger is a Digital-developed, retargeted version of the VAX Debugger. The XD Ada Debugger offers window-based host debugging at Ada source, assembler, and machine code levels on all XD Ada-supported targets. Source-level debugging support is high-level and fully symbolic. The XD Ada Debugger furnishes the same environment for target debugging as that provided by the VAX Debugger on the host.

15.3 XD Ada Runtime Software

The XD Ada runtime software includes the XD Ada target runtime system (XD Ada RTS) and XD Ada target debug kernel.

15.3.1 XD Ada Target Runtime System

The XD Ada RTS supports the full Ada language without a target operating system. Only runtime modules used by an application are included at build time.

The XD Ada RTS is supplied in source and object form. While the RTS is preconfigured for a standard target, it is designed to enable reconfiguration for specific target-variant configurations. The RTS comes preconfigured for these target hardware configurations:

- Motorola MVME133XT VMEmodule monoboard microcomputer (for Motorola MC68*xxx* microprocessor support)
- Fairchild SBC–50 board using the F9450 microprocessor (for MIL–STD–1750A microprocessor support)

15.3.2 XD Ada Target Debug Kernel

The target debug kernel is the XD Ada Debugger target component. It is a small program that runs separately from the XD Ada program, allowing host tools to load, start, and debug the XD Ada program image on the target. The XD Ada target debug kernel can be removed when downloading and debugging are no longer required.

15.4 XD Ada Development Cycle

You create an XD Ada system on a VMS host by simply using the XD Ada toolset. However, if the target hardware is not yet available, you may want to start initial development, testing, and debugging using the VAX Ada compiler, user-defined test suites, and the VAX Debugger. Later, you can recompile your application for a different target using the XD Ada cross-compiler. To create an XD Ada system in this way, follow these steps:

1. Write your Ada application (which is to be included in an XD Ada system) on a VMS host using VAX Ada code. You have the option of developing part of your application in macro assembly language.
2. Compile your Ada application using the VAX Ada compiler.
3. Link your program image using the VAX Ada program library manager interface to the VMS Linker.
4. Execute your program image on the VMS host system under the control of the VAX Debugger.
5. Test the dynamic behavior of your application on the VMS host system by means of user-defined test suites in conjunction with VMS CASE tools.
6. Use the XD Ada program library manager and XD Ada librarian to create target-specific libraries.
7. Use the XD Ada compiler (which fully compatible with the VAX Ada compiler) to recompile your Ada application for the target computer. Specify the target memory configuration (that is, the location, size, and type of each memory area) and the mapping of Ada and assembly units onto these memory areas.

   If you have written part of your application in macro assembly language, assemble this code using the XD Ada macro assembler.
8. Use the XD Ada builder (which uses the target memory configuration and memory mapping specified earlier) to create an executable program image.
9. Downline load your program image to the target computer using the XD Ada loader.

   Alternatively, use the XD Ada formatter to reformat your program image. You can use this reformatted file with programmable read-only memory (PROM) programming equipment or for loading across industry-standard interfaces into microprocessor simulators and emulators.
10. Use the XD Ada Debugger to control and monitor the execution of the Ada program on the target system.

11. XD Ada-supported simulators and emulators offer additional facilities for target software testing and debugging.

Note

An RS–232–C, DR11–W parallel-line, or DRV11–WA parallel-line interface is required for program loading and debugging from the host system.

Figure 15–1 shows how you can use the VAX Ada compiler, VMS CASE tools, and VMS development tools in conjunction with the XD Ada toolset to build a typical XD Ada system.

XD Ada Host-to-Target Communication Link Options

Three methods of communication are available for transferring data from the XD Ada host development system to the target. You can use any of these communication standards, provided there is a compatible host facility. The communication facilities available are:

- Standard RS–232–C port connected to a VAX terminal interface
- Fast parallel link to a DR11–W or DRV11–WA parallel-line interface
- VMS mailbox interface

15.5 XD Ada Hardware and Software Requirements

XD Ada hardware and software requirements are specified next.

Host Hardware Requirements

The XD Ada *System Support Addendum (SSA)* includes a list of supported XD Ada host processors. Contact your Digital sales representative to obtain the latest XD Ada *SSA*.

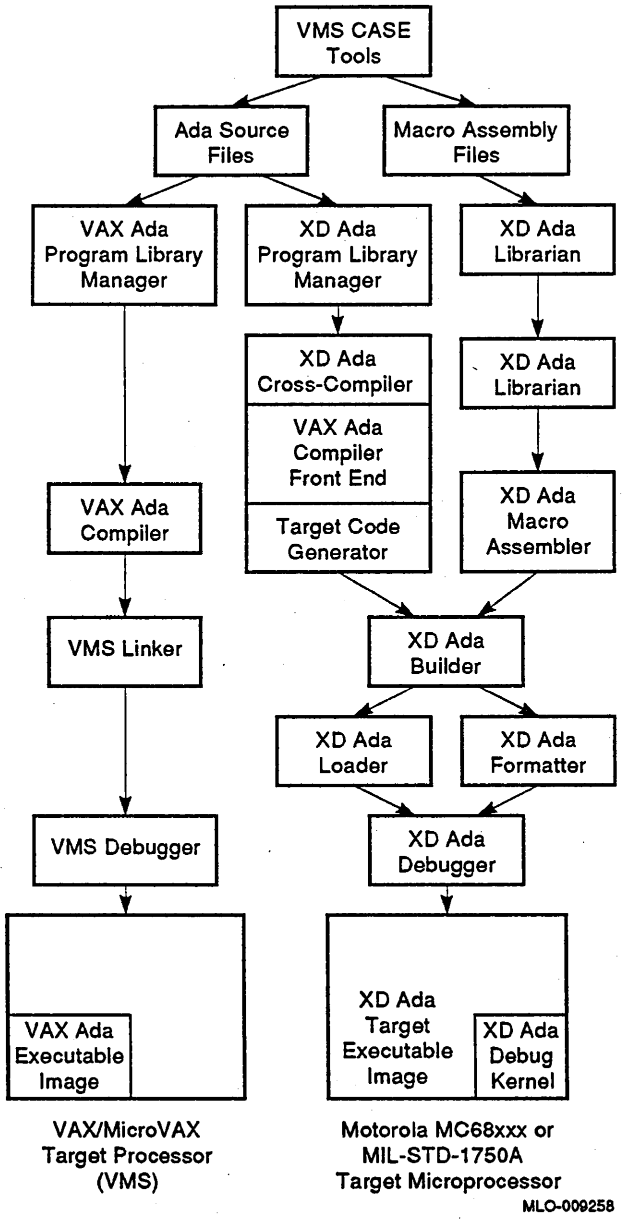
Target Hardware Requirements

The XD Ada toolset is targeted to the Motorola MC68*xxx* and MIL–STD–1750A microprocessors.

Host Software Requirements

The VMS operating system is required for host development of an XD Ada system.

Figure 15–1 Developing a Typical XD Ada System



Part VI

The Digital COHESION Environment

Part VI introduces the Digital COHESION environment and surveys many key Digital tools that compose the COHESION solution. This part contains the following chapters:

- Chapter 16, Overview of the COHESION Environment, presents the Digital COHESION environment and strategy.
- Chapter 17, VMS CASE Tools in the COHESION Environment, describes VMS CASE tools, including several tools for structured analysis, structured design, and realtime modeling.
- Chapter 18, UNIX CASE Tools in the COHESION Environment, Environment, describes UNIX CASE tools including DECset, FUSE EnCASE and the C++ Support Kit.
- Chapter 19, DECfactory Products in The Realtime Environment, describes the DECfactory services and products designed to provide manufacturers with accurate, timely information and management capabilities needed to achieve manufacturing excellence.
- Chapter 20, DECmessageQ in the Realtime Environment, describes DECmessageQ as a flagship NAS product for application integration. DECmessageQ offer a flexible approach to application integration through interprocess message queuing across multiple environments.

16 Overview of the COHESION Environment

Software development and maintenance costs are often the largest single expense in a realtime project. Most scientific software used today is developed for custom research and laboratory applications. In the aerospace and government markets (where long-term, large-scale projects are common), software development and maintenance costs dominate hardware-acquisition costs over the life cycle of major projects. With global competition forcing the scientific community to reduce the overall time to market, software development is becoming a key competitive factor.

Two major problems thwart the efficiency of software development today. First, the phases of the development cycle (from analysis and design to coding, testing, documentation, and ongoing maintenance) are not well integrated. Consequently, the effects of change are difficult to manage, both within and across the phases, which results in more wasted time and money. Second, most large organizations have acquired multiple hardware and software platforms over the years. Various departments conduct their work in isolation. As a result, similar wheels may be invented time after time. This approach wastes an organization's time and money.

Successful research organizations are increasingly using computer-aided software engineering (CASE) tools to reduce the time and effort required to write application code, and leave more time for product research. Digital's CASE solution, the Digital COHESION environment, is the industry's most comprehensive and inclusive CASE environment for developing, using, and managing software. The COHESION environment also furnishes a highly efficient, full-service development environment for developing realtime applications.

Discrete manufacturers are under competitive pressure to lower costs, improve quality, and meet difficult customer delivery requirements. Their manufacturing processes are becoming increasingly complex and difficult to manage and they are besieged by an increasing burden of compliance with environmental procedures and reporting requirements. Digital's comprehensive, worldwide DECfactory services and products provide integrated factory floor solutions for all styles of manufacturing in the Aerospace, Electronics, Automotive and General Discrete (AGD) Industries.

This chapter describes the following:

- Digital's Vision of the COHESION Environment, Section 16.1
- Introduction to the Digital COHESION Environment, Section 16.2
- Components of the Digital COHESION Environment, Section 16.3

For additional sources of information on the Digital COHESION environment for CASE, refer to Appendix A.

16.1 Digital's Vision of the COHESION Environment

Digital's vision for software development is to enable organizations worldwide to develop, deploy, and manage software in the multivendor environment. Encompassed in this vision are all industries and markets, all project sizes, and all styles of computing. To realize this vision, the strategic goals of the COHESION solution include:

- **Full range of quality software.** Build a full range of quality software, spanning enterprisewide information systems to services and products that can run on multiple platforms.
- **Software to implement corporate goals.** Make software an asset in implementing corporate direction and goals by linking business planning to technology planning.
- **Common software development environment.** Offer a common environment for developing and running software across the enterprise, with specific solutions to satisfy the needs of commercial, technical, industrial, and product development segments.
- **Integrated software development platform.** Provide an integrated software development platform so that customers can choose the best software from Digital and other vendors to meet their needs.
- **Highest level of service.** Provide the highest level of service, enabling organizations to move forward with new technologies and practices.

16.2 Introduction to the Digital COHESION Environment

One of the fundamental goals of the COHESION solution is to enable a wide spectrum of software developers to create and maintain high-quality software efficiently. By adhering to industry standards, Digital can offer products that facilitate application portability across platforms. Industry standards also allow Digital to create integrated applications with functions that can be distributed across a multivendor network. Using a strategy grounded in support for standards, both independent software vendors and Digital can integrate components into the COHESION environment.

The Digital COHESION solution is composed of integrated products and services that support the entire software development life cycle—from planning to maintenance and evolution. This environment provides integration within a project, within a department, and within an enterprise. Hence, groups can link software development with the business plans not only of individual projects and departments, but of the entire organization.

Comprehensive services and support put Digital's CASE expertise and experience to work for you. Multilevel consulting, training, and support services assist you throughout the entire computing life cycle. This support helps ensure the effective implementation of Digital solutions in your enterprise.

16.3 Components of the Digital COHESION Environment

To meet the objectives outlined in Section 16.1, the Digital CASE strategy is built on three elements, which are described in the next sections:

- **Comprehensive services and support.** A set of services that enable organizations to implement better software development technologies and practices.
- **Network Application Support (NAS).** A set of architectural standards-based interfaces and components that enable multivendor software development and application integration.
- **Tools for Software Life Cycle and Management.** A set of tools that automate all software development life-cycle and management functions.
- **Tools for Factory Management.** An offering of products and services to provide manufacturers with protection for their investments.

16.3.1 Comprehensive Services and Support

Digital's CASE products are backed by comprehensive services and support (including consulting, training, and maintenance) to ensure effective implementation. Through Digital's Enterprise Integration Centers, systems integration services are offered to support enterprisewide CASE environments. These services help businesses tie their product, service, and business goals directly into the software development and management capabilities furnished by Digital and Complementary Solutions Organizations (CSOs).[1] Significant benefits can be gained by using these services prior to selecting tools.

16.3.2 Network Application Support (NAS)

Network Application Support (NAS) services implement the application layer of Digital's system architecture to facilitate interoperability and portability in a distributed, networked environment. NAS services allow you to create, use, and maintain integrated applications with functionality distributed across hardware and operating system platforms from multiple vendors. With NAS services, you can also build applications that are more easily ported across platforms and devices. Not only are the COHESION tools built upon NAS services so that they work in a distributed multivendor environment, but they are also used to build NAS applications.

The base integration furnished by NAS services and the NAS principle of compliance to standards makes tools within COHESION easy to use and widely accessible. DECwindows and DECmessageQ, NAS components, illustrate this point.

DECwindows in NAS

DECwindows extends the capabilities of the industry-standard X Window System, Version 11, by supplying powerful software that simplifies the development of graphics-oriented applications. Using DECwindows, you have a consistent window or means through which to view different applications. For further information on DECwindows and the X Window System, refer to Chapter 7.

1 A CSO is a leading third-party application supplier that allies with Digital through the Cooperative Marketing Program (CMP) and through joint marketing, joint product development, system reseller, or Complementary Software House (CSH) agreements. Digital and its CSOs share product goals and directions, supplying Digital's customers with the best integrated solutions on the market.

DECmessageQ in NAS

DECmessageQ offers a common programming interface (API) across different operating systems and networks, simplifying the integration of new and existing applications. DECmessageQ uses the client/server architecture to facilitate a simple and reliable message queuing mechanism so that applications running on different computers can share data. With DECmessageQ, program modules can be developed or deployed anywhere in the DECmessageQ network configuration, whether in stand-alone mode, a VAXcluster, a LAN, or at remote sites. The queued message bus structure of DECmessageQ means applications do not have to establish unique links to each of their application partners. One system acts as the message server to exchange messages among client implementations on VMS, ULTRIX on a VAX or MIPS-based system, MS–DOS, and OS/2 systems. For further information on DECmessageQ, refer to Chapter 20.

Many NAS component products are now available as a single product. NAS 300, for example, is a factory-installed NAS product available on selected VAX 4000 systems and the MicroVAX 3100 Model 80.

16.3.3 Tools for Software Life Cycle and Management

The Digital COHESION solution defines two major categories of CASE tools. The first set, life cycle tools, focuses on facilitating and automating software development and maintenance. The second set, management tools, addresses management of the processes and procedures that deal with software.

Digital offers a comprehensive portfolio of industry-leading CASE tools. These tools address all aspects of software development and maintenance for business, scientific, technical, and embedded applications. Digital's CASE offerings consist of design and analysis tools and products that cover documentation, coding and testing, configuration management, data management, communications, and expert system shells. Digital's CASE products help improve software reliability and increase software development productivity.

16.3.4 Tools for Factory Management

Digital's DECfactory services and products combine with third-party VMS and UNIX applications to provide manufacturers with protection for their investments. By integrating DECfactory products with legacy systems, such as Manufacturing Resource Planning (MRP II), shop floor control, time and attendance, and cell control, DECfactory solutions support Digital's Open Advantage Campaign for Manufacturing (OAM).

DECfactory solutions provide answers to a wide range of factory floor problems. By filling the gap between MRP II systems and automation equipment, such as Programmable Logic Controllers (PLCs) and robots, DECfactory solutions capture, use, and store data for scheduling, work-in-process tracking, maintenance, and quality monitoring and analysis. Digital's manufacturing customers can increase production flexibility, reduce paperwork, improve quality control, access and share a common set of manufacturing data, lower costs, react more appropriately to changes on the factory floor, and meet government regulations more consistently.

17 VMS CASE Tools in the COHESION Environment

This chapter highlights the key VMS CASE and NAS tools that form the Digital COHESION environment. Included in this survey are DECdesign and several third-party tools for structured analysis, structured design, and realtime modeling.

The tools in this chapter are arranged as follows:

- VMS DECset CASE Tools, Section 17.1
- Network Application Support (NAS) for VMS, Section 17.2
- Additional CASE Tools, Section 17.3
- Third-Party VMS CASE Tools, Section 17.4

For additional sources of information on Digital's CASE offerings, refer to Appendix A. You can also contact your Digital sales representative to obtain the latest *Software Product Description (SPD)* and *System Support Addendum (SSA)* for each software tool.

17.1 VMS DECset CASE Tools

DECset, an integrated group of CASE tools, creates an environment in which you work on a single system rather than with a series of unrelated tools and files. The DECset user environment includes Motif user interfaces, a consistent layout, and automatically sets up routine programming tasks for the user. DECset tools are ideal for facilitating teamwork among groups of programmers. This section describes the following DECset tools:

- DEC Code Management System
- DEC Module Management System
- DEC Test Manager
- DEC Language-Sensitive Editor

- DEC Performance and Coverage Analyzer
- DEC Source Code Analyzer
- Integrated Symbolic Debugger Support

17.1.1 DEC Code Management System

DEC Code Management System (CMS), an online file librarian, helps track software development and maintenance. It furnishes a method of storing and organizing groups of ASCII files into a project library, tracking changes and monitoring access to the library.

CMS allows you to:

- Retrieve, modify, and test files in your own directory
- Obtain an historical account of file modification, including when, why, and by whom the modification was made
- Determine the origin of each line of a file, either as an annotated listing or as comments in the file
- Manage concurrent file modifications
- Merge separately developed file modifications
- Create successive versions ("generations") of files
- Compare two generations of a file within a library
- Retrieve previous file generations
- Organize related library files into groups
- Define a set of file generations as a class to make up a base level or release version of a project

CMS allows mulitple team members to store and track all changes to files. CMS can be used as part of a repository-based solution for large scale team efforts.

17.1.2 DEC Module Management System

The DEC Module Management System (MMS) is a productivity tool that automates and simplifies the building of software systems. This tool tracks interdependencies and component modifications. You can choose to have MMS rebuild only those components that have changed since the last build or have it rebuild an entire system. In addition, MMS can interact with CMS to offer an enhanced software development package.

17.1.3 DEC Test Manager

The DEC Test Manager (DTM) is a flexible, automated regression-testing system for use during both the development and maintenance phases of the software life cycle. This tool automates the organization, execution, and review of test results and allows several developers to use one set of tests concurrently.

Integration of the DEC Test Manager with the DEC Performance and Coverage Analyzer (PCA) and CMS supplies a further enhanced testing environment, with corresponding productivity gains.

17.1.4 DEC Language-Sensitive Editor

The DEC Language-Sensitive Editor (LSE) is a powerful, multilanguage, multimodule, multiwindow, screen-oriented text editor specified for program design, development, and maintenance. The editor is language sensitive in that it provides templates of major constructs in support of a wealth of languages from Digital, including Ada, BASIC, BLISS, C, COBOL, DIBOL on VMS, FORTRAN, Fortran HPO, MACRO, Pascal, PL/I, SCAN, and VAXELN Pascal.

This NAS component, template feature encourages productive programming by fostering faster and more accurate code development. Using the Motif interface, you can write code and then edit, compile, review, and correct compilation errors in the same session. The editor is also extendable, allowing customization to fit your programming needs.

LSE is tightly integrated with the programming development environment. It is invoked on the command line and works in concert with supported Digital languages and other CASE tools. On VMS, LSE works with the VAX Symbolic Debugger, CMS, and SCA to render a highly interactive program development environment. LSE facilitates the edit-compile-debug-analyze portion of the program development cycle. CMS files can be reserved automatically and replaced in a library during an editing session. And, all SCA commands are available through LSE.

DECset includes enhancements to LSE, SCA, and Digital language compilers that support detailed or module design. The Program Design Facility permits you to express a design using pseudocode, capture the design information, and supply various reports that list design cross-references, call trees, and the use of data structures. Users can verify program designs early instead of finding errors after the application is coded.

LSE can also be called from the DEC Performance and Coverage Analyzer (PCA) and the VMS Mail Utility.

17.1.5 DEC Source Code Analyzer

The DEC Code Analyzer (SCA) is an interactive, multilanguage, multimodule, source code cross-reference and static-analysis tool. Designed to clarify the complexities of entire software systems, this tool benefits both the implementation and maintenance phases of application development.

SCA is tightly integrated with LSE and may be viewed as a logical extension of the editor's powerful error prevention, detection, and correction facilities.

SCA has a new graphical call tree and data structure information. The mouse-sensitive displays focus LSE on corresponding source code and static analysis of the program to give users a more intuitive understanding of the application.

17.1.6 DEC Performance and Coverage Analyzer

The DEC Performance and Coverage Analyzer (PCA) is a fast, multilanguage, source-profiling tool designed to help you analyze and improve the execution behavior of application programs. A new graphical user interface allows users to click on easy-to-understand options and graphically display collected data. It also offers test-coverage analysis by measuring the parts of a user program that are executed or not executed by a given set of test data.

PCA can be used to analyze applications with a Motif interface. PCA automatically highlights the differences down to a single pixel and identifies which segments of code have been executed during testing.

PCA consists of two parts: the Collector and the Analyzer. The Collector gathers performance or test-coverage data as a program runs and stores it in a data file. The Analyzer then processes the collected data and presents it either in tables or histograms.

17.1.7 Integrated Symbolic Debugger Support

The VAX Symbolic Debugger that is bundled with the VMS operating system is also included as a component of DECset for VMS. By allowing you to interactively observe and manipulate a program as it executes, the VAX Debugger aids in locating runtime programming or logic errors.

17.1.8 DECset for VMS Software Requirements

This section identifies software requirements for developing and running applications in a DECset for VMS environment.

VMS Version 5.4 or higher is required for installing the DECset software and developing applications using DECset tools. In a VMS workstation environment, DECset requires VMS DECwindows Motif Version 1.0 or higher.

17.2 Network Application Support (NAS) for VMS

Digital's Network Application Support (NAS) products provide solutions to the problems created by the complexity of today's computing environment.

NAS is a comprehensive set of standards-based software that consists of well-defined programming interfaces, toolkits, and products to help developers build applications that are integrated and easily ported across a distributed, multivendor environment. Two major components comprise the NAS solution, as follows:

- Distribution of resources across the enterprise, network, and multivendor systems
- Integration of components within the information system and with other information systems.

NAS helps to insulate applications from changes, including changes in hardware platforms and operating systems, and the need for different network protocols. NAS does this by providing the infrastructure from which applications can draw common services, masking the differences among the underlying platforms. The NAS infrastructure, along with a set of tools to help develop and maintain applications that are built upon it, forms a complete, platform-independent, virtual system.

VMS NAS component products are bundled into complete sets targeted for specific environments, as follows:

- NAS 200 for VMS: Integration at the Desktop
- NAS 250 for VMS: VAXstation Integration
- NAS 300 for VMS: Distributed Client/Server Computing
- NAS 400 for VMS: Critical Large Business Operations

17.2.1 NAS 200 for VMS: Integration at the Desktop

NAS 200 for VMS provides a complete set of networking and distributed computing capabilities for basic print, file, and data-sharing services for PCs, Macintosh, and workstations. It is targeted for environments where applications run primarily on the desktop and where users of these applications need to share files, printers, and data.

The following software and product components are included with NAS 200 for VMS:

- DECnet–VAX Extensions
- PATHWORKS for VMS

- PATHWORKS for Macintosh
- Remote System Manager Client
- Rdb/VMS Runtime Option
- VMS/ULTRIX Connection (UCX)

NAS 200 for VMS supports key standards such as OSI, LAN Manager, TCP/IP, NFS, and SQL.

17.2.2 NAS 250 for VMS: VAXstation Integration

NAS 250 for VMS provides a complete set of NAS software to enable the new VAX workstations to fully integrate into a client/server or distributed environment.

The following software and product components are included with NAS 250 for VMS:

- DECnet–VAX EN
- DECnet–VAX Extensions
- Rdb/VMS Runtime Option
- VMS/ULTRIX Connection (UCX)
- VAXcluster software
- VMS DECwindows Motif
- DEC ACA Services Runtime Library

NAS 250 for VMS supports key standards such as OSF's Motif, OSI, TCP/IP, NFS, LAN Manager, OMG/ORB, and SQL.

17.2.3 NAS 300 for VMS: Distributed Client/Server Computing

NAS 300 for VMS integrated software products provide a complete set of runtime services for client/server, distributed, or general host-based applications. They provide integration capabilities including distributed user-interface facilities, object-oriented application linking and control, support for creating multimedia documents and management of multimedia data, and desktop integration.

The following software and product components are included with NAS 300 for VMS:

- DECnet–VAX Extensions
- PATHWORKS for VMS

- PATHWORKS for Macintosh
- Remote Ssytem Manager Client
- Rdb/VMS Runtime Option
- VMS/ULTRIX Connection UCX
- ALL–IN–1 Mail Server (X.400)
- DEC ACA Services VMS Runtime Library
- DECforms Runtime System
- DECmessageQ for VMS Runtime Option
- VMS DECwindows Motif

NAS 300 for VMS supports key standards such as OSI, TCP/IP, OSF's Motif, NFS, LAN Manager, X Windows, FIMS, ODA, X.400, OMG/ORB, and SQL.

17.2.4 NAS 400 for VMS: Critical Large Business Operations

NAS 400 for VMS offers additional NAS and system software for environments that require higher levels of availability, manageability, and reliability for transaction processing and other mission-critical applications. NAS 400 supports performance tuning and capacity planning, dosk shadowing, clustering, journaling, transaction processing, application distribution and integration, and desktop integration.

The following software and product components are included with NAS 400 for VMS:

- DECnet–VAX Extensions
- PATHWORKS for VMS
- PATHWORKS for Macintosh
- Remote System Manager Client
- Rdb/VMS Runtime Option
- VMS/ULTRIX Connection (UCX)
- ALL–IN–1 Mail Server (X.400)
- VAX ACA Services VMS Runtime Library
- DECforms Runtime System
- DECmessageQ for VMS Runtime Option
- VMS DECwindows Motif

- VAX ACMS Runtime Library
- DECtrace Runtime Library
- VAX RMS Journaling
- VAXcluster Software
- Volume Shadowing
- DECps Data Collector

NAS 400 for VMS supports key standards such as OSI, LAN Manager, TCP/IP, NFS, SQL, OSF's Motif, X Windows, OMG/ORB, FIMS, ODA, and X.400.

17.3 Additional CASE Tools

This section describes the following additional CASE tools available from Digital:

- DEC Graphics Kernel System
- DEC PHIGS
- DECdesign
- DECgraph
- VAX Notes
- VAX Performance Advisor
- VAX Software Performance Monitor
- VMS DECwindows Motif
- VMS Workstation Software

17.3.1 DEC Graphics Kernel System

With the DEC Graphics Kernel System (DEC GKS) development tool, you can produce portable, device-independent graphics applications that create computer-generated pictures.

DEC GKS is a subroutine library (packaged as a shareable image) that implements the ISO (IS–7942) and ANSI (ANS–X3.124–1985) GKS standard for two-dimensional, device-independent graphics. This tool conforms to level 2c of the GKS standard, which provides full output capabilities (including workstation-independent segment storage, level 2) and full input capabilities (synchronous and asynchronous input, level c).

17.3.2 DEC PHIGS

The DEC Programmers Hierarchical Interactive Graphics System (DEC PHIGS) is a sophisticated, three-dimensional, graphics support system. This tool manages the definition, modification, and display of hierarchical graphics data that is stored in a conceptually centralized database. A key feature of DEC PHIGS is its device independence, which means that the same program can generate graphical output on different devices without modification to the source code.

DEC PHIGS is Digital's implementation of the ANSI and ISO PHIGS Standard for three-dimensional, device-independent graphics.

17.3.3 DECdesign

DECdesign, a DECwindows product, graphically supports the analysis and design phases of the software development life cycle. As a multiuser system, DECdesign is capable of supporting large or small distributed development teams. It features a single DECdesign database with concurrent access, data security, and permanent data storage.

DECdesign-supported modeling techniques include Yourdon Structured Design and Gane & Sarson for process modeling, Ward-Mellor extensions to Yourdon Structured Design for realtime modeling, and Extended Entity Relationship (EER) for data modeling.

Based on object-oriented technology, DECdesign is integrated with the VAX Common Data Dictionary/Repository (VAX CDD/Repository). Selected data definitions stored in VAX CDD/Repository can be reused by importing and linking to models produced in DECdesign. Likewise, selected data definitions created in DECdesign can be exported and linked to VAX CDD/Repository.

A key benefit of DECdesign is the ability to create, modify, and reuse the results of the analysis and design phases. This results in more cost-effective development and lower maintenance.

DECdesign produces Compound Document Architecture (CDA)-compliant reports of the analysis and design effort (through predefined report templates) that can be edited in other Digital CDA products, such as DECwrite.

17.3.4 DECgraph

DECgraph is a general-purpose graphics plotting package that allows you to create, change, display, and print graphs.

17.3.5 VAX Notes

VAX Notes is a computer-conferencing software product designed to let you create and access online conferences and meetings. Computer conferencing is an electronic messaging technology that allows you to conduct meetings by computer with people in different geographic locations. Participants can join in a discussion from their own desks when they choose.

Another feature of VAX Notes is its detailed record keeping. Proceedings can be searched by an assortment of criteria including name of participant, subject, or keyword.

You can use VAX Notes for various applications, such as an electronic bulletin board or collaborative document authoring and review. VAX Notes also:

- Saves travel time and money by facilitating problem solving in a conference
- Can be used for internal classes or seminars
- Can provide expertise to groups that lack resources in a given area by referencing experts in the company

17.3.6 VAX Performance Advisor

The VAX Performance Advisor (VPA) is a performance-management and capacity-planning tool. VPA can analyze any VMS standalone system, Local Area VAXcluster, or mixed-interconnect VAXcluster system. To do this, it uses an extensive knowledge base of rules, covering CPU, I/O, memory, clusterwide activities, and hardware-specific thresholds.

In performance management, VPA quickly and thoroughly identifies performance problems, thereby hastening problem resolution. This tool gathers and analyzes VMS system data, identifies specific conditions causing performance degradations, produces conclusions on system performance, and presents detailed evidence to support its conclusions. Also, VPA offers recommendations for problem resolution. This product may be used as a monitoring tool when systems are running smoothly, as well as a troubleshooting tool to identify current and potential problems.

In addition to its expert-system analysis, VPA furnishes capacity-planning features to determine future system performance. For example, this tool helps you determine the effects of changes in workload, configuration, and application on the performance of your systems.

17.3.7 VAX Software Performance Monitor

The VAX Software Performance Monitor (VAX SPM) collects, archives, displays, analyzes, reports, and graphs performance statistics for VAX and VAXcluster systems. This statistical information is useful in system tuning, trend analysis, and workload forecasting. Included in these statistics are resource utilization and load-balance data for single-node, multiple-CPU, and VAXcluster systems.

With VAX SPM, you gain a flexible facility for collecting and archiving performance data. Data may be collected by selecting a variety of parameters. In addition, using a single terminal, you can start and stop data collection for all nodes in a VAXcluster system or from remote nodes in a distributed system. Also, you can archive all the performance data in a single file.

17.3.8 VMS DECwindows Motif

VMS DECwindows Motif, a VMS layered product, features windowing and graphics capabilities based on the industry-standard X Window System developed at the Massachusetts Institute of Technology. VMS DECwindows supplies remote graphics and windowing functionality so that you can run applications on remote nodes and have the application transparently displayed on a local workstation.

A hallmark of the VMS DECwindows architecture is a set of flexible, powerful tools. Software functional components are modular, allowing customized access to the workstation's capabilities.

VMS DECwindows also has a base set of applications, including:

- Bookreader, for viewing the contents of books distributed online
- Calculator, a four-function calculator
- Calendar, a personal time-management system
- Cardfile, a computerized address book
- Clock, an analog or digital date and time display
- DDIF viewers, tools for reading DDIF documents
- Mail, a DECwindows user interface to the VMS Mail Utility
- Notepad, a text editor
- Paint, a bitmap graphics editor
- TPU/EVE, a DECwindows user interface to the VMS TPU/EVE editor

17.3.9 VMS Workstation Software

The VMS Workstation Software (VWS) is a VMS layered product that provides graphics support for the following workstations: VAXstation I, VAXstation II, VAXstation II/GPX, VAXstation II/RC, VAXstation 2000, and VAXstation 2000 Color workstation.

The graphics support provided by VWS includes:

- Windowing support
- VT220 emulation with technical character set
- TEK4014 emulation
- A graphical programming interface
- VWS SIGHT, a menu-driven application that allows you to create illustrations containing text and detailed graphics

17.4 Third-Party VMS CASE Tools

Several popular independent software vendors have adapted their CASE tools to run on VMS systems. These tools are described in the following sections:

- EXCELERATOR
- Software through Pictures
- Statemate
- TEAMWORK

17.4.1 EXCELERATOR

EXCELERATOR, a VMS tool developed by Index Technology Corporation, which supports structured analysis and design, realtime modeling, and data modeling. This tool is integrated with VAX Common Data Dictionary /Repository (VAX CDD/Repository).

17.4.2 Software through Pictures

Interactive Development Environments, Inc. offers a tool called Software through Pictures. This VMS tool consists of a set of graphical editors that support structured analysis and design techniques, realtime modeling, data modeling, code generation, testing, and automatic documentation. Software through Pictures is integrated with the DEC Code Management System (CMS) and DEC Language-Sensitive Editor (LSE).

17.4.3 Statemate

Statemate is a VMS analysis and design tool developed by i-Logix, Inc., for modeling and simulating realtime-aerospace and complex-reactive systems. It enables early error detection through powerful modeling and executable specifications. By executing the model, you see how the system would behave long before implementation. Statemate also generates prototype code and custom documentation directly from the specification. This software has an interface to CMS and is layered on Rdb for VMS.

17.4.4 TEAMWORK

Cadre Technologies, Inc. offers TEAMWORK, a coordinated set of VMS CASE tools that is integrated with LSE, CMS, and DOCUMENT. This product consists of the following components, presented next:

- TEAMWORK/SA, for systems analysis

  TEAMWORK/SA, an environment for systems analysis, offers a graphics-based, integrated tool set that uses structured analysis techniques. It supplies tools for the creation and editing of data flow diagrams, process specifications, and data dictionary entries. TEAMWORK/SA checks for completeness and balancing. It also furnishes facilities for maintaining requirements traceability and collecting project management data.

- TEAMWORK/SD, for systems design

  TEAMWORK/SD, an environment for systems design, supports structured design techniques. Features include creation and editing of structure charts, module specifications, and data dictionary entries. Syntax checking of structure charts is available on a per sheet or whole model basis, at the user's discretion.

- TEAMWORK/Access, for systems integration

  TEAMWORK/Access, an environment for systems integration, is a database access and integration tool. It enables the TEAMWORK package of CASE tools to offer integrated support for the software development life cycle. TEAMWORK/Access opens the TEAMWORK project database to allow integration of the TEAMWORK/SA, TEAMWORK/SD, and TEAMWORK/RT analysis and design tools with documentation, project management, and software development packages.

- TEAMWORK/RT, for realtime modeling

TEAMWORK/RT, an integrated environment for realtime modeling, supports control-flow modeling at the analysis stage of development. The data-flow diagram editor furnishes control flows and control-specification connectors. A state-transition diagram editor and matrix editor are also included; the matrix editor creates objects such as decision tables, state-transition tables, process-activation tables, state/state matrices, and state-event matrices. Control-specification balancing is verified.

18 UNIX CASE Tools in the COHESION Environment

This chapter highlights the key CASE tools that form the Digital COHESION environment for ULTRIX and UNIX systems.

The CASE products in this chapter include:

- DECset tools for ULTRIX Systems, Section 18.1
- PC DECwindows Display Facility, Section 18.2
- Network Application Support (NAS) for ULTRIX, Section 18.3
- FUSE tools for UNIX Systems, Section 18.4

The Source Code Control System (SCCS), the dbx debugger, and the make utility are bundled with the ULTRIX base system. However, they form an integral part of the DECset and FUSE tools, as they provide functionality equivalent to the VMS DECset tools.

Other CASE tools must be ordered separately. For additional sources of information on Digital's CASE offerings, refer to Appendix A. Contact your Digital sales representative to obtain the latest *Software Product Description (SPD)* and *System Support Addendum (SSA)* for each software tool.

UNIX base systems include many of the tools that comprise these CASE tool offerings. CASE environments integrate and expand the functionality of program development tools. FUSE lets you add other tools to FUSE to make your CASE environment fit your organization's needs.

18.1 DECset Tools for ULTRIX Systems

DECset, an integrated group of CASE tools for ULTRIX systems, lets you work in one environment rather than using a series of unrelated tools and files. DECset tools are ideal for facilitating teamwork among groups of programmers. DECset tools are not sold separately. The following sections describe DECset tools:

- SCCS Code Manager
- CMS to SCCS Library File Converter
- make Program Builder
- DEC Test Manager
- dbx Program Debugger
- DEC Language-Sensitive Editor
- DEC Performance and Coverage Analyzer
- DEC Source Code Analyzer

18.1.1 SCCS Code Manager

The DECset SCCS Code Manager provides a DECwindows Motif interface to the SCCS source code control system bundled with the ULTRIX operating system. (SCCS is a standalone base system tool and a component of both the FUSE and DECset CASE tool offerings.) The SCCS system is an online file librarian that tracks software development and maintenance. It stores and organizes groups of files into a project library, tracking changes and monitoring access to the library.

The SCCS system allows you to perform the following tasks:

- Identify the current status of any file, including the name of the person editing it.
- Reconstruct earlier versions of files. For each version, SCCS stores and changes made to produce that version, the name of the person making the changes, and the reasons for the changes.
- Prevent the problems that can occur when two people change a file at the same time without each other's knowledge.
- Maintain multiple branch versions of files. Branched versions can be merged back into the original sequence if desired.
- Protect files from unauthorized modification.

SCCS stores files in a reserved directory. Each generation, history, and change information is kept in this directory.

18.1.2 CMS to SCCS Library File Converter

The CMS to SCCS Library File Converter tool allows users to bring selected elements, generations of elements, groups, and classes from the DEC Code Management System (CMS) over to the SCCS source code control system on ULTRIX systems. The conversion tools provides the following tasks:

- A CMS file selection mechanism
- Methods for moving selected files from VMS to ULTRIX by tape, DECnet, or NFS mounted disk
- SCCS directory creation and popluation on ULTRIX systems

18.1.3 make Program Builder

The DECset `make` Program Builder provides a DECwindows Motif interface to the `make` utility bundled with the ULTRIX operating system. The `make` utility is a productivity tool that builds up-to-date versions of programs. (The `make` utility is a standalone base system tool and a component of both the FUSE and DECset CASE tool offerings.) It is useful for large programming projects in which multiple source files are combined to form a single program. The `make` utility does not, however, address the problem of maintaining more than one version of the source file. After editing a file that is part of a larger program, you can execute the `make` command with a variety of options. The `make` command creates files based on the last time and date the target file was created. Only objects modified after the target's creation date are rebuilt.

The `make` utility allows you to perform the following tasks:

- Combine the instructions for creating a large program in a single file
- Define macros to use within the `make` description file
- Use shell commands
- Create or update libraries
- Include files from other programs

The `make` utility uses time stamps on files. On a distributed system, you need to ensure that date and time are synchronized for all systems in the network.

18.1.4 DEC Test Manager

The DEC Test Manager is a flexible, automated regression-testing system for use during both the development and maintenance phases of the software life cycle. This tool automates the organization, execution, and review of test results and allows several developers to use one set of tests concurrently. With The DEC Test Manager users can perform the following tasks:

- Create software tests which can include descriptions
- Group tests into meaningful combinations
- Execute specific tests, groups of tests, or combinations of groups of tests
- Compare the results on the executed tests with benchmark test results and determine differences
- View test results interactively
- Update benchmarks as needed
- Filter test results to ignore output that is expected to change for each run
- Mask screen output results for tests of X Windows applications to ignore areas of screens

Users have the added flexibility of running DEC Test Manager benchmark tests in playback mode on systems that do not have a DECset license.

The DEC Test Manager is integrated with the DEC Performance and Coverage Analyzer (PCA). SCCS further enhances the testing environment and increases productivity gains.

18.1.5 dbx Program Debugger

The DECset dbx Program Debugger provides a DECwindows Motif interface to the dbx symbolic debugger bundled with the ULTRIX operating system. The DECset dbx debugger is a multilanguage, multiwindow, intuitive interface to both Ultrix and VMS debuggers. Using point and click manipulation you can set breakpoints and variables to display from source code, view and navigate through the calling stack, or use any other debugger feature customarily provided.

The DECset dbx interface allows you to retain debugging information between executions so there is no startup time to run a new version of the application. Using the DECset debugger you can find the problem, then localize it using LSE, SCA, and the Code Manager. You can fix the problem using the Code Manager, LSE, and the compiler, and then build the system using the Code Manager and Program Builder.

18.1.6 DEC Language-Sensitive Editor

The DEC Language-Sensitive Editor (LSE) is a multilanguage, multimodule, multiwindow, screen-oriented text editor for program design, development, and maintenance. The editor is language sensitive in that it provides templates of major constructs in supported ULTRIX languages, including DEC Ada, C languages, and DEC Fortran.

The template fosters rapid and accurate code development. You can write code and then edit, compile, review, and correct compilation errors in the same session. The editor is also extendable, allowing customization to fit your programming needs.

LSE is tightly integrated with the programming development environment. It is invoked on the command line and works with supported Digital languages and other CASE tools. On ULTRIX, LSE works with the dbx debugger, SCCS, and SCA in an interactive program development environment. LSE facilitates the edit-compile-debug-analyze portion of the program development cycle.

DECset includes enhancements to LSE, SCA, and Digital language compilers that support detailed or module design. These capabilities permit you to express a design using pseudocode, capture the design information, and supply various reports that list design cross-references, call trees, and the use of data structures.

18.1.7 DEC Performance and Coverage Analyzer

The DEC Performance and Coverage Analyzer (PCA) is a multilanguage, source-profiling tool designed to help you analyze and improve the execution behavior of application programs. It also offers test-coverage analysis by measuring the parts of a user program that are executed or not executed by a given set of test data.

PCA consists of two parts: the Collector and the Analyzer. The Collector gathers performance or test-coverage data as a program runs and stores it in a data file. The Analyzer then processes the collected data and presents it either in tables or histograms.

18.1.8 DEC Source Code Analyzer

The DEC Source Code Analyzer (SCA) is an interactive, multilanguage, multimodule, source code cross-reference and static-analysis tool. Designed to clarify the complexities of entire software systems, this tool benefits both the implementation and maintenance phases of application development.

SCA provides navigation capabilities to assist users in locating and viewing source code components. SCA stores compiler-generated information about a set of source files in an SCA library, then allows users to perform queries about their source code in the following ways:

- Using a name browser to locate all items that match a search string
- Specifying a cross-reference query to find how and where program symbols are used
- Specifying a data structure query to graphically display data types or to find symbols in the source code

SCA is tightly integrated with LSE and may be viewed as a logical extension of the editor's error prevention, detection, and correction facilities.

18.2 PC DECwindows Display Facility

The PC DECwindows Display Facility is an MS–DOS application that implements an X server using the industry-standard X Window System, Version 11 (X11) protocol. DECset components executing on a remote ULTRIX system with DECnet may be displayed on, and receive input from, the personal computer. The PC DECwindows Display Facility is supported only on the Intel 80286-, 80386-, and 80486-based machines. Refer to the *PATHWORKS for DOS Software Products Description (SPD)* for full hardware and software requirements.

18.3 Network Application Support (NAS) for ULTRIX

Digital's Network Application Support (NAS) products provide solutions to the problems created by the complexity of today's computing environment.

NAS is a comprehensive set of standards-based software that consists of well-defined programming interfaces, toolkits, and products to help developers build applications that are integrated and easily ported across a distributed, multivendor environment. Two major components comprise the NAS solution, as follows:

- Distribution of resources across the enterprise, network, and multivendor systems
- Integration of components within the information system and with other information systems.

NAS helps to insulate applications from changes, including changes in hardware platforms and operating systems, and the need for different network protocols. NAS does this by providing the infrastructure from which applications can draw common services, masking the differences among the underlying platforms. The NAS infrastructure, along with a set of tools to help develop and maintain applications that are built upon it, forms a complete, platform-independent, virtual system.

NAS for ULTRIX component products are bundled into complete sets targeted for specific environments, as follows:

- NAS 200 for ULTRIX: Integration at the Desktop
- NAS 300 for ULTRIX: Distributed Client/Server Computing

18.3.1 NAS 200 for ULTRIX: Integration at the Desktop

NAS 200 for ULTRIX provides a complete set of networking and distributed computing capabilities for basic print, file, and data-sharing services for PCs, Macintosh, and workstations. It is targeted for environments where applications run primarily on the desktop and where users of these applications need to share files, printers, and data.

The following software and product components are included with NAS 200 for ULTRIX:

- PATHWORKS for ULTRIX
- Remote System Manager (RSM) Client
- DECnet/OSI for ULTRIX

NAS 200 for ULTRIX supports key standards such as OSI, LAN Manager, TCP/IP, and NFS.

18.3.2 NAS 300 for ULTRIX: Distributed Client/Server Computing

NAS 300 for ULTRIX integrated software products provide a complete set of runtime services for client/server, distributed, or general host-based applications. They provide integration capabilities including distributed user-interface facilities, object-oriented application linking and control, support for creating multimedia documents and management of multimedia data, and desktop integration.

The following software and product components are included with NAS 300 for ULTRIX:

- PATHWORKS for ULTRIX
- Remote System Manager (RSM) Client

- DECnet/OSI for ULTRIX
- DEC ACA
- DECmessageQ for ULTRIX on RISC systems

NAS 300 for ULTRIX supports key standards such as OSI, TCP/IP, NFS, LAN Manager, and OMG/ORB.

18.4 FUSE Tools for UNIX Systems

The Friendly Unified Software Environment (FUSE) integrated programming environment on Digital's ULTRIX ans Sun's SunOS operating systems. FUSE provides an intuitive style of software development and maintenance commensurate with the portability requirements of the UNIX community.

FUSE is an integrated, graphically oriented environment for Digital's ULTRIX and Sun's SunOS operating systems. FUSE commands and utilities are based on OSF/Motif user-interface standards. It features dynamic, mouse-sensitive, graphical tools that greatly simplify problem identification and program navigation. It also provides tool-to-tool interoperation to enhance the power and efficiency of several programming development, analysis, and maintenance tasks.

This section describes the following FUSE tools:

- FUSE Control Panel
- FUSE Builder
- FUSE Call Graph Browser
- FUSE Code Manager
- FUSE Cross-Referencer
- FUSE Debugger
- FUSE Editors
- FUSE Online Help
- FUSE Profiler
- FUSE EnCASE Kit
- DEC FUSE Support for DEC C++ Kit
- DEC FUSE C++ Support Kit

Many FUSE tools are based on standard UNIX utilities, such as make, GNUmake, SCCS, RCS, prof and dbx, that are included with the operating system software. Programmers who are experienced in developing software with utilities provided by UNIX based operating systems have an advantage when working on programs in FUSE. For programmers who are less familiar with such operating systems, FUSE provides a consistent user interface and online help for each tool.

18.4.1 FUSE Control Panel

The FUSE Control Panel is the central point for managing FUSE tools and obtaining FUSE system information. Its display shows what tools are up and running and what files are associated with each tool.

The FUSE Control Panel is also used to create and manage tool groups, which enable tool interoperation.

18.4.2 FUSE Builder

The FUSE Builder allows programmers to initiate and control program builds. It is based on the UNIX make utility and uses conventional makefiles. No modifications are required to existing makefiles for use with the FUSE Builder.

Use the Builder to complete the following tasks:

- Create simple makefiles
- Build (compile) programs
- Analyze information in makefiles
- Review transcripts of builds
- Track and correct build errors
- Execute distributed builds

Use the Builder with other FUSE tools to track and correct build and runtime errors.

18.4.3 FUSE Call Graph Browser

The FUSE Call Graph Browser is used to analyze a program's call structure. The Call Graph Browser displays the relationships among functions in source files in a program. Use the Call Graph Browser to complete the following tasks:

- Analyze the call relationships in a program using the call graph
- Review information on calls, functions, source files, and directories

- Track program execution through the call graph

Usually, the Call Graph Browser is used with the FUSE Editor and Debugger to analyze a program's call structure while also viewing the associated code. When the Debugger is executing a program, the Call Graph Browser can visually track the execution. Breakpoints can be set to allow detailed analysis of the call structure within a section of the program; review of associated source code; and editing of source code, if necessary.

18.4.4 FUSE Code Manager

The FUSE Code Manager creates and maintains Source Code Control System (SCCS) and Revision Control System (RCS) libraries. Use the Code Manager to complete the following tasks:

- Navigate quickly through a library using the library graph
- Manage files within a library, including creation, check in, check out, compare, and view
- Review the transcript of a code management session
- Create a code management library
- Review file information

Use the Code Manager with the Debugger and editors to access and update a series of files to compress the file editing and storing process.

18.4.5 FUSE Cross-Referencer

The FUSE Cross-Referencer searches for and identifies the location of references within a directory, program, or source file. Use the Cross-Referencer for the following tasks:

- Request location information about specific entities in the source code
- Examine specific entities in the source code

You can use the Cross-Referencer with an editor to locate items and edit the associated source files. Either a regular expression or a literal can be used to locate references, assignments, functions, calls, and declarations in the directory, program, or source file.

18.4.6 FUSE Debugger

The FUSE Debugger is based on the dbg utility and provides the following extensive program debugging capabilities:

- Source file editing
- Quick command execution using buttons, dialog boxes, or a command line interface
- Visual event setting
- Extensive debugging and program monitoring
- Debugging environment customization

Use the Debugger with other FUSE tools for the following tasks:

- Debug a program and view associated source code
- Trace program execution
- Correct source files, rebuild the program, and test again

Source files associated with the executable file being debugged are loaded into the Debugger along with the executable file. You can edit source files while debugging a program, using editing commands in the Debugger's source text area.

The Debugger provides six windows to monitor the program and display debugging information. Isolating information in separate windows displays the information more efficiently. There is a view window for each of the following debugging and program activities:

- **Registers.** The contents of general and floating point registers are displayed and updated at events. (Not available with DEC FUSE for Sun.)
- **Stack.** The contents of the stack are displayed and updated at events.
- **Status.** All events with the number, type, and description are listed and updated as they are added or removed.
- **Watch.** Variables and locations can be monitored by value, at breakpoints, or on execution of the source line.
- **Assembly.** The assembly language instructions before and after the currently executing line are displayed at each breakpoint.
- **Input/Output.** All output from the executing program is displayed. You enter interactive program input in this window.

The debugging environment can be customized to maximize debugging efficiency. Customization can occur from within the Debugger through a Debugger initialization file that executes automatically each time the Debugger starts.

18.4.7 FUSE Editors

FUSE provides three editors: the FUSE Editor, vi, and EMACS. All editors associated with a tool group and can respond to directives from other FUSE tools in the same group. For example, if you click on an entry in the Cross-Referencer's display area, the source file and line associated with that entry is displayed in the editor.

FUSE contains an extensive set of keyboard bindings for editing and generating C, Pascal, and FORTRAN language statements. The keyboard bindings for the editing commands are the standard Motif key bindings.

The File, Edit, and Buffer menus are available in many FUSE tools. Often, the File and Edit menus are used to manipulate and edit a tool's transcript.

18.4.8 FUSE Online Help

FUSE uses the standard OSF/Motif help tool to provide extensive online help about how to use each tool and its dialog boxes.

18.4.9 FUSE Profiler

The FUSE Profiler is the main tool for performance testing. The Profiler generates and graphically displays the following runtime statistics:

- CPU cycles used by each function
- The number of calls made by each function
- The execution time required by each function
- A list of functions not executed

The Profiler in DEC FUSE for ULTRIX also generates and displays machine code runtime statistics.

18.4.10 FUSE EnCASE Kit

The FUSE EnCASE kit provides support for integrating non-FUSE tools into FUSE. Once a tool is integrated, you can start it from the FUSE Configuration menus and monitor it using the Message Monitor (FUSE tool available only with the EnCASE Kit). Integrated tools use the FUSE message server for interprocess communication and can use FUSE tools to complete tasks.

FUSE EnCASE includes the following facilities for integrating tools:

- **Tool Integration Language (TIL) and compiler.** TIL defines how tools are named, the tool's location, startup specifications, and message list.
- **Call interface functions and commands.** Callable C functions and script commands provide the messaging interface between a nonFUSE tool and FUSE.
- **Message Monitor.** The Message Monitor displays messages sent between FUSE tools during a FUSE session.

18.4.11 DEC FUSE C++ Support Kit

The optional DEC FUSE C++ Support kit provides you with a new tool and allows you to use other FUSE tools with programs that were built using AT&T cfront compatible compilers and Glockenspiel programs. You can use the tools as follows:

- The C++ Class Browser allows you to graphically view the class hierarchy of your applications. You can view classes, members, and inheritance relationships.
- The Debugger supports all C++ debugging requirements, including:
  - C++ symbol name conversion to user names and vice versa (mangling and demangling), for C++ object, function, and class member names
  - Expression evaluation of C++ expressions and operators
  - Resolution of C++ overloaded functions and operators
  - Scope resolution using the scope resolution operator
  - Display of active stack information, including local symbols within function blocks
  - Instruction display which includes demangling function names
  - Display of data objects at all scopes
  - Access to C++ name and structure information using dbg commands, including the `info`, `whatis`, and `whereis` commands.
- The Call Graph Browser supports C++ program structure in its call graph.
- The Cross-Referencer supports querying for C++ programming entities.
- The Profiler supports C++ user names.

18.4.12 DEC FUSE Support for DEC C++ Kit

The DEC FUSE Support for DEC C++ Kit (on ULTRIX only) provides all the C++ cfront support in the DEC FUSE C++ kit as well as support for DEC C++. This kit enables DEC C++ programmers to use the FUSE integrated development environment with DEC C++, Digital's native, optimizing C++ compiler.

The DEC FUSE Support for C++ kit is used in conjunction with DEC FUSE for ULTRIX and DEC C++ for ULTRIX. DEC C++ for ULTRIX has a command-line interface and includes the DEC C and DEC C++ compilers, the DECladebug debugger, and class libraries. Using these products together provides a powerful, window-based DEC C++ programming environment.

DEC FUSE Support for DEC C++ contains the following tools:

- The C++ Class Browser allows you to graphically view the class hierarchy of your applications. You can view classes, members, and inheritance relationships.
- FUSE DECladebug provides a window-based interface, with full DEC C++ debugging capabilities, to the DECladebug debugger.
- An incremental linker (`ild`) provides a faster edit-to-execute cycle by processing only changed modules.
- The Call Graph Browser supports the DEC C++ program structure in its call graph.
- The Cross-Referencer supports querying for DEC C++ entities.
- The Profiler supports DEC C++ names.

19 DECfactory Products in The Realtime Environment

DECfactory products and services provide comprehensive factory floor solutions for targeted industries in all geographies. DECfactory solution products offer a full range of DECfactory services and Network Application Support (NAS) information technology components. From single product needs to a systems integration project, DECfactory services address all the associated people, business, and technological issues. DECfactory products, as part of Digital's NAS Environment for Manufacturing, provide manufacturers with multiplatform, standards-based technology in the areas of integration and information. All DECfactory products run on both VAX and MIPS-based ULTRIX platforms. DECmessageQ can be used to integrate additional platforms.

The products and services in this chapter are arranged as follows:

- Digital's DECfactory Services, Section 19.1
- BASEstar to Integrate Manufacturing Applications and Devices, Section 19.2
- DECmessageQ for Application Communications, Section 19.3
- DECómni to Simplify Device Integration, Section 19.4
- DECosap to Link Siemens Devices, Section 19.5
- @aGlance to Access Data, Section 19.6

When companies integrate manufacturing operations, they typically are faced with the task of facilitating communication among heterogeneous devices and computing systems. The factory is usually a mixture of intelligent floor devices supplied by many vendors with little communication between different levels of plant management. A common information-management system is needed to increase productivity and distribute device control beyond the shop floor. Networking this heterogeneous environment in a piecemeal fashion can be expensive, difficult to maintain, and therefore, generally short-lived.

To ease the task of manufacturing integration and to prepare for future expansion and technological advancements, the International Standards Organization (ISO) developed the Open Systems Interconnection (OSI) Manufacturing Message Specification (MMS). MMS specifies a standard method of communication between host computers or cell controllers and intelligent floor devices such as programmable logic controllers (PLCs), numeric controllers (CNCs), distributed control systems (DCSs), and robot controllers. The MMS standard specifies a device communications protocol and a set of services for communication between applications and devices.

For manufacturers with devices that do not use the MMS standard, DECfactory products also support defacto standard software.

DECfactory products include standards-based integration and communication products. DECmessageQ facilitates communication between applications while the other DECfactory products facilitate communication between devices and applications. All DECfactory products work together in a distributed, heterogeneous environment to provide manufacturers with standards-based technology to help achieve manufacturing excellence and preserve current equipment and software investments.

For additional sources of information on the DECfactory environment, refer to Appendix A.

19.1 Digital's DECfactory Services

DECfactory Services consist of a worldwide portfolio of planning, implementation, and support services. Through these services, Digital and its customers can establish and maintain long-term relationships to address business needs and integrate systems.

DECfactory Services cover two broad areas; Aerospace/Electronics and Automotive/General Discrete manufacturing practices. These services include education, change management, human systems consulting, re-engineering of business and manufacturing processes, and information technology.

19.2 BASEstar to Integrate Manufacturing Applications and Devices

BASEstar integrates manufacturing applications with a variety of industrial control devices including PLCs, robots, numerical controllers, gauges, and bar-code readers. The core of BASEstar's integration ability is its memory-resident, current value data manager, which provides event-driven, distributed, global naming capabilities for storing and accessing factory data.

BASEstar facilitates the integration of manufacturing applications with plant equipment, accelerates development of integrated manufacturing systems, and provides a consistent architecture for developing manufacturing applications. The interface to BASEstar contains a menu interface, a Command Line Interface (CLI), and an Application Programming Interface (API). BASEstar has features to perform the following tasks:

- Application integration
- Device integration
- Configuration management

The application integration facilities include the ability to collect, manage, and distribute plant data, automatically notify applications of critical changes in plant information, and synchronize execution of manufacturing applications. Distributed capabilities allow globally-defined objects to be used by applications running on different nodes in a BASEstar network.

BASEstar device integration facilities give generic device access and control for plant equipment through a protocol- and device-independent interface. Device connection management depends on equipment communications (facilitated through BASEstar device access software (DAS)) which allows data obtained through standard mechanisms to be made available through the BASEstar network. Device connection management facilities can also start and stop devices, upload and download to programmable devices, and perform other standard functions offered by industrial control devices.

BASEstar configuration management facilities let you store and track file development, control access to files, and transfer files to and from devices. By providing a history of file changes, a file can be tracked from its development stage through testing and production to archive. By controlling file distribution, an outdated control program can be kept from running in production.

19.2.1 Application Integration

BASEstar data comes from a variety of sources including plant devices, work cell applications (area or plant), and user input. BASEstar data is referenced by name, regardless of the source or complexity of the data. Data elements are called logical points and can reference a single data item or complex data structures. Because BASEstar data is referenced by name, applications are independent of data sources and do not require alteration when data sources change.

BASEstar collects and distributes manufacturing data to integrated applications. You can use BASEstar data management facilities to change point value definitions, perform arithmetic or local operations on point values, or receive notification of point value changes.

The integration of BASEstar with DECmessageQ allows BASEstar applications to receive data change notification through DECmessageQ queues. Communication in a BASEstar application is often event-driven, requiring both synchronous and asynchronous messages. DECmessageQ allows for interprocess communication in a distributed, heterogeneous environment between independent tasks. Messaging in BASEstar supports point-to-point messaging, messaging over a circuit between two ports, or messaging to a circuit cluster port that forwards the message to multiple destinations.

BASEstar application control allows you to initiate startup or shutdown procedures from any BASEstar system. You can control the entire system or specific resources within the system to synchronize access. Security features let you limit access to these features.

The BASEstar kit contains two software development tools to aid in application development. The Value Notification Utility is a testing tool to track logical point value changes. The Language Sensitive Editor (LSE) provides expandable tokens for BASEstar callable services, DAS support routines, and programming interface (API) constants.

19.2.2 Device Integration

Device integration software allows applications to interact with plant devices without knowing the device characteristics such as location, protocol, or data formats. Data collection can be event-driven or done through polling.

Device access software (DAS) consists of code that emulates the specific device and a network interface to distribute the data. BASEstar DAS modules are available for many leading industrial control devices. The following are three examples of DAS modules to communicate between devices.

- **RS-232 DAS.** Enables communication with devices having an RS-232 serial port
- **DECnet and TCP/IP DAS.** Enables communication with applications through DECnet and TCP/IP networking software running on an MS–DOS personal computer or on a UNIX system with BASEstar device connection management software
- **DECdevice DAS.** Emulates the memory of a simple manufacturing device for testing applications using BASEstar device connection management functionality

Using the BASEstar library system, programs can be downloaded or uploaded from the library to a device or vice versa. The version of a file stored on a programmable device can be compared with library files. The BASEstar library supports up to 20 concurrent users and a maximum of 2500 files.

19.2.3 Configuration Management

BASEstar configuration management facilities allow you to use named objects such as plant devices, users, and point values to manage plants tasks. Named objects can be organized and referenced as collections; named objects can have either local or global scope to control use of the named objects; and access to named objects can be controlled through BASEstar security facilities.

Event logging provides a way to centrally record application, system, and network events, such as a point value change. Events can be logged from an application or through the API. Text from event messages can be stored and used later for debugging purposes.

19.2.4 BASEstar CIMfast for Application Development

BASEstar CIMfast reduces development time and application complexity and aids in rapid prototyping through a customizable, high-level, event-driven language. The CIMfast Event Language (CEL) can be integrated into existing BASEstar code and is supported by the Language Sensitive Editor (LSE). CEL allows the user to define events and specify an action when the action occurs. For example, the production floor foreman may decide to stop a rolling machine if the product thickness exceeds 6 millimeters. In this case, the event is a thickness change over a specified acceptance level and the response is to halt the rolling machine so that corrections can be introduced without further stock spoilage.

19.3 DECmessageQ for Application Communications

DECmessageQ is a NAS software integration product used in the electronics and semiconductor industries to develop distributed applications running in multivendor environments. DECmessageQ provides recoverable interprocess messaging services between two or more cooperating processes or applications. The message is stored on a disk or memory-based file until it is needed by the receiving application. If there are difficulties at the receiving end, such as CPU, application, or network problems, the message is saved until it can be sent.

DECmessageQ offers a common application programming interface (API) across different operating systems and networks, simplifying the integration of new and existing applications. With DECmessageQ, program modules can be developed or redeployed anywhere in the DECmessageQ network configuration, whether standalone, in a VAXcluster configuration, in a local area network (LAN), or at remote sites. The queued message bus structure means that applications do not have to establish links with each of their application partners.

DECmessageQ uses a client/server architecture, with one system acting as the server to exchange messages among multiple client implementations. Refer to Chapter 20 for more information on the platforms supported by DECmessageQ.

19.4 DEComni to Simplify Device Integration

Digital's OSI Manufacturing Network Interconnect (DEComni), implements the Manufacturing Message Specification (MMS), an international standard communications protocol (ISO/IEC 9506–1 and ISO/IEC 9506–2). When combined with prerequisite hardware and software, DEComni interoperates with other MMS-compliant systems and devices.

DEComni consists of a library of callable routines and runtime services which allow host applications to communicate with multivendor, intelligent manufacturing devices such as robot controllers, programmable logic controllers, numerically controlled machine tools, and distributed process control systems. DEComni runtime library routines can be called from an application written using any Digital language.

DEComni includes utility programs for creating and managing definitions and data structures. The application programming interface (API) is an implementation of MMS and facilitates use of utilities, libraries, and services in a seamless, distributed, network application. Service classes support are in the following categories:

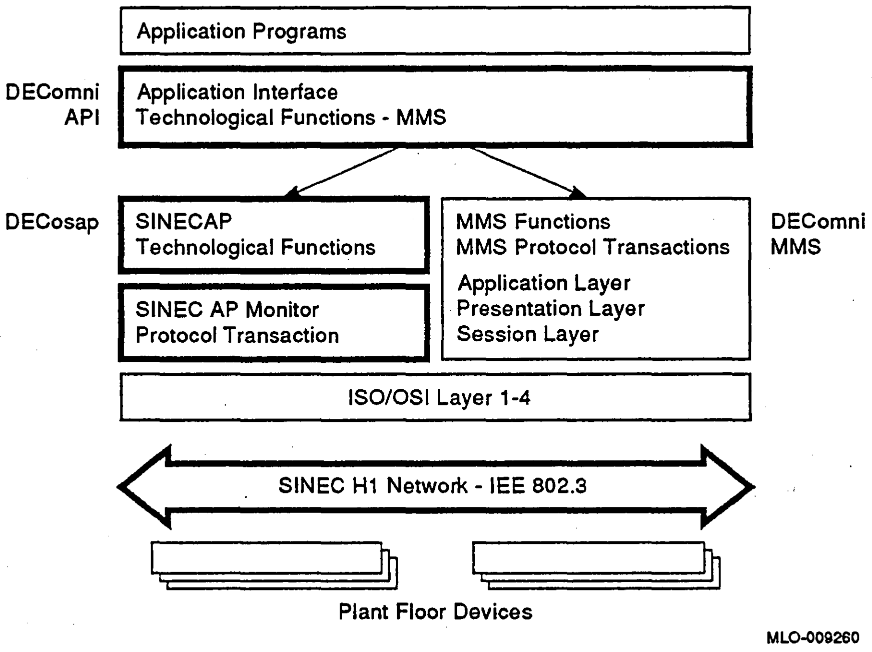
- Environment
- Virtual Manufacturing Device (VMD)
- Domains
- Program Invocations
- Variables
- Files

The DEComni Object Definition Facility (ODF) is used to define local and remote objects and attributes of a Virtual Manufacturing Device (VMD). ODF maps MMS variable types to and from VMS data type definitions, but is not yet supported on ULTRIX.

DEComni can be used from a DECwindows environment or a character cell terminal.

Figure 19–1 shows how DEComni and DECosap work together to form layers of industry standard protocols between the application interface and plant floor devices.

Figure 19–1 Integration of DEComni and DECosap



19.5 DECosap to Link Siemens Devices

DECosap is a network communications product that allows applications to communicate with shop floor devices that use the proprietary Siemens SINEC Application Protocol (AP) and SINEC H1 communication protocols. Digital, in cooperation with Siemens, developed DECosap to connect VAX VMS and MIPS-based ULTRIX computers with Siemens communications processors. As defined, these protocols provide communications functionality for plant automation over 802.3/Ethernet local area networks (LANs). This attention to standards results in higher performance and lower operating costs.

SINEC AP and SINEC H1 are defined for high-speed communications covering all automation levels. They are application-to-application protocols developed on top of the Transport Layer of the Open System Interconnection (OSI) reference model. Although primarily oriented to the manufacturing environment, SINEC AP and SINEC H1 address generic applications and provide a common communication method between different Siemens automation devices, such as Programmable Logic Controllers (PLCs), numerical controllers, robots, and personal computers.

DECosap defines its Application Programming Interface (API) through DEComni, Digital's device interconnect API for manufacturing devices that is modeled on the ISO/IEC 9506 Manufacturing Message Specification (MMS). The result is a migration path for applications to technologies based on international standards. The following list highlights DECosap support for both VMS and ULTRIX systems.

- **DECosap fully supports SINEC AP services.** These services include environmental management, virtual manufacturing device support, variable access, serial transfer, program invocation, domain services, and message exchanging.
- **DECosap fully supports SINEC H1 services.** These services include environmental management, variable access, message exchanging, and device management.
- **DECosap functionality relies on DECnet/OSI.** This feature ensures a smooth migration to an intended full OSI implementation by Siemens and protects investments in installed applications.
- **DECosap supports all supported VAX and MIPS-based processors.** This results in an integrated application environment in which information can be transferred easily from plant floor devices throughout an enterprise-wide network.

- **DEComni services complement DECosap functionality.** Tools that support the creation and managment of plant floor device applications are provided in conjunction with DEComni. Omni Directory Services support configuration and management of application addressing information.

Software Requirements for DECosap

Both DECosap for VMS and DECosap for ULTRIX require DECnet/OSI. In addition, they each DEComni for their respective operating systems (DEComni for VMS or DEComni for RISC ULTRIX). DECosap for VMS also requires the Rdb/VMS Runtime Option.

19.6 @aGlance to Access Data

@aGlance integrates popular desktop applications with control systems across multivendor platforms. Manufacturers can access plant data easily from a variety of spreadsheet, statistical analysis, or graphical interface applications from any control system in the plant or from a remote location. @aGlance is used primarily by plant personnel to prepare managerial overviews of plant operations, analyze control loops, or economic analyses.

@aGlance is both an application developer's toolkit and runtime facility which enables client and server developers to produce interoperable applications. @aGlance is layered on Digital's Application Control Architecture (ACA) Services, which provides server registration, method invocation, and data exchange functions across a variety of hardware platforms in a networked environment.

20 DECmessageQ in the Realtime Environment

Digital's DECmessageQ product is a solution to application integration across a range of environments. As the first of a family of message queuing products under Digital's Network Application Support (NAS) umbrella, DECmessageQ is a key component in the NAS strategy.

This chapter describes the following:

- DECmessageQ Overview, Section 20.1
- DECmessageQ for UNIX, Section 20.2
- DECmessageQ for VMS, Section 20.3
- DECmessageQ Hardware and Software Requirements, Section 20.4

The original name for DECmessageQ was PAMS, which stood for Process Activation and Message Support. DECmessageQ greatly expands the capability of PAMS, but the DECmessageQ API has preserved the original product acronym in the name of each callable service to protect customer investment in application development.

DECmessageQ meets the demands for a loosely-coupled, flexible mechanism for integrating applications in a distributed, multivendor environment. For additional sources of information on the DECmessageQ product, refer to Appendix A.

20.1 DECmessageQ Overview

Distributed computing is an important trend which has evolved over the past several decades. Distributed applications allow for increased accuracy and timeliness of data collection and enable greater use of data by many users. But distributed applications require access to the data, even if that data resides in a heterogeneous computing environment. DECmessageQ contributes to distributed computing in the following way:

- **A common application programming interface (API).** The common API across different operating systems and networks, simplifies the integration of new and existing applications. The queued message bus structure of DECmessageQ means that applications do not have to establish unique links with each of their application partners.
- **Two user interface (UI) driver routines.** DECmessageQ provides for both MOTIF and character cell terminals. Application actions are performed by calling an action routine and passing it the appropriate data structures. Therefore, a single set of user interface action routines serves both MOTIF and character cell interfaces. All DECmessageQ MOTIF user interface components share a common look and feel regardless of the runtime environment.
- **A client-server architecture.** The client-server model of message exchange distributes application components such that they can run in a multivendor environment, thus maximizing system efficiency while distributing access to a central database. The client-server model divides processing application among server processes (that manage a single resource) and client processes (that issue requests to the servers). By communicating through messages, clients and servers efficiently send and receive data. The client-server model lends itself to multiprocessing environments if applications can be decomposed into smaller tasks.

  If tasks are distributed in a homogeneous hardware and software environment, communication between cooperating tasks is relatively simple. However, it is common to find distributed systems that make use of different computing platforms or operating systems. Often, these systems are connected by more than one network service. For example, in a manufacturing facility process monitoring and control may be VAX-based and report distribution may be done using an IBM transaction system to printers on PCs. In this case, data may be moved around the company using a combination of network services. Each network service has its own communication mechanism, which must be understood on both ends of the data conversation.
- **A software message bus for efficient network communications.** The queued message bus structure of DECmessageQ means applications do not have to establish unique links with each of their application partners in heterogeneous environments and across multivendor hardware and operating system platforms. The benefits of using a single communication mechanism in such a situation can be seen in reduced development and maintenance costs, improved service, and investment protection.

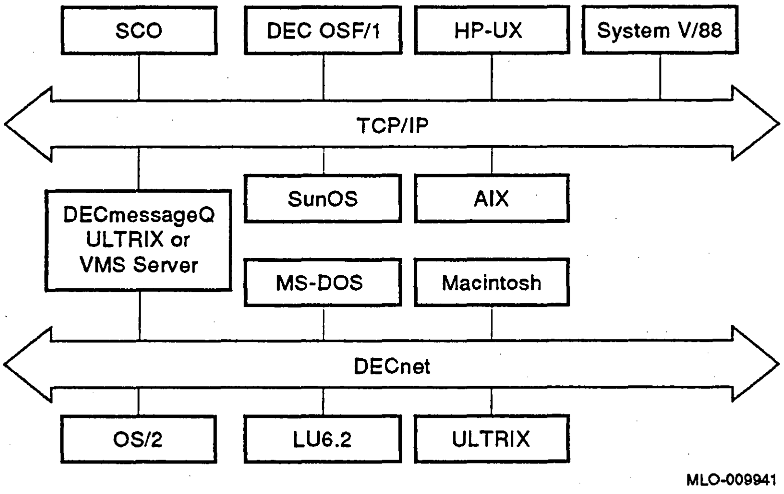
Development and maintenance costs are reduced when application designers are freed from customizing networking tasks. DECmessageQ offers a common application programming interface (API) across different operating systems and networks, simplifying the integration of new and existing applications. With DECmessageQ, program modules can be developed or re-deployed anywhere in the DECmessageQ network configuration, whether in a standalone mode, a VAXcluster, a LAN, or at remote sites. Development of new and maintenance of existing applications is reduced because the need for design compromises is reduced. DECmessageQ provides the following benefits in a distributed multivendor environment:

- Reduced software development overhead when implementing a client-server architecture
- Shared data among applications running on different computers
- Centralized reporting from remote sources within the company
- Distributed data stored on IBM host systems
- Distributed applications on PCs

• **Interoperability between Digital, nonDigital, and IBM.** DECmessageQ connectivity to IBM platforms is through the LU6.2 services. DECmessageQ network connectivity options include DECnet, SNA, and TCP/IP. Since DECmessageQ uses a client/server architecture, at least one system is needed to act as a message server to exchange messages among client implementations on any of the other systems supported by DECmessageQ. between heterogeneous environments across multivendor

Figure 20–1 shows how DECmessageQ communicates with client implementations in a distributed environment.

Figure 20–1 DECmessageQ Communication Paths



DECmessageQ is one of Digital's Open Systems offerings that supports multivendor platforms and industry standards, which leads to less proprietary systems that can meet a variety of demands. DECmessageQ provides reliable message exchange in a multivendor environment using a single portable application programming interface. The benefits of using DECmessageQ as the basis for application design can be summarized as follows:

- **Productivity.** DECmessageQ provides a standardized approach to interprocess messaging that speeds development and insulates applications from changes in network and operating system software.
- **Simplicity.** The DECmessageQ Message Queuing Bus offers a single connection point for each application to communicate with others.
- **Reliability.** Advanced DECmessageQ messaging features offer a simple and reliable queuing mechanism for message delivery.
- **Portability.** The DECmessageQ Application Programming Interface (API) is a common programming interface for all environments. Applications are written once, then ported to other systems.
- **Interoperability.** DECmessageQ software integrates applications running in a multivendor environment, including the following products:

  VMS (server)
  VMS LU6.2 services for IBM CICS and IMS (client)
  ULTRIX (server)

Macintosh (client)
MS–DOS (client)
OS/2 (client)
HP-UX (server)
Sun (server)
SYSTEM V/88 (server)

DECmessageQ support of each of these platforms depends on the specifics of the operating system. Please check the individual *Software Product Description (SPD)* for product-specific information, requirements, and restrictions.

20.1.1 DECmessageQ Services

Digital offers a full range of services for every phase of a DECmessageQ integration project. DECmessageQ services include Digital standard services such as Loan of Product, Technical Training, Integration Consulting, Customer Integration Service, DECstart Startup Service, and Customer Services support.

In addition to these standard services, the DECmessageQ Expertise Center provides a range of custom services to assist customers in planning, designing, and implementing an even broader range of application communications capabilities.

- Experienced DECmessageQ consultants are available to advise customers in the planning and design of integrated application systems.
- As part of its Custom Integration service, the DECmessageQ Expertise Center has developed a software toolkit called the DECmessageQ Q-Adapter (DmQA). The DmQA software toolkit is used to expedite the integration of applications running on systems not currently supported by the standard DECmessageQ products.

With DECmessageQ, DmQA, and expert consulting, Digital offers shortened development time, reduces the risks associated with developing complex applications, and provides competitive off-the-shelf solutions.

20.1.2 DECmessageQ Features

The DECmessageQ communication implementation is designed for ease-of-use, expandability, and efficiency. DECmessageQ on VMS and UNIX all include the following features:

- Message sizes up to 32,000 bytes
- Priority queuing of messages
- Selective reception of messages by queue number (FIFO), priority, or by direct item access methods

- Shared input queues using Multi-Reader Queues (MRQ)
- Remote message delivery
- Inter- and intra-CPU message delivery
- Set of message delivery options
- A maximum of 999 queues per DECmessageQ Group
- A maximum of 32,000 DECmessageQ Groups
- Integration with timers
- Support for local languages
- Utilities for monitoring the network configuration and flow of messages
- Portable call interface
- Connectivity to DECmessageQ implementations on other systems

20.1.3 DECmessageQ Queues

Messaging is the deliberate transmission of data between two or more cooperating applications. These cooperating applications can run on the same processor or on several processors in a single multiprocessing CPU. The process can also run on several CPUs in a VAXcluster, across a local area network (LAN), or across a wide area network (WAN).

A messaging system provides communication through a standard message-passing mechanism. Communication can flow in either direction and it can be synchronous or asynchronous. That is, the sender process can wait for a reply from the receiver process or it can continue as soon as the message is queued. By queuing messages, processes can execute independently and are not blocked during message transmission.

Messages can be placed on a queue in a FIFO or priority-based manner. Messages can be extracted from a queue either sequentially or by some selection criteria (priority, sender, message, class, etc.)

The message queue is a memory storage location (queue number) for DECmessageQ messages. An application can attach to more than one queue and can read from multiple queues. Once a message is read, it is removed from the message queue.

DECmessageQ provides three types of queues. Any process can insert a message into any queue. These queues are accessed directly by DECmessageQ procedures, which are called by user-written applications. The three types of DECmessageQ queues are:

- Primary Queue (PQ) - Each process that attaches to the message queuing bus is assigned a Primary Queue. This queue is used to receive messages from processes using DECmessageQ.
- Secondary Queue (SQ) - Any process may attach to one or more secondary queues. These queues can also be used to receive messages. The order in which queues are scanned for messages is defined by the DECmessageQ selection rules.
- Multi-Reader Queue (MRQ) - A Multi-Reader Queue is a single queue that can be shared by up to 40 simultaneous readers.

20.1.4 Message Selection

DECmessageQ software allows application developers to create message selection filters for sending and receiving messages. A message can be selected by queue type, message attribute, message source, or by a combination of selection criteria.

Message attributes include priority, type, class, and sender. These attributes allow application developers to assign classifications to messages, then select criteria to filter which messages are delivered. Support for message selection attributes may vary depending on the operating system.

Selection by message priority allows programs to read messages according to their relative importance. Each queued message has a priority of either 0 or 1. Applications can then request delivery of one priority before delivery of the other. Using priority queuing, applications can read more-critical messages before less-critical ones.

20.1.5 Message Recovery

When an application sends a message across a communications network, the final receipt of the message can be interrupted by various failure conditions, including a task abort or system crash. DECmessageQ for VMS provides the following set of options for recoverable and nonrecoverable message delivery:

- **Datagram.** A nonrecoverable attempt is made to deliver a message. If the message cannot be delivered to a target then, an error is logged.
- **Wait for enqueue.** The sending process will block until the message is written to the target queue. A return status will indicate if the message successfully enqueued to the queue.
- **Blocking and nonblocking message Store-and-Forward.** The sending process will send messages that are stored in a disk file locally, before they are sent over any communication link(s).

- **Blocking and nonblocking message disk queuing.** The sending process will send messages that will be delivered to the target's disk queue.

DECmessageQ for UNIX provides support for only the "datagram" and "wait for enqueue" delivery options.

By using a recoverable delivery option, DECmessageQ for VMS stores the message on a nonvolatile disk to be delivered as soon as it is possible. These options are available through the Message Recovery Services (MRS).

20.1.6 DECmessageQ Application Programming Interface

DECmessageQ offers code portability through its standard application programming interface (API). The DECmessageQ API is a set of services for application developers that support the exchange of information in a multivendor environment. Basic services enable applications to attach to the DECmessageQ message queuing bus, send and receive messages, and detach from the message queuing bus. DECmessageQ extended services enable applications to receive messages asynchronously, use timers, locate a queue, use journaling for message recovery, and confirm recoverable message delivery.

The basic services common to all DECmessageQ environments include the following functions:

Table 20–1 DECmessageQ Basic Service Functions

Function	Service	Description
pams_attach_q	Attaching	Attaches a process to the DECmessageQ message queuing bus at a specified queue
pams_put_msg	Sending	Captures the contents of the user's buffer and sends it to a target queue or broadcast stream
pams_get_msg	Receiving	Reads the next message from the queue and moves the message to the user's local buffer
pams_get_msgw		If the queue is empty, waits for a message, then reads the message from the queue and moves the message to the user's local buffer
pams_exit	Detaching	Detaches the process from the DECmessageQ message queuing bus

These functions must be used to establish basic messaging functionality in an application using the DECmessageQ software. Each program must attach to the queuing bus at a specified queue and each process must detach from queuing bus when finished with DECmessageQ services. Sending messages is done through the **pams_put_msg** function, which specifies how and where each message is sent as well as the selection criteria. Messages can be received synchronously or asynchronously. You can associate a timer with the wait period, affording the programmer more control over process synchronization.

The extended DECmessageQ services are designed to be used with the basic functions. The extended services provide greater control over sending and receiving messages. The DECmessageQ extended services include the following functions:

Table 20–2 DECmessageQ Extended Service Functions

Function	Service	Description
pams_get_msga	Receiving Asynchronously	Allows multiple asynchronous read operations with fully selective reception
pams_cancel_get		Cancels pending get-message requests that match the selection filter
pams_set_timer	Using Timers	Sets a relative DECmessageQ timer and sends a message on timer expiration
pams_cancel_timer		Cancels specified DECmessageQ timers and their associated timer expiration messages
pams_locate_q	Locating a Queue	Returns the queue address for the specified queue name
pams_open_jrn	Journaling	Requests an identifier used in requests for records from an MRS postconfirmation journal (VMS only)
pams_read_jrn		Returns the next message from the specified postconfirmation journal file (VMS only)
pams_close_jrn		Closes the specified postconfirmation journal file (VMS only)
pams_confirm_msg	Confirming	Informs the MRS Server that the receiver program processed the message (VMS only)

By receiving messages asynchronously, an application becomes interrupt-driven. Whenever the selected message is queued, it is delivered to the receiver process, which immediately executes the action routine. Timer services allow the user to set a timer and receive notification when the timer expires. Timers allow the application to receive messages after a specified period of time and to send a priority 1 message on timer expiration. Timers can also be set to dequeue messages that cannot be delivered within a specified period of time.

Message Recovery Services (MRS) (VMS only) uses journaling for message recovery. The journaling calls allow for opening, reading from, and closing a postconfirmation journal file. Note that not all environments currently support Message Recovery Services.

20.2 DECmessageQ for UNIX

DECmessageQ for UNIX is the product's implementation for use on a host system running the UNIX operating system. It provides easy-to-use, efficient task-to-task communications among processes on ULTRIX on a VAX or MIPS-based system, HP-UX, SunOS, and SYSTEM V/88.

A common call interface allows messages to be delivered via local interprocess communications for intra-CPU applications, or via Transmission Control Protocol/Internet Protocol (TCP/IP) for applications. Applications can be designed so that client applications can be redeployed easily anywhere within the DECmessageQ network configuration, whether in a standalone node, a local area network (LAN), or a wide area network (WAN).

DECmessageQ for UNIX includes the following features:

- High speed local message delivery using local interprocess communications
- Remote message delivery via TCP/IP and DECnet
- Selective reception of messages by queue number (FIFO), priority, or by direct item access methods
- Use of intra-CPU naming through a local naming service
- User-specified timers with timer expiration delivered via messages placed in the user's primary queue
- Dynamic addition of CPUs to the communication network
- Support for the local C compiler
- Connectivity to DECmessageQ implementations on ULTRIX, VMS, SunOS, HP-UX, and SYSTEM V/88.

DECmessageQ for UNIX can also be used as an integration tool to merge many external events with the message queuing bus. In addition to integrating messages from local and remote processes, DECmessageQ provides for the integration of other events such as timer settings, simulated messages, or external events, such as special hardware I/O.

20.2.1 DECmessageQ for UNIX Message Recovery Services

Message Recovery Services is not supported for this release of DECmessageQ for UNIX, but will be added in a future release.

20.2.2 DECmessageQ for UNIX Selective Broadcast Services

Selective Broadcast Services is not supported for this release of DECmessageQ for UNIX but will be added in a future release.

20.3 DECmessageQ for VMS

DECmessageQ for VMS is the product's implementation for use on host systems running VMS. It provides efficient task-to-task communications among processes using DECmessageQ on VMS, UNIX, MS–DOS, SunOS, Macintosh, and OS/2.

DECmessageQ messaging is asynchronous and nonblocking. A single message or a series of messages can be queued for delivery and the sending process is not blocked while waiting for message delivery. This flexibility results in faster communication with less overhead.

The DECmessageQ communication implementation is designed for ease-of-use, expandability, and efficiency. In addition to the features listed in Section 20.1.2, DECmessageQ for VMS includes the following features:

- Fully asynchronous and synchronous receipt of messages
- Message recovery using Message Recovery Services (MRS)
- Message broadcasting using Selective Broadcast Services (SBS)
- Use of intra-CPU naming through high-speed global sections, and inter-CPU naming through VAX Distributed Name Service
- Remote message delivery via DECnet and TCP/IP for inter-CPU applications
- Message simulation facility for use in application testing
- Selectable tracing of Messages and calls to DECmessageQ
- Integration of VMS timers
- Support for languages adhering to the VAX Common Language Calling standard
- Dynamic addition of message queues and CPUs to the communication network

- Message interface for retrieving DECmessageQ configuration information on-line
- Connectivity to DECmessageQ implementations on VMS, UNIX, OS/2, MS–DOS, Macintosh.
- DECmessageQ LU6.2 Services for VMS systems

DECmessageQ for VMS can be used as an integration tool to merge external events with the message queuing bus. In addition to integrating messages from local processes and remote processes, DECmessageQ provides for the integration of other events such as timer settings, simulated messages, LU6.2 conversations, or other external events such as special hardware I/O.

20.3.1 DECmessageQ for VMS Message Recovery Services

DECmessageQ messages have selectable persistence. If your implementation supports Message Recovery Services (MRS), you can track the progress of messages through the network. DECmessageQ Recovery Services enable applications to define the level of recovery required. Using nonvolatile storage, Message Recovery Services ensure that message delivery can be completed following network, system, or application failure. Information critical to an application will arrive at its destination.

Message Recovery Services (MRS) extends data recovery to the level of pending messages. With MRS, the sender is relieved of the responsibility of tracking the progress of a message through its next level of processing. This functionality can be used at both the client and server sides of the application.

Some of the needs addressed by MRS are:

- Sender wishes to insure delivery of messages when the receiving process is available but does not wish to monitor the delivery.
- Sender wishes to know that a message is recoverable to avoid the cost of reconstructing it but does not care when it is finally delivered.
- Receiver wishes to maintain a journal of all messages received by it for audit trail or reprocessing.

Message Recovery Services are implemented primarily by an MRS server, a nonprivileged program attached to the DECmessageQ Message Bus. MRS actions are invoked by standard DECmessageQ send-and-receive message calls.

MRS is oriented toward messages, not processes. That is, not all messages sent from or directed to a particular process need to be processed by MRS. This allows applications to incur the additional processing imposed by MRS for just those messages that are not easily recovered. Message recovery characteristics are set by the sending process.

Message Recovery Services increase the robustness of DECmessageQ by permitting applications to recover from message delivery failures due to a program abort, communication line failure, or system crash.

20.3.2 DECmessageQ for VMS Selective Broadcast Services

Two important application requirements are handled by the Selective Broadcast Services (SBS). The first is the ability to send a message to many targets without going through multiple send message requests. The second is the ability to generate broadcast messages without the originator knowing the quantity or location of the target processes.

Selective Broadcast Services provide a broadcast stream of data into which any process can insert a message. Any process can select messages from this broadcast stream for delivery. Messages may be selected using a set of rules that provide relational comparisons against DECmessageQ header information or user's message data information. The Selective Broadcast Services operate in a single server environment or between cooperating DECmessageQ servers. When the SBS is operating between nodes, it can operate using DECnet services or using direct Ethernet operations. In the direct Ethernet operations, the SBS will optimize message traffic by using Ethernet's multi-casting capabilities.

20.3.3 DECmessageQ for VMS Developer's Tool Kit

The DECmessageQ Developer's Tool Kit provides the tools necessary to speed development time. With Capture, Simulation, and Replay, building, testing, and debugging become more efficient.

The DECmessageQ Capture tool can be used to selectively capture messages that are sent or received from a process. These messages can be displayed on the output device or collected in a disk file. The messages are displayed in an ASCII file using the DECmessageQ scripts syntax.

The DECmessageQ Simulation tool uses a disk file containing DECmessageQ commands to simulate message traffic to a process. This allows you to simulate messages from an application not yet developed in order to test and debug an existing application. Using the simulator, developers can test code response to queued messages immediately without waiting for integration testing.

The DECmessageQ Replay tool simulates messages that were previously captured, making repetitive application testing easier. Message replay software also generates a log file suitable for isolated unit testing.

20.3.4 DECmessageQ for VMS LU6.2 Services

IBM Lu6.2 is a synchronous and proprietary communications protocol. DECmessageQ LU6.2 services allow you to connect to machines supported by DEC's SNA gateway running LU6.2 APPC using standard DECmessageQ messages.

DECmessageQ LU6.2 Services for VMS systems (License Option) allows user programs attached to a DECmessageQ Message Queuing Bus to request, accept, and conduct APPC (Advanced Program-to-Program Communications) conversations with programs running under CICS/VS, IMS/VS, and other IBM environments over SNA LU6.2 sessions.

DECmessageQ LU6.2 Services allows users unfamiliar with SNA communications to develop quickly and easily applications that operate with IBM-based APPC applications. Users are insulated from the details of both SNA and APPC. DECmessageQ provides additional insulation between the two environments by translating transaction program names known to DECmessageQ user programs to names known to the IBM system(s) and vice versa.

DECmessageQ LU6.2 Services for VMS consists of a Port Server, a data structures library, server management utilities, and development tools. DECmessageQ LU6.2 Services for VMS has the following limits:

Table 20–3 DECmessageQ for VMS LU6.2 Services Limits

Description	Limit
Maximum user message size	31,982 bytes
Maximum number of active LUs	256 per Port Server
Maximum number of remote transaction programs	512 per DECmessageQ LU6.2 Server (user-written servers: unlimited)

DECmessageQ LU6.2 Services for VMS includes a management utility that allows a suitably privileged user to start, stop, and configure DECmessageQ LU6.2 servers and observe their activity (through an event monitoring facility).

Users can develop specialized LU6.2 servers by using the DECmessageQ LU6.2 Server tool kit. The tool kit consists of a DECmessageQ for VMS User Callback (a "user callback" is a special User Exit module) and 21 predefined DECmessageQ message structures.

20.4 DECmessageQ Hardware and Software Requirements

This section identifies hardware and software requirements for developing and running applications using DECmessageQ.

DECmessageQ supports DECnet/Ethernet connections for VMS (server), Macintosh, MS–DOS, and OS/2 (client). TCP/IP connections are supported for UNIX and ULTRIX (servers).

20.4.1 DECmessageQ Hardware Requirements

A configuration for running DECmessageQ requires the following hardware:

- A system to act as the server host system that runs networked applications
- A workstation or terminal to act as the client system that runs networked applications
- An Ethernet connection between the server system and the client workstation or terminal

For a list of host and client processors supported by the DECmessageQ software, refer to the *System Support Addendum (SSA)* for the specific DECmessageQ product. You may obtain these from your Digital Sales Representative.

20.4.2 DECmessageQ Software Requirements

This section outlines the minimum software required to run DECmessageQ on VMS and UNIX platforms. For additional requirements, refer to the *System Support Addendum (SSA)* for the specific DECmessageQ product.

DECmessageQ for VMS Requirements

- VMS Operating System
- VMS/ULTRIX Connection (UCX) for the TCP/IP protocol environment
- DECnet/SNA APPC LU6.2 Programming Interface (for LU6.2 Services Option only)

DECmessageQ for UNIX Requirements

- ULTRIX, HP-UX, SunOS, or SYSTEM V/88 Operating System
- DECnet–ULTRIX networking software for the DECnet protocol environment

20.4.3 DECmessageQ System Configuration

Before you can develop and run DECmessageQ applications, you must define the runtime environment that is best for your application and system configuration.

Using DECmessageQ in a VMS environment, you must allocate global memory, define VMS symbols, and set system parameters. Using DECmessageQ in a UNIX environment, you must allocate shared memory, semaphores, and process slots then rebuild the UNIX kernel.

Part VII

DEC Realtime Integrator

Part VII surveys DEC Realtime Integrator as a development tool for realtime applications on ULTRIX, VAXELN, and VMS systems. This part contains the following chapters:

- Chapter 21, DEC Realtime Integrator Overview, describes the DEC Realtime Integrator for ULTRIX, VAXELN, and VMS.
- Chapter 22, DEC Realtime Integrator Programming, discusses the icon-based graphical user interface and subroutine libraries of the DEC Realtime Integrator software.

21 DEC Realtime Integrator Overview

The DEC Realtime Integrator is a window-based icon toolkit that can run on VMS systems and ULTRIX on MIPS-based systems. It allows for rapid development of realtime data acquisition and test applications by drawing them as flow diagrams. Typical applications include data acquisition, IEEE–488 and RS–232 instrument control, and test and measurement.

Each DEC Realtime Integrator icon represents a function such as an analog or digital input, an arithmetic operation, or a logical function. With the DEC Realtime Integrator you create applications by using a mouse to select icons from different icon libraries, placing them on a work surface on the computer screen, connecting the icons with data flow or signal flow lines, and performing further setup. Finally, to run a program, you simply click on the start button.

For the original equipment manufacturer (OEM) and the end user working with test and realtime applications, DEC Realtime Integrator can simplify the development of high-quality test and research solutions. The more difficult the programming challenge, the larger the productivity benefit provided by DEC Realtime Integrator.

This chapter describes the following:

- DEC Realtime Integrator General Features, Section 21.1
- DEC Realtime Integrator Use, Section 21.2
- DEC Realtime Integrator for VAXELN, Section 21.3
- DEC Realtime Integrator Hardware and Software Requirements, Section 21.4

With DEC Realtime Integrator, application developers may use icons included with the DEC Realtime Integrator, custom icons developed with a DEC Realtime Integrator Development System, or icons sold separately by third parties.

21.1 DEC Realtime Integrator General Features

DEC Realtime Integrator provides window-based icon tools for creating and running realtime applications on ULTRIX on MIPS-based systems, VAXELN, or VMS systems. It enables test engineers and researchers who lack programming skills to create new applications, while allowing sophisticated programmers to create applications of great complexity. It increases OEM productivity by providing a DECwindows-based platform upon which customized test and measurement applications can be easily built. Users need not know how to program DECwindows or Compound Document Architecture (CDA) applications.

As a window-based product, DEC Realtime Integrator provides an integration platform for ULTRIX, VAXELN, or VMS workstations, eliminating the need for custom programming to link test instruments to such environments as RS/1, SAS, DDIF, DTIF, or NAS (depending on the operating system).

DEC Realtime Integrator features include:

- **Graphical programming approach.** The icon-based method of programming allows users to create applications simply, without coding programs. As a result, less training and development time is required for producing test and research applications.
- **Ability to develop your own icons.** Using template code files, users can use the C language to create their own icons to represent software functions. Thus you can expand DEC Realtime Integrator and customize applications by adding functions.
- **Modularity of applications.** A complete application can be shrunk to a single icon or macro. Once created, the application icon can be used like any other icon or macro. It can be treated as a reusable code module, serving as a building block for more complex applications.
- **Macro editing.** Application developers can expand and edit icon macros in a separate window.
- **DECnet and TCP/IP communication.** Applications distributed across multiple computing systems can share data directly with each other through DECnet or TCP/IP communication.
- **Automatic data type conversion.** DEC Realtime Integrator now converts data types automatically. You no longer need to use the Converter icon, except for conversion to and from fixed-length ASCII formats.
- **Multiple control panels.** Multiple control panels are allowed in each application. This features allows for more sophisticated interfaces.

- **Data paths can be prioritized.** A path priorith of Low, Medium, or High offers more precise control over order of execution.
- **Debug utility.** The debug utility streamlines application development.
- **Data can be timestamped.** You can associate a timestamp with data as it is collected and reference data by its timestamp.
- **Support of VAX and MIPS-based realtime hardware options.** DEC Realtime Integrator supports I/O hardware that allows you to perform testing and measurement on ULTRIX, VAXELN, and VMS systems.

  Under ULTRIX, the supported hardware includes IEEE–488 instrument bus interfaces, TURBOchannel-to-IEEE–488 adapters, and RS–232 serial interfaces.

  Under VAXELN, the supported hardware includes VAX/VME single-board computer, the KAV30–AD.

  Under VMS, supported hardware options include analog-to-digital converters, digital-to-analog converters, IEEE–488 instrument bus interfaces, SCSI-to-IEEE–488 adapters, RS–232 serial interfaces, and realtime clocks.
- **Seamless integration with other manufacturing and research applications.** Icons are available to transfer data to database and data analysis applications, such as RS/1, SAS, and Signal Technology's N!POWER. Other icons can transfer data through DDIF and DTIF standards to CDA applications.
- **Multitasking environment.** DEC Realtime Integrator can run multiple test and measurement applications concurrently.
- **DECwindows compatibility.** Applications can be controlled from or displayed on a workstation or terminal that is compatible with the X Window System.

21.2 DEC Realtime Integrator Use

DEC Realtime Integrator is a significant part of Digital's solution in the scientific, engineering, and industrial research marketplaces. DEC Realtime Integrator is targeted at single-user and research team applications. But it is easily integrated into a total scientific, engineering, or industrial computing environment, including larger VMS and ULTRIX on MIPS-based systems at the departmental and organizational levels. Therefore, DEC Realtime Integrator should be viewed as a sophisticated and powerful research and testing tool that you can use to:

- Model or create an experiment
- Control distributed serial devices over the network
- Gather, reduce, and analyze test data
- Store, retrieve, or display data in a number of graphical or textual formats

DEC Realtime Integrator provides an integration base in testing, research, and laboratory environments such as these:

- Electronics
  - Wafer and component test
  - Off-line quality test
  - Design verification
  - RF and microwave testing
- Automotive
  - Engine dynamometer test
  - Materials research and test
  - Fatigue and life tests
  - Environmental test
  - Subassembly testing
- Aerospace electronics
  - Subassembly test
  - Maintenance test
  - Design verification
  - Performance characterization
- Government and education
  - Research and development
  - Teaching tool
- Process
  - New products and materials research
  - Pilot plant simulation

- Utilities
  - Energy monitoring
  - Battery testing
  - Power measurements

21.3 DEC Realtime Integrator for VAXELN

The VAXELN runtime environment is useful in realtime applications where a dedicated VAX system is used to repeatedly perform a set of tasks or where the presence of a disk-based general-purpose operating system such as VMS or ULTRIX is not required. An example would be a process monitoring system in a chemical plant; in such an application, predictable performance of a predefined set of tasks is an important selection criterion for a realtime system. A VAXELN system image can be built under VMS and can contain only the VAXELN kernel executive and the programs, services, and device drivers needed to accomplish the defined tasks. The system image is loaded into a target VAX, where it performs its set of tasks, independently of the VMS development system.

DEC Realtime Integrator for VAXELN is an icon-based, graphical programming environment that allows for rapid development of realtime data acquisition and test applications in an embedded VME environment. This version of DEC Realtime Integrator runs on a KAV30–AD single-board computer. It provides the DEC Realtime Integrator drawing worksurface and icon libraries for creating and running applications graphically.

The VAXELN Realtime kernel provides the necessary system services and interfaces to allow for application development in an embedded realtime environment. The VAXELN system image is created on a host VMS system and is then loaded into the target single-board computer over the Ethernet. Application development can be done directly on the single-board computer (SBC) using an X-windows terminal.

DEC Realtime Integrator for VAXELN provides tools to develop custom icons on the host VMS system, then include the icons for use on the embedded system using the VAXELN Toolkit System Builder. For example, you can reuse previously written code by including it in a DEC Realtime Integrator icon.

DEC Realtime Integrator and VAXELN systems complement each other in Digital's distributed realtime architecture and can coexist beneficially in the laboratory, factory, and other realtime environments. Figure 21–1 illustrates how DEC Realtime Integrator fits into Digital's distributed realtime architecture. The following systems are in a laboratory or testing environment:

- DECstation 3100, running ULTRIX and loaded with DEC Realtime Integrator tools and applications
- rtVAX 4000 Model 300 system, running a customized VAXELN system image and functioning as a dedicated realtime platform
- VAXstation 3200, running the VMS operating system and loaded with DEC Realtime Integrator tools and applications

Realtime instrument control is being performed by the DECstation 3100 using Digital's IEZ11 SCSI to IEEE–488 converter to connect to IEEE–488 compatible laboratory instrumentation. At the same time, realtime test and measurement is being performed by the VAXstation 3200 using Digital's IEQ11 Q–bus to IEEE–488 converter. And, the rtVAX 4000 Model 300 serves as a dedicated VAXELN system that is accumulating realtime data from a production system.

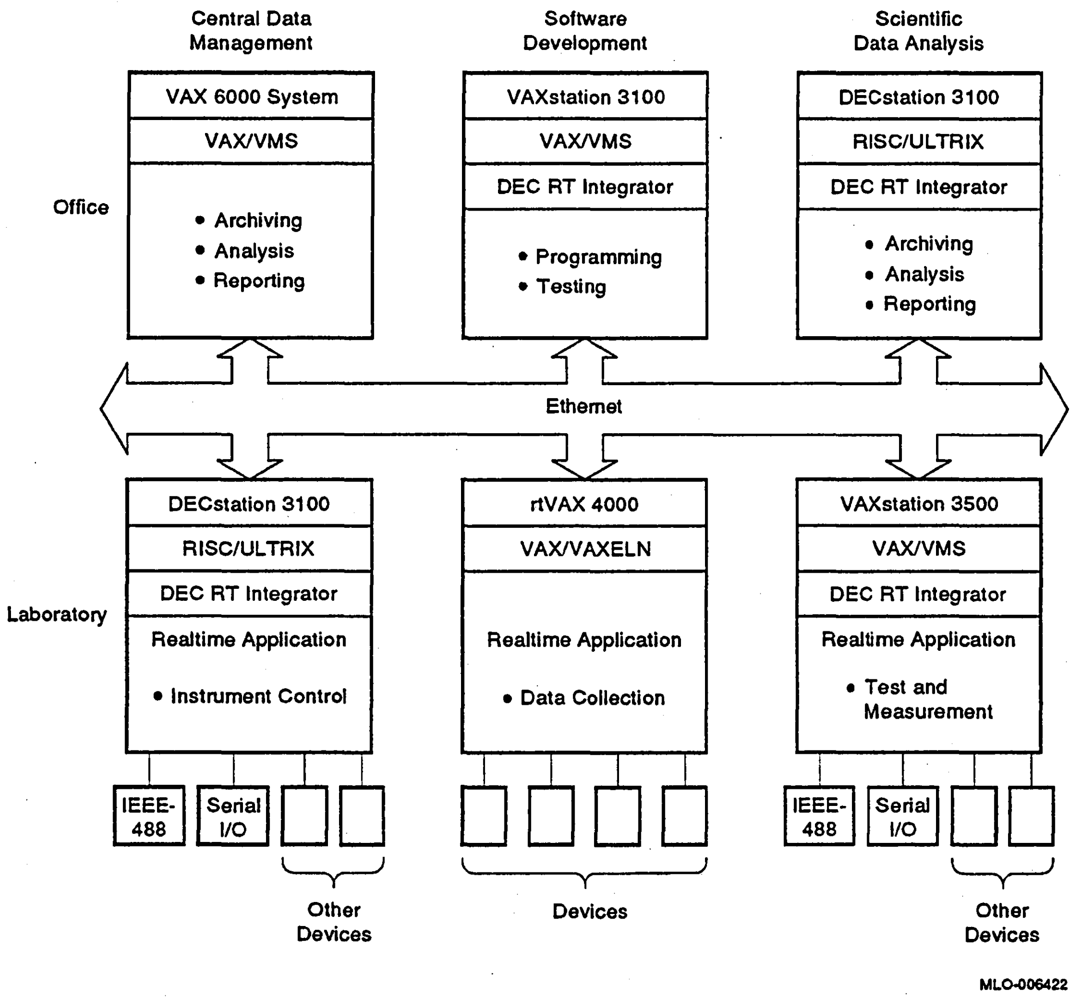
In a scientist's office, a DECsystem 3100 runs ULTRIX and is loaded with DEC Realtime Integrator tools and applications. The DECsystem is used for analysis of the data that is being collected from the other three realtime systems in the laboratory, signal processing, and graphic display. Since this system is part of the corporate network, the scientist can conveniently archive experimental data that later can be accessed and analyzed by other individuals and work groups.

In a programmer's office, a VAXstation 3100 is running VMS and is loaded with DEC Realtime Integrator tools and applications. The VAXstation is used to program and test DEC Realtime Integrator applications, including building blocks for future laboratory or testing applications. Application configurations are saved in a machine-independent format and therefore are portable between systems loaded with DEC Realtime Integrator. Thus, applications created on this VAXstation 3100 can be transported to any of the other DEC Realtime Integrator systems, whether VAX or MIPS-based, and run with very few modifications.

21.4 DEC Realtime Integrator Hardware and Software Requirements

DEC Realtime Integrator runs in ULTRIX, VAXELN, and VMS environments.

Figure 21–1 DEC Realtime Integrator in Digital's Distributed Realtime Architecture



For ULTRIX the minimum hardware required for DEC Realtime Integrator is a DECsystem or DECstation as specified in the *System Support Addendum*. The software requirements are as follows:

- If you use terminals without a DECwindows interface, the only software requirement is ULTRIX Version 4.2A or higher

- If you use workstations (either with or without a DECwindows interface) the software requirements include the ULTRIX Worksystem Software Version 4.2A, DEC GKS–3D for ULTRIX Version 1.1 or higher, and the C language (if you are developing custom icons, customizing libraries, or including third party icons).

A DEC GKS–3D Runtime-only license is the minimum prerequisite software necessary for application developers to use the DEC Realtime Integrator icon-based visual programming environment. For developers using the VAXlab Software Library (VSL) subroutine libraries in the DEC Realtime Integrator Runtime, a DEC GKS–3D development license may also be necessary to allow use of the GKS development facilities.

For VAXELN the minimum hardware required for DEC Realtime Integrator is a VAX, MicroVAX, VAXserver, or VAXstation as specified in the *System Support Addendum*. The hardware requirement for the target processor is a KAV30–AD, VT1300, or VXT2000. The software requirements are as follows:

- If you use terminals without a DECwindows interface, the software requirements are VMS Version 4.7 or higher and VAX C Version 3.2 or higher (if you are developing custom icons, customizing libraries, or including third party icons).
- If you use DECwindows, the software requirements include the VMS Operating System, VMS DECwindows Compute Server, VMS DECwindows Device Support, VMS DECwindows Programming Support, and VAX C Version 3.2 or higher (if you are developing custom icons, customizing libraries, or including third party icons).

For VMS, the minimum hardware required for DEC Realtime Integrator is a VAX, MicroVAX, VAXserver, or VAXstation as specified in the *System Support Addendum*. The software requirements are as follows:

- If you use terminals without a DECwindows interface, the software requirements are VMS Version 5.4 or higher.
- If you use workstations running VWS, the software requirements include the VMS Operating System Version 5.4 or higher (or VMS Workstation Software [VWS] Version 4.4 or higher), VMS DECwindows Compute Server, VMS DECwindows Device Support, and VMS DECwindows Programming Support.

  Note that the DEC Realtime Integrator graphical drawing editor is not available on a VWS window system.

- If you use workstations running DECwindows, the software requirements include the VMS Operating System, VMS DECwindows, DEC GKS for VMS Version 4.1 or higher, and VAX C Version 3.2 or higher (if you are developing custom icons, customizing libraries, or including third party icons).
- DRB32 VMS drivers Version 3.0 are a prerequisite for LIO support of the DRB32 and DRB32W devices and I/O subsystems that interface to the VAXBI through the DRB32 and DRB32W.
- The RTC01 BI clock is a prerequisite to use BI devices through LIO I/O support.

22 DEC Realtime Integrator Programming

DEC Realtime Integrator provides an icon-based, graphical programming environment. Instead of using a conventional programming language, the user can create and run realtime applications by drawing them as flow diagrams.

This chapter describes the following aspects of DEC Realtime Integrator programming:

- Programming Environment, Section 22.1
- Icon Libraries, Section 22.2
- Subroutine Libraries, Section 22.3

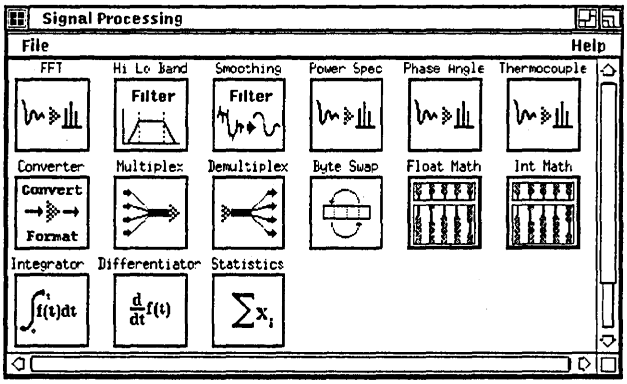
22.1 Programming Environment

Each DEC Realtime Integrator icon represents a function, such as an analog or digital input, an arithmetic operation, or a logical function. DEC Realtime Integrator provides libraries of such functions (described in Section 22.2). Figure 22–1 shows one of the icon libraries provided by DEC Realtime Integrator, the Signal Processing Library.

With DEC Realtime Integrator you build an application by using the point-and-click method now standard in graphical user interfaces. You use the mouse to move icons from the libraries to the work surface and to connect the icons with data flow or signal flow lines. Once the icons have been set up, the application is ready to run.

The work surface acts as a graphical editor, letting you move, copy, delete, add, cut, and paste icons representing devices and sections of code. Thus, you can create and change applications much more quickly than with traditional programming languages.

Figure 22–1 Signal Processing Icons



Each icon has a setup dialog box that allows customization of its functioning for the particular application. Once built, the application can be saved and restored without the need to repeat the setup operations. Since the configuration is saved a machine-independent format, it can be moved to any other system running DEC Realtime Integrator, whether MIPS-based or VAX, and run with very few modifications. (One constraint on application portability would be the use of platform-specific icons, for example, the VMS-only ADQ32 converter icon, in the application.)

DEC Realtime Integrator includes several icon libraries, and you can extend its capabilities by creating your own icons. DEC Realtime Integrator includes a C template and detailed instructions on how to modify the template. You edit the template and then compile and link the function with the user library. The icon is now part of your library and can be used like any other. Since all icons are modular functions, your icon can be used in future applications.

By placing icons together and shrinking them to form a single icon, you can build macros (compound groups of icons) and then manipulate the macros like any other individual icon. A macro editing facility allows users to edit icon macros in a separate editing window. This allows you to create higher-order representations of an application from basic building blocks. For example, you could create a macro, representing a complex flow diagram, for an instrument and then use it in subsequent test applications.

Context-sensitive online help makes the software environment easy to learn. By using the point-and-click method, you can get help on a selected icon. When building an icon, you can add help text for that icon.

When you use the data display and panel input icons, DEC Realtime Integrator automatically creates a control panel. The control panel is a simple user interface representing an instrument control panel that nonprogrammers can use to interact with the application. A user can customize the control panel by selecting the appropriate icons and positioning the devices they represent (such as sliders, push buttons, and so forth) on the control panel surface. Multiple control panels in a single application allow developers to build more sophisticated user interfaces using multiple windows.

Each application has two conceptual views: a data flow view and a signal flow view. DEC Realtime Integrator allows you to switch between these views with the click of a mouse button. In the data flow view, the connection between icons represents the flow of data through the application; the signal flow view displays the application's logic structure.

Figures 22–2 and 22–3 show the data flow and signal flow of a sample IEEE–488 application. This application uses the IEEE–488 icon to send commands to the IEZ11 controller and receive data from an instrument connected to the IEEE–488 bus. Pushing a button on the control panel sends a read signal to the IEZ11 device, which then receives data from the instrument. When the data arrives, it is sent to the counter, and then to the scope icon for display on the control panel. Each time the counter receives data, it sends a ticked output signal to the IEEE–488 icon's read port, which causes the read cycle to begin again. Data is thus continuously read from the instrument and plotted on the control panel.

The DEC Realtime Integrator programming environment simplifies IEEE–488 instrument control. You directly program the instrument and need no knowledge of the bus itself. String manipulation and other support icons simplify the construction of command strings. The service request (SRQ) can be detected as a signal in the signal flow view, while the SRQ status byte is available as data in the data flow view. The mechanisms allow instrument events to be detected and serviced with the same graphical techniques used to create the rest of the application.

During data collection, you can monitor data, displaying it as a line plot, point plot, or scrolling text. Multiple plotting windows are supported on all supported terminals. You can insert display icons in any path of the data flow and use them to probe or trace the data as it flows through the application. You can also generate PostScript hardcopy records of the plots.

Figure 22–2 IOtech WAVE488 Application Data Flow View

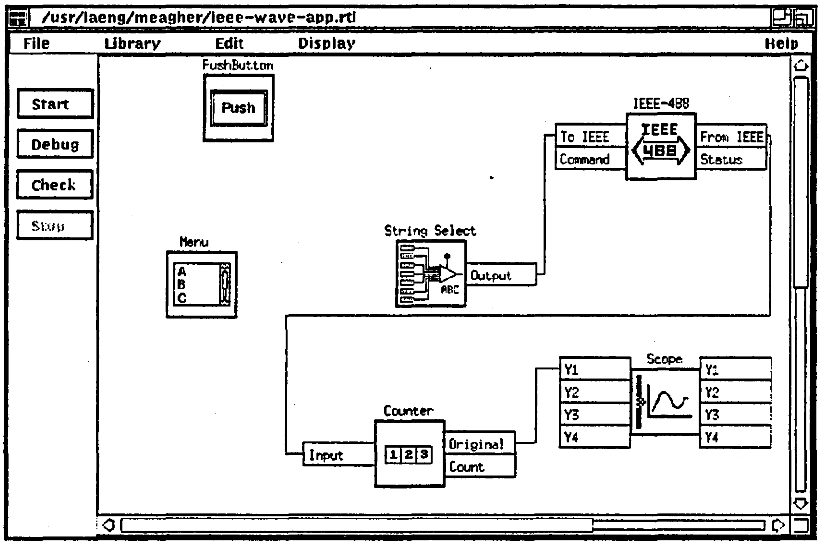
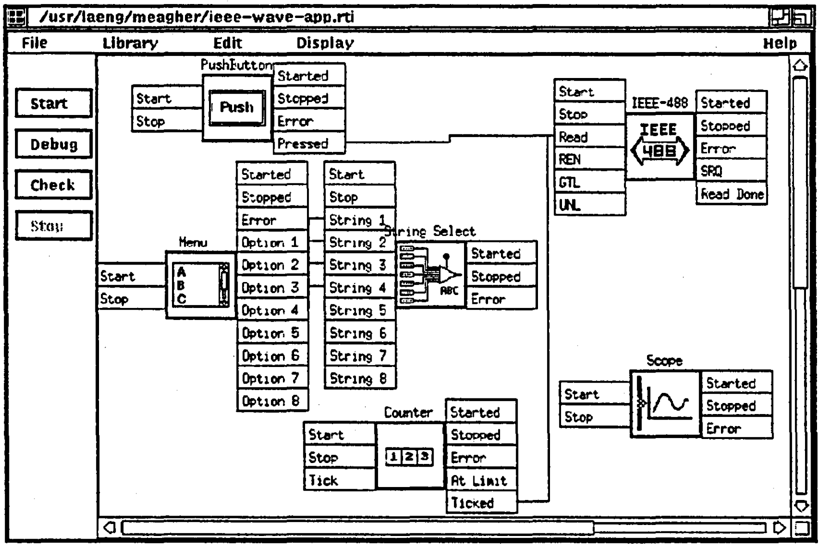


Figure 22–3 IOtech WAVE488 Application Signal Flow View



A Debug Window is provided to help you debug an application. You can set breakpoints on data and signal flow lines to step through an application and debug it.

In addition to traditional mathematical and trigonometric functions, icons are provided to perform noise filtering, scaling, multiplexing, or signal processing functions such as fast Fourier transformations (FFTs) and power spectrum analysis. A range check icon lets you determine if a data buffer falls outside a given value range and take corrective action.

Time generation functions allow you to trigger applications or events using a realtime or system clock. In addition, a time stamp can be placed on data buffers.

Because DEC Realtime Integrator is compatible with DECwindows, applications can be controlled from or displayed on any workstation, personal computer, or terminal that is compatible with the X Window System.

22.2 Icon Libraries

DEC Realtime Integrator includes the following icon libraries, which together provide an extensive selection of icons that are useful in realtime application development:

- Hardware I/O Library
- Software I/O Library
- Text Manipulation Library
- Signal Processing Library
- Control Library
- Data Display Library
- Panel Input Library

22.2.1 Hardware Input/Output Library

Hardware I/O icons help control physical I/O devices connected to your system. Table 22–1 lists the Hardware I/O Library icons for ULTRIX, VAXELN, and VMS.

Table 22–1 Hardware I/O Icons

Icon	Function
IEEE–488	Communicates with bus devices that conform to the IEEE–488 or the IEC-625 standard.

(continued on next page)

Table 22–1 (Cont.) Hardware I/O Icons

Icon	Function
Serial Line	Performs input and output to a serial line using RS–232 and DEC422 connections.
File Input	Reads data of any type from an existing file.
File Output	Writes data of any type to a file.
ASCII Report Generator	Uses numbers as input data to create a report containing these numbers in table format. The report is saved as an ASCII file.
DTIF Table	Creates and reads tables of data.
System Time	Generates the system date and time and sends the data as ASCII text or binary output.
Network Input	Provides a DECnet or TCP/IP network connection so that you can receive data from another application.
Network Output	Provides a DECnet or TCP/IP network connection so that you can send data from another application.
Binary Input	Reads raw binary data from an existing file.
Binary Output	Writes raw binary data from an existing file.
DTIF Output	Creates a DTIF data file and writes output to it.
ADQ32[1]	Permits an ADQ32 analog-to-digital converter with DEC Realtime Integrator.
ADV11-D[1]	Permits an ADV11-D analog-to-digital converter with DEC Realtime Integrator.
AAV11-D[1]	Permits an AAV11-D analog-to-digital converter with DEC Realtime Integrator.
KWV11-C source[1]	Permits a KWV11-C realtime clock device as a rate generator or to trigger another device.
KWV11-C counter[1]	Permits a KWV11-C realtime clock device to count events, measure time intervals, and measure frequency.
DRQ3B[1]	Permits a DRQ3B DMA parallel I/O controller with DEC Realtime Integrator.
Preston[1]	Permits a Preston GM or EM series analog-to-digital converter with DEC Realtime Integrator.

[1]VMS only.

22.2.2 Software Input/Output Library

The software I/O icons provide I/O functions not directly associated with physical I/O devices. Table 22–2 lists the Software I/O Library icons.

Table 22–2 Software I/O Icons

Icon	Function
Function Generator	Simulates a function generator and produces three kinds of waves: sine, square, and triangle.
Fan In	Copies the contents of all connected input ports to the output port.
Fan Out	Copies the data on the data path coming into it and sends the copied data out.
Accumulator	Accumulates data in a buffer and sends it as output; you can use this icon to create a data stream with a specified buffer size.
Interprocess Input	Transfers data from a DEC Realtime Integrator or Laboratory Input/Output (LIO) application on the same system. (Not supported for VAXELN.)
Interprocess output	Transfers data to a DEC Realtime Integrator or Laboratory Input/Output LIO application on the same system. (Not supported for VAXELN.)
Constant	Sends a constant value.
Initial Value	Fills the first buffer of data with an initial value when an application starts, then passes data through; used for control loops.
Scale	Adjusts values in an input data buffer to a predetermined range and sends the scaled result as output.
Record	Creates records to be used by other icons.
Get Field	Makes record data available for use by icons that are not capable of processing records directly.
Set Field	Makes data in standard buffers available for processing by icons that use record-type buffers.
Get Timebase	Accesses timebase information associated with your data.
Set Timebase	Associates timebase information with your data.
Null Input	Used as a placeholder data source icon; passes no data.

(continued on next page)

Table 22–2 (Cont.) Software I/O Icons

Icon	Function
Null Throughput	Used as a placeholder throughput icon; passes data unchanged.
Null Output	Used as a placeholder data sink or throughput icon.
Comment	Adds documentation or comments; the text is saved in the setup dialog box of the comment icon.
Ramp Generator	Generates a sequence of numbers in a predetermined range and sends the numbers as output.
Global Variable	Changes parameter values in the setup dialog boxes for icons while an application is running.

22.2.3 Text Manipulation Library

The text manipulation icons manipulate alphanumeric data. Table 22–3 lists the Text Manipulation Library icons.

Table 22–3 Text Manipulation Icons

Icon	Function
String Concatenation	Concatenates two strings.
Substring	Extracts a substring from an input string.
String Select	Selects up to eight ASCII text strings and send them as output.
Capture Token	Extracts one token (word, line, or number) from an alphanumeric data stream.
Capture Stream	Extracts any number of tokens from an alphanumeric data stream.
Packetizer	Accumulates one or more columns of data from small input buffers into a larger packet or splits a large input buffer containing alphanumeric data into smaller packets.
If_Then_Else	Adds conditional logic to text manipulation applications.

22.2.4 Signal Processing Library

The signal processing icons modify or process signal data. Table 22–4 lists the Signal Processing Library icons.

Table 22–4 Signal Processing Icons

Icon	Function
FFT (fast Fourier transformation)	Performs an FFT (forward or inverse) on data.
Hi-Lo-Band Filter	Performs four types of nonrecursive filtering: highpass, lowpass, bandpass, and notch (bandstop).
Smoothing Filter	Performs polynomial filtering. The icon can filter either raw data or perform one of the following on the data prior to filtering: first derivative, second derivative, or third derivative.
Power Spectrum	Calculates the power spectrum or amplitude of the power spectrum using FFT output.
Phase Angle	Calculates the phase angle and the modulus (vector resultant amplitude) using FFT output.
Thermocouple	Converts thermocouple voltages to centigrade (Celsius) temperatures.
Converter	Converts data from one type to another; types include integer, floating point, binary, and ASCII.
Multiplex	Multiplexes from one to four single-channel buffers to one multichannel buffer.
Demultiplex	Demultiplexes one multichannel buffer to one to four single-channel buffers.
Byte Swap	Reverses byte order from Big Endian to Little Endian or the converse. Can be used to convert Little Endian bytes for Big Endian devices.
Floating-point Math	Performs floating-point mathematical operations on data.
Integer Math	Performs integer mathematical operations on data.
Integrator	Integrates data using the trapezoidal method of numerical integration.
Differentiator	Takes the numerical derivative, using a finite difference of the nth and (n-1) point.
Statistics	Computes one or more statistical values on a data sample; uses 4-byte floating-point format.

22.2.5 Control Library

The control icons produce signals or act on signals to control the application. Table 22–5 lists the Control Library icons.

Table 22–5 Control Icons

Icon	Function
Counter	Counts either data buffers or signals.
Switch Input	Selects either of two inputs and sends the result as output; input can be data, signals, or both.
Switch Output	Receives input and sends it as up output or down output, depending on the switch state; input can be data, signals, or both.
Range Check	Determines if data is above, below, or within a particular range.
Feedback	Compares an input value to a reference value and sends a control value as output.
Time Delay	Sends a signal in response to an input signal after a specified time has elapsed or at a specific time.
State transition	Generates one or more output signals in response to input signals.
Subprocess	Creates a subprocess (child process or spawned process). (Not supported for VAXELN.)
Sequencer	Sends up to eight signals in a sequence.
Floating-point IF	Evaluates a logical expression and performs a floating point mathematical operation based on its value.
Integer IF	Evaluates a logical expression and performs an integer mathematical operation based on its value.
Stop	Stops an application; can be used in a runtime-only DEC Realtime Integrator environment, when the user cannot press the Stop button.
Application	Loads and executes a previously saved (slave) application while you run another (master) application.
Show Panel	Makes a specified control panel visible or hidden.

22.2.6 Data Display Library

The data display icons display data on the control panel. Table 22–6 lists the Data Display Library icons.

Table 22–6 Data Display Icons

Icon	Function
Scope	Plots one channel of data dynamically, in oscilloscope fashion.
Extended Scope	Plots X and Y data dynamically, in oscilloscope fashion.
Line Plot	Plots X and Y data lines in a variety of line thicknesses and colors (the X data is optional). (Not supported for VAXELN.)
Point Plot	Plots X and Y data points in a variety of shapes, sizes, and colors. (Not supported for VAXELN.)
Text Display	Displays ASCII text on the control panel.
Bell	Rings a bell as a signal in an application.
Buffer Watch	Displays details and performance data about buffers received. The information is displayed on the control panel.
Bitmap	Uses X11 bitmaps to create graphics for a control panel.
Light	Lights a button on a control panel in response to a signal from other icons.
Digital Meter	Displays numerical data on a control panel.
Analog Meter	Displays data on a control panel in the form of a moving needle.
Single Value Display	Displays data on a control panel in the form of a single moving histogram.

22.2.7 Panel Input Library

The panel input icons let you control the application from the control panel while it runs. Table 22–7 lists the Panel Input Library icons.

Table 22–7 Panel Input Icons

Icon	Function
Push Button	Sends a signal from a control panel to icons in an application.
Toggle	Turns a setting in an application on or off.
Slider	Inputs numerical (integer or floating-point) values from a range.
Text Entry	Inputs ASCII text from the control panel.
Menu	Creates a menu on the control panel; the created menu can be used to select an option, which sends a signal.

22.3 Subroutine Libraries

DEC Realtime Integrator utilizes subroutine libraries from the previously available VAXlab Software Library (VSL) to perform some of its functions. Although the subroutines used by DEC Realtime Integrator are available as icons (functions) to the end user, the libraries are also directly accessible from program code for creating custom icons or application programs. Under DEC Realtime Integrator for VMS, the subroutines can be accessed from VAX Ada, VAX BASIC, VAX C, VAX FORTRAN, and VAX Pascal. Under DEC Realtime Integrator for MIPS, the subroutines can be accessed from C or FORTRAN for RISC.

DEC Realtime Integrator contains the following subroutine libraries:

- Laboratory Input/Output (LIO) Library
- Laboratory Signal Processing (LSP) Library
- Laboratory Graphics Package (LGP) Library

22.3.1 Laboratory Input/Output (LIO) Library

The Laboratory I/O (LIO) library is a collection of subroutines and realtime device drivers that permit application programs written in high-level languages to perform data collection and control. The LIO library has these features:

- **Support for several types of realtime device I/O.** The LIO library supports polled I/O, interrupt-driven I/O, and direct memory access (DMA) between supported realtime devices and the system's memory. I/O requests can either be synchronous or asynchronous.
- **Device control and data buffering.** The following device control and data buffering functions are available:
  - Attach a device and set it up for QIO, connect-to-interrupt I/O, or mapped I/O
  - Specify the setup parameters of an I/O device
  - Verify the setup parameters of an I/O device
  - Read a buffer of data from a device (synchronous I/O)
  - Write a buffer of data to a device (synchronous I/O)
  - Obtain a buffer from a device queue and put it in the user queue (asynchronous I/O)
  - Enqueue multiple buffers to a device for continuous data transfer (asynchronous I/O)

- — Forward a buffer from one device queue to another (asynchronous I/O)
- — Detach a device and shut down the I/O process in an orderly fashion

22.3.2 Laboratory Signal Processing (LSP) Library

The Laboratory Signal Processing (LSP) library is a set of subroutines that calculate power spectra and perform data format translation, fast Fourier transformations (FFTs), signal processing, digital filtering, and interval histogramming.

The LSP library has these functions:

- **Data format translation.** Converts raw, analog-to-digital binary data to floating-point voltages and vice versa.
- **Fast Fourier transformations (FFTs).** Handles real and complex data in one or two dimensions, in both the forward and inverse directions. Spectral-windowing functions are also available.
- **Calculate power spectra.** Calculates power spectra of equispaced data.
- **Digital filtering.** Can be used as either a low-pass, high-pass, band-pass, or band-cutoff filter. Also included is a polynomial filter based on simple interpolation of polynomials; first, second, and third derivative polynomial filtering; and nonrecursive filtering.
- **Phase angle conversion.** Converts from rectangular to polar coordinates in one or two dimensions.
- **Interval histogramming.** Counts the number of elements in a data stream that fall into one or more predefined categories.
- **Correlation.** Auto- and cross-correlates a data stream.
- **Thermocouple conversion.** Converts thermocouple voltages into temperatures. Thermocouple conversion is available for B-, E-, J-, K-, R-, S-, and T-type thermocouples.

22.3.3 Laboratory Graphics Package (LGP) Library

The Laboratory Graphics Package (LGP) is a comprehensive library of powerful plotting subroutines that require minimal knowledge of computer graphics.

The LGP library performs the following functions:

- **Graphic output to terminal, hardcopy device, or disk file.** You can direct LGP output to a terminal or hardcopy device, or you can save it in disk files for future use.

- **Extensive plotting capabilities.** The LGP subroutines enable you to perform the following plotting-related tasks:
  - Create a two-dimensional linear or logarithmic axis system and plot an array
  - Plot additional data sets on a previously defined graph (up to 16)
  - Provide autoscaling, spline curve generation, and data interpolation
  - Plot standard deviation markers on data points
  - Create a shaded two-dimensional contour projection or add a contour chart to the existing one
  - Draw a histogram
  - Custom plot annotation
  - Plot a three-dimensional array with hidden-line removal, tilt, and rotation
  - Display a menu and return the value selected
  - Prompt for a text string or numeric input
  - Select *n* points from a graph
  - Create multiple plots on a screen
- **DECwindows support.** Supports the DECwindows client/server environment.
- **Graphics for inclusion in compound documents.** Applications can write LGP graphs as Digital Document Interchange Format (DDIF) files for inclusion in compound documents generated by DECwrite or other Compound Document Architecture (CDA) compatible applications.
- **Layering on DEC GKS for a high degree of device independence.** The LGP plotting library is layered on the DEC Graphics Kernel System (DEC GKS), which provides a high degree of device independence.

Part VIII

Realtime Hardware

Part VIII surveys Digital's realtime systems and hardware options offerings. The following chapters are included:

- Chapter 23, Realtime Hardware Overview, introduces Digital's realtime hardware systems and options.
- Chapter 24, Realtime Hardware Product Families, presents Digital's realtime hardware product family for VAX and MIPS-based systems.
- Chapter 25, Realtime Options, describes Digital's realtime bus hardware options for Q–bus, SCSI, TURBOchannel, and VAXBI.

23 Realtime Hardware Overview

Digital's realtime products support a broad range of systems including general-purpose VAX and MIPS-based processors, hardware options, and I/O options for mass storage and communication. Additionally, Digital offers systems and options designed or packaged specifically for realtime applications.

The realtime hardware product family offers chip-level processors (CLPs), single-board computers (SBCs), workstations, and system-level configurations of Digital's 32-bit and MIPS-based computers, housed in many types of enclosures. This realtime product family spans a wide range of processing power and can meet realtime computing needs on many levels.

Digital's realtime systems and options include rtVAX products that serve as VAXELN target platforms, I/O hardware usable in VMS, MIPS-based, or VAXELN realtime environments, and industrial terminals built for manufacturing environments. In addition, third parties offer realtime hardware options for use with Digital processors.

Digital's rtVAX products are VAX computers with VAXELN software, for dedicated realtime computing. The products include the rtVAX 300 daughterboard, rtVAXstation workstations, and rtVAX system platforms. See Chapter 24 for more information about the rtVAX realtime computers.

The DECelx Toolkit includes board support packages for the R3000 and Motorola architectures and SBCs for those architectures. For more information on DECelx hardware support, see Chapter 2.

Realtime I/O options include analog/digital converters, IEEE–488 instrument bus controllers, BITBUS controller, parallel I/O controllers, realtime clocks, I/O processor (IOP) boards, and industrial terminals. Busses supported for realtime I/O include the Q–bus, SCSI, TURBOchannel, and the VAXBI bus. For more information on Digital's realtime hardware options, see Chapter 25.

For more detailed hardware configuration information and descriptions of the third-party realtime options available for Digital systems, consult the *Systems and Options Catalog* for Realtime, UNIX, and VMS Systems.

24 Realtime Hardware Product Families

Realtime hardware products are designed to meet the demanding requirements of many factory, laboratory, and simulation activities, and to perform either as standalone computing solutions or as integral parts of corporate-wide, distributed networks. Digital's realtime products include VAX and MIPS-based components, workstations, systems, software, and networking tools to provide the complete environment you need for developing realtime solutions for your organization. Realtime products can be migrated to equipment and options with higher performance and functionality.

The realtime hardware product family offers chip-level processors (CLPs), single-board computers (SBCs), workstations, and system-level configurations of Digital's 32-bit VAX and MIPS-based computers, housed in many types of enclosures. The realtime hardware product family spans a wide range of processing power and can meet realtime computing needs on many levels. This chapter describes the following product families:

- Overview of Hardware Products for Realtime Systems, Section 24.1
- Realtime Chip-Level Processors, Section 24.2
- Realtime Single-Board Computers, Section 24.3
- Workstations for Realtime Systems, Section 24.4
- MicroVAX 3100 Realtime Workstations, Section 24.5
- rtVAX Realtime VAX Systems, Section 24.6
- VAX and MIPS-Based Systems, Section 24.7

24.1 Overview of Hardware Products for Realtime Systems

Digital's realtime products combine hardware and software products that are suitable, and in many cases designed specifically for realtime applications.

Both VAX and MIPS-based workstations and systems offer increased performance and expandability across a wide range of industry segments. When used with realtime software, such as VAXELN, DECelx, or the DEC OSF/1 realtime kernel, these workstations become powerful, low-cost realtime workstations for high-performance graphics applications. They combine workstation capability with realtime software for sophisticated realtime requirements, offering the X Window System-based interface, industry-standard networking, and a highly efficient runtime environment. These workstations are designed around industry-standard I/O busses and support communications options suited for distributed processing.

Digital's *rtVAX* products are dedicated realtime computers with VAXELN software that provide cost-effective solutions for meeting realtime computing needs. These products include the rtVAX 300, KAV30, rtVAX workstations, and rtVAX system-level computing platforms.

Realtime computing often requires less memory and storage than general-purpose computing. Digital's rtVAX products are VAX systems streamlined for these conditions, with the cost savings in hardware passed on to you. At the same time, Digital provides the potential for expansion; memory and disk space can be increased when customer needs grow. Digital offers many types of enclosures (rackmount, wallmount, tabletop, pedestal, and cabinet) to suit all types of environments.

24.2 Realtime Chip-Level Processors

Applications for chip-level processors (CLPs) realtime products can be found in all industries where highly specialized, dedicated, realtime processors and controllers are needed. Typical of these industries are discrete and continuous manufacturing processes, such as found in the automobile, chemical, and paper industries. Laboratory and scientific enterprises, medicine, aerospace, government defense agencies, utilities, and oil and gas industries also use chip-level processors in dedicated applications such as robotics, simulation, process control, and spacecraft control and tracking.

The chip-level members of the rtVAX product family, the rtVAX 300 combines a complementary metal oxide semiconductor (CMOS) MicroVAX processor with floating-point and Ethernet coprocessors on a small board. These processors can be included as a component in a design with industry-standard or proprietary I/O hardware. Chip-level processors are extremely flexible and can be adapted to virtually any application that can benefit from a fully supported network connection, an integrated realtime operating environment, and a processor.

Because the RAM and ROM memory sizes are selectable, the amount of memory incorporated into the realtime target is determined by the requirements of the application, further improving the cost/performance of realtime targets.

24.2.1 rtVAX 300

The rtVAX 300 realtime target, chip-level processor (CLP) is extremely flexible and can be adapted to virtually any application that can benefit from a fully supported network connection, an integrated realtime operating environment, and a 2.7 VUPs processor.

Features of the rtVAX 300

The rtVAX 300 board has these features:

- 2.7 VUPs CMOS VAX processor
- CMOS floating point coprocessor
- CMOS Ethernet coprocessor, for direct connection to Ethernet
- Integrated on a small (4.61-inch by 3.11-inch), embeddable daughterboard
- VAX architecture extended to the realtime component level
- Ability to interface to an industry-standard or proprietary I/O bus, potentially allowing a wide range of devices
- Ability to interface to industry-standard LSI/VLSI peripheral chips
- Memory size determined by the user, according to the application's minimum requirements (4 Mbytes to 16 Mbytes)
- Memory accesible from rtVAX, SCSI, and VMEbus
- Bootable from the network or external ROM
- VAXELN runtime target license included, with DECnet end-node and TCP/IP networking functionality (no additional networking license required)

These features allow development of realtime applications and in high-level languages using the powerful CASE tools of VMS and the VAXELN Toolkit.

The rtVAX 300 chip-level processor is a small subsystem that you can use as a component in your realtime system designs. Nearly every facet of the final system, from memory and I/O to power and packaging, is under your control. Unlike other rtVAX products that incorporate the Q22–bus or XMI bus architecture for I/O, the rtVAX 300 design lets you connect the processor to non-Digital, industry-standard busses and third-party devices. In addition, industry-standard Large Scale Integration (LSI) and Very Large Scale Integration (VLSI) components can be integrated into the VAX architecture, permitting use of highly specialized hardware coprocessors and controllers. VAXELN drivers for these devices can be written in high-level languages. A console terminal is supported by the rtVAX 300 with the addition of a Dual Universal Asynchronous Receiver/Transmitter (DUART) serial line chip.

Because the designer selects RAM and ROM sizes, the amount of memory incorporated into the realtime target is determined by the application's exact requirements. This further improves the cost/performance ratio for realtime targets built around the rtVAX 300.

In some cases, the hardware design based on the rtVAX 300 may require modification of the rtVAX 300 kernel executive provided in the VAXELN Toolkit. For example, you may need to modify the kernel to provide appropriate mapping for DMA transfers across a non-Digital I/O bus. In order to modify the rtVAX 300 kernel, you need materials available in the VAXELN Source Kit, which is sold separately from the VAXELN Toolkit. Source Kit materials include templates, command files, and documentation explaining how to tailor the rtVAX 300 kernel.

The rtVAX 300 can be applied in all industries that use dedicated realtime processors. Examples are manufacturing (both discrete and continuous processes), laboratory and scientific enterprises, medicine, aerospace, government defense agencies, utilities, and oil and gas industries. Applications within these industries include data acquisition, process control, robotics, simulation, command and control, and aircraft and spacecraft control and tracking.

24.3 Realtime Single-Board Computers

Single-board computers (SBCs) for realtime are built around realtime chip-level processors (CLPs). Generally, a single-board computer contains the following:

- One or more realtime processors
- A floating-point coprocessor
- An Ethernet coprocessor
- A SCSI controller
- Onboard main memory

Digital offers two board-level processors for realtime computing using VAXELN, the KA620 and the KAV30. The KA620 includes an optimized MicroVAX processor and a Q22–bus interface.

The KAV30 is a VMEbus-based VAX realtime single-board computer that runs VAXELN, dedicated realtime systems. Built around the rtVAX 300 realtime processor, the KAV30 contains a 2.7–VUP CMOS VAX processor, a CMOS floating-point coprocessor, Ethernet coprocessors, optional SCSI controller, and 4 or 16 Mbytes of main memory.

Features of the KAV30

The KAV30 has these features:

- Contains an rtVAX300 CPU daughter module
- Optional third-generation SCSI controller
- VMEbus Interface that conforms to ANSI/IEEE standard 1014-87, IEC821 and 297, support for AM codes, system controller functions, and VME interrupt handling
- VME Subsystem Bus (VSB) support
- Extra realtime devices; timers, watchdog timer (MAX696), battery backed up realtime clock (DP8570 CMOS RTC)
- VAXELN development tools on VAX/VMS including remote symbolic debugging
- Bootable from over an Ethernet, or booted from SCSI Disk or EPROM

With the KAV30, you develop applications on a VMS host system using the VAXELN Toolkit and high-level languages and runtime libraries such as Ada, VAX C, VAX FORTRAN, and VAXELN Pascal. The application can be built into a VAXELN runtime system, downline loaded into a KAV30 target via Ethernet, or booted from SCSI disk or EPROM.

These features provide an easy integration of VMEbus modules into the well accepted VAX architecture. The KAV30 allows users to integrate their VMEbus based applications into the computing and networking environment. Data can be communicated to and from any device on the network using DECnet or TCP/IP protocols, providing complete integration from the realtime device to the data center. The KAV30 complements Digital's family of rtVAX computers.

The VAX architecture, when combined with the powerful software development and runtime environment provided by the VMS host system and the VAXELN Toolkit, significantly reduces product development times. Applications, including device drivers, are developed in high level languages with compilers common between host and target. Extensive networking and integration capabilities are provided using DECnet and TCP/IP protocols.

24.4 Workstations for Realtime Systems

Industry standards determine how well a computer system works with other systems. Only through adherence to standards can you reap the benefits of open systems. Achieving open systems requires a broad suite of standards, including networking protocols, common user interface specifications, and standard programming interfaces.

In addition to adhering to standards, Digital continually improves workstation performance through new technology. Follow-on models for VAXstations running the VMS or VAXELN operating systems and DECstations running the ULTRIX operating system feature new, high-performance technology, improved graphical capabilities, support a wide range of networking and communication needs, and are designed to fit well as components in a distributed computing environment.

This section covers the VAXstation 4000 family (as the follow-on generation of the VAXstation 3100 family) and the DECstation 5000 family. The DECstation 5000 family of UNIX-based workstations, systems, and servers is based on RISC (Reduced Instruction Set Computer) technology.

This section describes the following:

- Workstation Overview, Section 24.4.1
- The VAXstation 4000 Family, Section 24.4.2
- The DECstation 5000 Family, Section 24.4.3

For additional sources of information on Digital's workstation products, see the appropriate *Systems and Options Catalog* and the *Software Product Description* for each product.

24.4.1 Workstation Overview

Digital offers a family of UNIX-based workstations, systems, and servers based on RISC (Reduced Instruction Set Computer) technology. The family includes DECstation 5000 workstations, DECsystem servers, and multi-user systems. Digital also offers a full range of software, networking products, PCs, terminals, and printers to complete an integrated computing system.

Digital's MIPS-based systems are available as single-user workstations, or workgroup or departmental systems and servers. From top to bottom, all run the ULTRIX operating system. The DECstation 5000 models 120 and 125 also currently support the DEC OSF/1 operating system. The versatility of Digital's

MIPS-based systems makes them perfect for a variety of tasks, including client/server computing, or high-performance 2D and 3D graphics.

Digital's DECstation and DECsystem products use the RISC chip technology from MIPS Computer Systems, Inc.

Digital's little-endian data format is the same format used in more than 80 million personal computers, ensuring easy data interchange between workstations, servers, and PCs.

The VAXstation 4000 workstations are Digital's newest VAX desktop systems and include the Model 60, 90, and VLC. They replace the VAXstation 3100 family, one of the most popular families of workstations in the industry, offering higher performance, lower cost, and increased flexibility and functionality.

The VAXstation 4000 family provides three times better price/performance than previous VAX workstations. The VAXstation 4000 Model 90 is Digital's highest performance desktop VAXstation — offering a price/performance similar to some MIPS-based systems.

Table 24–1 lists the members of Digital's DECstation 5000 and VAXstation 4000 product families.

Table 24–1 Workstation Products

Product	Performance	I/O Interface	Operating System
VAX 4000 VLC	6.2 SPECmarks[1]	SCSI	OpenVMS
			VAXELN
VAX 4000	12.0 SPECmarks	SCSI	OpenVMS
Model 60		optional TURBOchannel	VAXELN
VAX 4000	32.7 SPECmarks	SCSI	OpenVMS
Model 90		optional TURBOchannel	
DECstation 5000	16.3 SPECmarks	TURBOchannel	ULTRIX
Model 120		SCSI	DEC OSF/1
		optional VME	

[1]A SPECmark is the geometric mean of ten compute-intensive, public-domain benchmarks compared to the performance of a VAX–11/780.

(continued on next page)

Table 24–1 (Cont.) Workstation Products

Product	Performance	I/O Interface	Operating System
DECstation 5000	19.1 SPECmarks	TURBOchannel	ULTRIX
Model 125		SCSI	DEC OSF/1
		optional VME	
DECstation 5000	26.5 SPECmarks	TURBOchannel	ULTRIX
Model 133		SCSI	
		optional VME	
DECstation 5000	32.4 SPECmarks	TURBOchannel	ULTRIX
Model 240		SCSI	
		optional VME	
Personal DECstation	16.3 SPECmarks	TURBOchannel	ULTRIX
5000 Model 20		SCSI	
		optional VME	
Personal DECstation	19.1 SPECmarks	TURBOchannel	ULTRIX
5000 Model 25		SCSI	
		optional VME	
Personal DECstation	26.5 SPECmarks	TURBOchannel	ULTRIX
5000 Model 33		SCSI	
		optional VME	

Standards-compliant operating system software promotes application portability across multivendor networks.

Network Application Support enables systems from different vendors to work together. Support for graphics standards such as the X Window System and PEX provides interoperability and distributed graphics support. Support for industry standard communication protocols, such as MMS, provide for distributed communications.

24.4.2 The VAXstation 4000 Family

The VAXstation 4000 family features a new, high-performance 2-D graphics accelerator standard, giving performance similar to that of the SPX accelerator offered as an option with the VAXstation 3100 family. Graphics acceleration allows the CPU to offload the execution of graphics operations so that the CPU is available for computational operations.

The VAXstation 4000 family is configured for all communication and networking needs.

The VAXstation 4000 workstations are suited for both commercial and technical environments. The entry-level VAXstation 4000 VLC is ideal for commercial applications such as office automation, desktop publishing, insurance and banking, and factory automation.

The VAXstation 4000 family introduces a new, high-performance I/O architecture that uses Direct Memory Access (DMA) and synchronous SCSI support. This new hardware/software architecture increases SCSI throughput to three times that of the asynchronous SCSI interface supported on the VAXstation 3100 family. The new synchronous SCSI interface is backward compatible with the VAXstation 3100 workstations for easy migration to the new VAXstation family.

This section describes the following:

- The VAXstation 4000 VLC, Section 24.4.2.1
- The VAXstation 4000 Model 60, Section 24.4.2.2
- The VAXstation 4000 Model 90, Section 24.4.2.3

24.4.2.1 The VAXstation 4000 VLC

The VAXstation 4000 VLC offers attractive performance in a compact package at an extremely low price. The VLC is a viable alternative for VAX VMS users to the traditional terminal or PC, offering many advantages such as a windowing environment and more processing power on your desktop to increase your productivity. The VAXstation 4000 family protects your software and hardware investment and provides you with a path to future VAX VMS computing.

Features of the VAXstation 4000 VLC

The VAXstation 4000 VLC has these features:

- Enables connections to industry-standard disks, tapes, and peripherals
- Offers 5.5 SPECmarks (6 VUPS) of processing power using the SOC chip.

- Supports memory capacity from 8 to 24 Mbytes and an optional 121 Mbyte internal (RZ23L) disk
- Supports industry-standard SIMMs
- Supports synchronous SCSI speeds of up to 5 Mbytes per second
- Includes accelerated 2D graphics (8-plane color and 8-plane grayscale)
- Comes equipped with Ethernet, one DEC423 serial line, and one RS232D serial line with modem control
- Supports a number of grayscale and color monitors in a variety of sizes

Additional storage expansions options are available using the SCSI bus.

24.4.2.2 The VAXstation 4000 Model 60

The VAXstation 4000 Model 60 offers more expansion, flexibility, and capacity than the VAXstation 3100 family. Complete one-box systems are available and include the following:

- 852 Mbytes of internal mass storage (two RZ25 disks)
- A removable media device for up to 104 Mbytes of memory
- A choice of 5 monitors
- Both ThinWire and thickwire Ethernet
- Two asynchronous lines (one with modem control)

The VAXstation 4000 Model 60 also offers an optional single-width (50 Mbytes per second) TURBOchannel bus slot that allows you to configure your system with both Digital and third-party options.

The VAXstation 4000 Model 60 is designed for commercial environments and technical applications such as CASE, MCAD, geoprocess engineering, visualization, animation, and simulation.

Features of the VAXstation 4000 Model 60

The VAXstation 4000 Model 60 has these features:

- Offers 10.6 SPECmarks (12 VUPs)
- Supports memory capacity from 8 to 104 Mbytes
- Supports up to a total of seven SCSI devices, over 6.0 Gbytes of storage with an internal storage capacity of 852 Mbytes
- Supports industry-standard SIMMs
- Supports synchronous SCSI speeds of up to 5 Mbytes per second

- Includes accelerated 2D graphics (8-plane color, 4-plane grayscale, and dual-monitor 8-plane color)
- Supports a number of grayscale and color monitors in a variety of sizes
- Enables connections to industry-standard disks, tapes, and peripherals
- Comes equipped with Ethernet, one DEC423 serial line, and one RS232D serial line with modem control.

24.4.2.3 The VAXstation 4000 Model 90

The VAXstation 4000 Model 90 is a high-performance OpenVMS workstation using the same 14-nanosecond CMOS NVAX chip set as the VAX 6000 Model 600 datacenter system. The VAXstation 4000 Model 90 offers almost three times the performance of the VAXstation 4000 Model 60.

VAXstation 4000 Model 90 software licenses include the VMS operating system and Network Application Support — NAS 250 for VAXstations. Systems ordered with a disk will have the base operating system, VMS V5.5-2 and DECwindows Motif factory installed.

You can upgrade the VAXstation 4000 Model 60 to a Model 90. The basic upgrade kit consists of a new CPU board with eight memory slots. Graphics cards and monitors can be transferred from a Model 60 to an upgraded Model 90 system.

The VAXstation 4000 Model 90 includes the following:

- 852 Mbytes of internal mass storage
- A removable media device for up to 104 Mbytes of memory
- Both ThinWire and thickwire Ethernet
- Two asynchronous lines (one with modem control)
- OpenVMS operating system, including compliance with POSIX for application portability and compatibility with industry-standard software and applications
- Network Application Support (NAS) 250 for VAXstations

The VAXstation 4000 Model 90 also offers an optional single-width (50 Mbytes per second) TURBOchannel bus slot that allows you to configure your system with both Digital and third-party options.

The VAXstation 4000 Model 90 is designed for applications such as CAD/CAM /CAE, medical and other forms on imaging, econometrics, process control/CIM, mapping, geophysical analysis, scientific visualization, and provides a level of performance suited for the most computationally demanding applications.

Features of the VAXstation 4000 Model 90

The VAXstation 4000 Model 90 has these features:

- Offers 32.7 SPECmarks
- Supports memory capacity (SIMM technology) from 16 to 128 Mbytes (and 256 Kbytes of writeback cache)
- Supports up to a total of seven SCSI devices, over 8.0 Gbytes of storage with an internal storage capacity of 852 Mbytes
- Supports industry-standard graphics protocols including PHIGS and GKS
- Supports synchronous SCSI speeds of up to 5 Mbytes per second
- Includes accelerated 2D graphics (8-plane color and 8-plane grayscale)
- Hardware support for double buffering and Z-buffering
- Supports a number of grayscale and color monitors in a variety of sizes
- Enables connections to industry-standard disks, tapes, and peripherals
- Comes equipped with Ethernet, one DEC423 serial line, and one RS232D serial line with modem control.

The VAXstation 4000 Model 90 can be enhanced with SPXg and SPXgt high-resolution, 3D graphics accelerator options, enabling you to run graphics-intensive applications.

24.4.3 The DECstation 5000 Family

Digital's MIPS-based DECstation 5000 family includes a full range of UNIX-based desktop workstations. From top to bottom, all run the ULTRIX operating system. The DECstation 5000 models 120 and 125 also support the DEC OSF/1 operating system. The versatility of Digital's MIPS-based systems makes them perfect for a variety of tasks, including client/server computing, or high-performance 2D and 3D graphics.

Digital's little-endian data format is the same format used in more than 80 million personal computers, ensuring easy data interchange between workstations, servers, and PCs.

This section describes the following:

- The Personal DECstation 5000 Series, Section 24.4.3.1
- The DECstation 5000 Model 133, Section 24.4.3.2
- The DECstation 5000 Model 240, Section 24.4.3.3

All Digital MIPS-based systems support TCP/IP and OSI networking protocols. DECstation 5000 and DECsystem 5900 systems support high-performance FDDI networking, VMEbus, and TURBOchannel open I/O interconnect. Network management is simplified with Digital's DECmcc software.

Table 24–2 Graphics Options for All DECstation 5000 Systems

Option	Description
MX	High-resolution monochrome graphics, ideal for CASE and technical publishing applications
HX	Color and greyscale 8-plane graphics with a smart frame buffer for accelerated 2D graphics; ideal for entry-level design automation, financial modeling, and presentation graphics applications
PXG+	Double-buffered, high-resolution, 8- or 24-plane color; optional Z buffer; 44-MHz Intel i860 graphics accelerator for fast 2D and 3D geometric transformations; PixelStamp rendering chipset; ideal for CAD/CAE/CAM, molecular modeling, seismic interpretation, and true-color applications
PXG Turbo+	Double-buffered, high-resolution, 96-plane true color; Z buffer; 44-MHz Intel i860 geometry accelerator; PixelStamp chipset with two scan-converter chips operating in parallel; ideal for high-performance 3D modeling, visualization, and animation applications
TX	Graphics option specifically for DECvideo multimedia options; contains a 24-bit true color frame buffer and an independent 8-bit, color mapped frame buffer; the 24-bit frame buffer may display realtime video from the DECvideo/PIP card

The wide range of accelerated 2D and 3D graphical capabilities offered for DECstations make them particularly well suited for applications involving design automation (CAD, CAE, and CAM), molecular modeling, visual simulation, and realtime animation.

To protect against technological obsolescence, the entire DECstation 5000 family uses a CPU daughter card design. By placing the CPU on a removable daughter card, you can easily upgrade your system to higher performance CPUs in the future.

24.4.3.1 The Personal DECstation 5000 Series

The Personal DECstation 5000 Series workstations and servers are Digital's low-end RISC systems. They offer an excellent combination of performance and features at low entry prices. The Personal DECstation 5000 is available in three models, the Personal DECstation 5000 Model 20, 25, or 33. They differ only in the speed of the CPU subsystem, and provide an upgrade path by replacement of the CPU daughter card. All models will support an R4000 CPU daughter card at a future date.

All Personal DECstations include on-board 8 MB memory and 1024 x 768, 8-plane graphics. Additional memory may be added up to a maximum of 40 MB and graphics performance may be further enhanced with the addition of any TURBOchannel graphics option. The Personal DECstations are equipped with an integral SCSI bus, Ethernet port, single RS232 serial port, and two TURBOchannel expansion slots. A variety of options are available from Digital or third parties for the TURBOchannel I/O interconnect, providing multimedia capabilities, fiber-optic networking, and interfaces to other industry-standard buses such as VME. The Personal DECstation is designed to provide a high degree of expandability and flexibility unmatched by other entry-level workstations.

One of the most exciting trends in desktop applications is multimedia, the capability to incorporate other media, such as audio and video, into your applications. With the inclusion of voice-quality speaker and audio I/O (standard with each system), the Personal DECstation Series is designed to give users the ability to develop and execute multimedia applications. Full motion video is also available through the TURBOchannel-based DECvideo option.

Features of the Personal DECstation 5000 Series

The Personal DECstation 5000 Series Models 20, 25, and 33 have these features:

- Enable easy and economical CPU upgrade with the 3-inch by 5-inch removable CPU daughter card (R4000 daughter cards will be available for these systems at a future date.)
- Model 20: 16.3 SPECmark, 21.6 integer MIPS (Dhrystone), 2.4 MFLOPS (double precision), and a clock speed of 20 MHz
- Model 25: 19.1 SPECmark, 26.7 integer MIPS (Dhrystone), 2.8 MFLOPS (double precision), and a clock speed of 25 MHz
- Model 33: 26.5 SPECmark, 34.4 integer MIPS, 6.0 MFLOPS (double precision), and a clock speed of 33 MHz

- Use the R3000A/R3010 CPU/FPU chipset operating at 20 MHz (Model 20), 25 MHz (Model 25), and 33 MHz (Model 33)
- Support up to 40 Mbytes of 4-Mbit DRAM-based memory
- Support a maximum of 25.3 GB of combined internal and external storage
- Support 426 Mbytes internal storage and a 2.88 MB floppy drive
- Provide standard thick wire Ethernet support (two additional Ethernet cards may be added), ThinWire and twisted pair optional
- Provide high-performance, open TURBOchannel I/O interconnect (50 Mbytes per second) — complies with ANSI FDDI for interoperability in a multivendor environment. Two slots are available for options.
- Provide a 64-Kbyte data cache on the Models 20 and 25, and a 128-Kbyte data cache on the Model 33
- All provide a 64-Kbyte instruction cache
- Provide one integral SCSI controller; two more may be added as TURBOchannel options
- Each SCSI controller supports up to seven devices
- Support standard network facilities — TCP/IP, NFS, and DECnet
- Allow for development of realtime applications using powerful UNIX CASE tools

24.4.3.2 The DECstation 5000 Model 133

The DECstation 5000 Model 133 workstation is the mid-range of Digital's UNIX-based, desktop family. The DECstation 5000 Model 133 combines high-performance CPU, fast I/O bus, with a range of 2D and 3D graphics options.

The DECstation 5000/133 workstation is available as a complete workstation or as an upgrade for existing DECstation 5000 Model 120 or 125 workstations. The Model 133 has the memory expansion and internal disk storage capacity generally found in more expensive workstations. With the new HX graphics option, the Model 133 delivers excellent performance for any application requiring fast 2D drawing speeds, such as ECAD and MCAD.

Features of the DECstation 5000 Model 133

The DECstation 5000 Model 133 has these features:

- Enable easy and economical CPU upgrade with the 3-inch by 5-inch (8-cm by 13-cm) removable CPU daughter card (R4000 daughter cards will be available for these systems at a future date.)
- 26.5 SPECmark, 34.3 integer MIPS (Dhrystone), 6.0 MFLOPS (double precision), and a clock speed of 33 MHz
- Use the R3000A/R3010 CPU/FPU chipset operating at 33 MHz
- Support 16 to 128 Mbytes of 4-Mbit DRAM-based memory
- Support up to 32 Mbytes of 1-Mbit DRAM-based memory so you can use memory from your DECstation 2100 or 3100 workstations
- Support up to 25.3 Gbytes of maximum internal and external storage
- Support up to 852 Mbytes of internal storage
- Provide standard thick wire 802.3/Ethernet support (up to three additional Ethernet cards may be added); ThinWire and twisted-pair are optional.
- Provide high-performance, open TURBOchannel I/O bus (50 Mbytes per second) — complies with ANSI FDDI for interoperability in a multivendor environment; three slots are available for options
- Support a 128-Kbyte data cache, 64-Kbyte instruction cache
- Provide one integral SCSI controller; three more may be added as TURBOchannel options
- Each SCSI controller will support up to seven devices
- Support standard network facilities — TCP/IP, NFS, and DECnet
- Allow for development of realtime applications using the powerful UNIX CASE tools

24.4.3.3 The DECstation 5000 Model 240

The DECstation 5000 Model 240 workstation is Digital's highest performance MIPS-based workstation, designed for compute-intensive 2D and 3D graphics applications. The DECstation 5000 Model 240 offers an unprecedented combination of power and expandability plus a spectrum of accelerated 2D and 3D graphics. The DECstation 5000 Model 240 combines balanced performance, a high-performance CPU, and superior graphics capabilities in a compact desktop design.

Features of the DECstation 5000 Model 240

The DECstation 5000 Model 240 has these features:

- Offers 32.4 SPECmark and 43 integer MIPS (Dhrystone) performance
- Use the R3000/R3010 CPU/FPU chipset operating at 40 MHz
- Include TURBOchannel, an open, high-performance (100 Mbytes per second) I/O channel to provide three option slots for extensive expansion
- Provide one integral SCSI controller; three more may be added as TURBOchannel options
- Each SCSI controller will support up to seven devices
- Support up to 480 Mbytes of ECC (Error Correcting Code) memory
- Support up to 28.98 Gbytes of storage
- Provide a high performance, open TURBOchannel I/O bus (100 Mbytes per second) — complies with ANSI FDDI for interoperability in a multivendor environment
- Provide standard ThickWire 802.3/Ethernet support (up to three additional Ethernet cards may be added). ThinWire and twisted-pair are optional.

24.5 MicroVAX 3100 Realtime Workstations

The MicroVAX 3100 Models 30, 40, and 80 offer increased system performance and expandability across a wide range of industry segments. These models are designed around Digital's CMOS technology, use the industry-standard SCSI bus for storage, and support communications options suited for distributed processing. The simplified memory configurations allow for easy upgrades and expandability.

The MicroVAX 3100 Models 30 and 40 uses the same System-On-A-Chip (SOC) CPU used in the VAX 4000 Model 200. The Model 80 uses the same chip set used in the VAX 6000 Model 500. The mid-range MicroVAX 3100 models offer a competitive range of performance.

The enhanced communications options of the MicroVAX 3100 Models results in higher performance. These options can be purchased separately or together, offering greater scaleability, flexibility, and growth potential.

The MicroVAX 3100 Models are offered with industry-standard Single In-Line Memory Modules (SIMM), which are easily accessible for upgrades and maintenance.

Features of the MicroVAX 3100

The MicroVAX 3100 workstations have these features:

- Models 30 and 40 use the System-On-A-Chip (SOC) CPU which combines the CPU, FPU, clock, and 6 Kbytes of cache on one chip.
- The Model 80 CPU includes the CPU, FPU, clock, and 128 Kbytes of cache.
- VAX processor providing up to 5.0 VUPs (Model 30) or 10.0 VUPs (Model 80) of processing throughput
- Up to two high-speed, 64 Kbytes per second, Wide Area Network (WAN) connections for an extended choice of networking connectivity.
- 8 Mbytes of main memory, expandable up to 72 Mbytes (Model 80)
- Direct Memory Access (DMA) for faster I/O
- High-speed SCSI bus architecture
- Four asynchronous communication lines can be expanded up to 20 total (Model 80)
- BA42A and BA42B enclosures
- Comes with factory-installed VMS software.
- Optionally comes with factory-installed NAS 300 software.

All models support both asynchronous and synchronous communications. Optional VAX Wide Area Network (WAN) Device Drivers support various industry-standard communications protocols, including Digital Data Communications Message Protocol (DDCMP), High-Level Data Link Control (HDLC), and Synchronous Data Link Control (SDLC).

24.6 rtVAX Realtime VAX Systems

The rtVAX system-level platforms are specially packaged members of the VAX family, dedicated for use in realtime applications. rtVAX systems can be configured as system building blocks, with no peripheral I/O hardware, or with Ethernet, disk, and cartridge tape options. rtVAX systems include the rtVAX 3300, 3400, 4000 Model 200, 300, 500, and 600 which offer from 2.4 to 150 VAX units of processing (VUPs). The high-end rtVAX 6000 systems offer computing power from 2.8 to 72 VUPs. The rtVAX 9000 systems offer computing power up to 40 VUPS.

This section provides information about individual rtVAX system platforms or, where applicable, related groups of platforms:

- rtVAX 3300/3305/3400 Realtime Systems, Section 24.6.1

- rtVAX 4000 Family of Realtime Systems, Section 24.6.2
- rtVAX 6000 Family of Realtime Systems, Section 24.6.3
- rtVAX 9000 Model 110 and 310 Systems, Section 24.6.4

All rtVAX products come with a target license for VAXELN runtime operating software, which optimizes the dedicated realtime systems for predictability and fast response time. The VAXELN Toolkit, a set of tools for building efficient realtime applications, is available as a layered product running under the VMS operating system on a VAX development system. While a VMS system running the VAXELN Toolkit is required for development of VAXELN rtVAX applications, a VMS, ULTRIX, or VAXELN system connected by Ethernet can serve as the boot host in an rtVAX system configuration. (An application can also be booted from disk or tape.) The VAXELN Toolkit includes the DECwindows user interface, which is based on the industry standard X Window System. (For more information about the VAXELN host-target relationship, see Chapter 4; for more information on VAXELN DECwindows support, see Chapter 7.)

Because realtime computing frequently occurs within a whole enterprise's computer network, Digital's rtVAX products are built to participate in corporate networks. Digital's interconnects can tie together your realtime systems, link them to larger systems, and connect them to local area and wide area networks (LANs and WANs). As a result, realtime data can be accessed and reported from any location in the network that is designed to allow access. rtVAX systems can include Ethernet, DECnet, and TCP/IP networking support, for connection to the corporate network and distribution of applications and data throughout the enterprise. (For more information about VAXELN networking support, see Chapter 6.)

Table 24–3 lists the members of Digital's rtVAX product family.

Table 24–3 rtVAX Products

Product	VUPs[1]	I/O Bus	Enclosure Type
rtVAX 1000	0.9	Q22–bus	BA23 (pedestal or rackmount) H9642 (cabinet)
rtVAX 3300	2.7	Q22–bus	BA215 (pedestal or wallmount)
rtVAX 3305	2.7	Q22–bus	BA214 (rackmount)

[1]One VUP equals the performance of a VAX–11/780.

(continued on next page)

Table 24–3 (Cont.) rtVAX Products

Product	VUPs[1]	I/O Bus	Enclosure Type
rtVAX 3400	2.7	Q22–bus	BA212 (rackmount) BA213 (pedestal or rackmount)
rtVAX 4000 Model 200	5.0	Q22–bus	BA215 (pedestal or wallmount) BA430 (pedestal) BA431 (rackmount)
rtVAX 4000 Model 300	8.0	Q22–bus	BA440 (pedestal) BA441 (rackmount)
rtVAX 4000 Model 400	16.0	Q22–bus	BA440(pedestal) BA441 (rackmount)
rtVAX 4000 Model 500	24.0	Q22–bus	BA440(pedestal) BA441 (rackmount)
rtVAX 4000 Model 600	32.0 to 150	XMI	BA440(pedestal) BA441 (rackmount)
rtVAX 6000 Model 400	7 to 72	VAXBI/ XMI	VAX 6000 Series cabinet
rtVAX 6000 Model 500	13 to 72	VAXBI/ XMI	VAX 6000 Series cabinet
VAX 6000 Model 600 Systems	32 to 150	VAXBI/ XMI	VAX 6000 Series cabinet
rtVAX 9000 Systems	40	XMI/ VAXBI	VAX 9000 Series cabinet

[1]One VUP equals the performance of a VAX–11/780.

24.6.1 rtVAX 3300/3305/3400 Realtime Systems

The rtVAX 3300, rtVAX 3305, and rtVAX 3400 systems provide 2.7 times the processing throughput of the entry-level rtVAX 1000. These systems feature an integrated storage element (ISE) interface, Ethernet controller, and 4 Mbytes of ECC memory tightly integrated in a CMOS single-board processor implementation. In addition to maximizing mass-storage throughput, this allows the total bandwidth of the Q22–bus to be dedicated to application-specific I/O.

The rtVAX 3300, rtVAX 3305, and rtVAX 3400 systems allow for development of realtime applications using the powerful VMS CASE tools and the VAXELN Toolkit in high-level languages.

Features of the rtVAX 3300/3305/3400

The rtVAX 3300, 3305, and 3400 systems have these features:

- CMOS VAX processor, based on the KA640 processor board, providing up to 2.7 VUPs of processing throughput
- On-board 4 Mbytes of ECC memory, Ethernet controller, and Integrated Storage Element (ISE) interface
- Flexible system configurations
- Main memory expandable up to 52 Mbytes
- Mass storage expandable up to 300 Mbytes (rtVAX 3300) or 1.2 Gbytes (rtVAX 3400)
- Q22–bus architecture, allowing use of a wide range of I/O devices from Digital and third-party suppliers
- VAXELN runtime target license included, with DECnet end-node and TCP/IP networking functionality (no additional networking license required)

The rtVAX 3300, 3305, and 3400 systems have these enclosures:

- The rtVAX 3300 housing consists of a BA215 enclosure (pedestal or wallmount), with a six-slot Q22–bus backplane, a power supply, up to two optional 150-Mbyte RF30 disk drives, and a 296-Mbyte TK70 cartridge tape drive.
- The rtVAX 3305 housing is configured in the small, diskless rackmount BA214 enclosure. All of the Q22–bus options supported by the rtVAX 3300 and 3400 (except disk controllers) are supported in the rtVAX 3305. Memory is expandable up to 52 Mbytes with up to three MS650–BA memory units.
- The rtVAX 3400 housing is in the BA213 enclosure (pedestal or rackmount), with 12 Q22–bus slots, two power supplies, up to three 400-Mbyte RF71 disk drives, and a 296-Mbyte TK70 cartridge tape drive. The system is also available in a BA212 rackmount enclosure that requires only 14 inches of rack space.

The rtVAX 3300 system makes an excellent distributed computing platform, especially for process control, factory data collection, and machine automation. The rtVAX 3305 system is traditionally used for data acquisition, monitoring, analysis, testing, event control, simulation, and high-energy physics.

The rtVAX 3400 system is ideal for factory data collection, machine automation, and distributed process platforms.

24.6.2 rtVAX 4000 Family of Realtime Systems

The rtVAX 4000 Models 200, 300, 400, 500, and 600 are high-performance realtime computing platforms. The rtVAX 4000 Model 200 through Model 600 are ideal for office, laboratory, or shop floor environments, and for running such applications as flow process monitoring, flight simulation, power plant training simulation, and medical imaging and diagnosis.

The rtVAX 4000 Model 400, 500, and 600 are follow-on models from the rtVAX 4000 Model 200 and 300. The newer models offer increased performance, an advanced I/O subsystem, and high-availability. The older rtVAX 4000 models are field upgradable to the newer models.

The configuration of the rtVAX 4000 systems includes one or more integrated Digital Storage System Interconnect (DSSI) disk adapters and an integrated Ethernet adapter with direct memory access (DMA). The integrated system module allows memory, Ethernet, and disk functions to bypass the I/O bus (a Q22–bus), making the full Q22–bus bandwidth available for additional Digital and third-party I/O options.

The rtVAX 4000 Model 200 can be ordered in pedestal, wallmount, and rackmount configurations. The lower-cost small pedestal or wallmount configuration includes the processor module, 8 Mbytes of memory, power supply, four open Q22–bus slots, and four storage bays for DSSI mass-storage devices. The BA430 pedestal and BA431 rackmount configurations include the processor module, 16 Mbytes of memory, power supply, ten open Q22–bus slots, and four mass-storage device bays.

The rtVAX 4000 Model 300 is housed in a BA440 pedestal or BA441 rackmount enclosure, which includes the processor module, 32 Mbytes of memory, power supply, seven open Q22–bus slots, and four storage bays for DSSI mass-storage devices.

Features of the rtVAX 4000 Model 200 and 300

The rtVAX 4000 Model 200 and 300 systems have these features:

- CMOS VAX processor providing high-speed processing throughput up to 5.0 VUPs (Model 200) or 8.0 VUPs (Model 300)
- Main memory expandable up to 64 Mbytes (Model 200) or 128 Mbytes (Model 300)
- CMOS Ethernet coprocessor, for direct connection to Ethernet (1 or 2 network adapters may be added on)
- Mass storage expandable up to 21 Gbytes (Model 200) or 28 Gbytes (Model 300)

- DSSI bus adapter, supporting up to seven other DSSI devices, such as RF-series mass storage controllers (total configurable capacity of DSSI devices, 21 for Model 200, 28 for Model 300)
- Q22–bus architecture, allowing use of a wide range of I/O devices from Digital and third-party suppliers
- VAXELN runtime target license included, with DECnet end-node and TCP/IP networking functionality (no additional networking license required)
- Allows development of realtime applications using the powerful CASE tools of VMS and the VAXELN Toolkit and in high-level languages (Ada, C, FORTRAN, and Pascal)
- Optionally comes with factory-installed NAS 200, 300, or 400 software.

The key improvements of the more recent rtVAX 4000 models over the earlier ones pertain to performance, improvements in cache and memory subsystem, clock speed, additional communication or storage options, and improved chip design.

Features of the rtVAX 4000 Models 400, 500, and 600

The rtVAX 4000 Models 400, 500, and 600 systems have these features:

- Two CMOS-based VAX processors provide high-speed processing throughput up to 16 VUPs (Model 400), 24 VUPs (Model 500), or 150 VUPs (Model 600)
- Main memory expandable up to 512 Mbytes
- A third processor drives the Ethernet controller, for direct connection to Ethernet
- Mass storage expandable up to 56 Gbytes by using DSSI Integrated Storage Elements (ISEs)
- Clock speed for the Model 600 is 12 nanoseconds (versus 14 nanoseconds in earlier models)
- Two embedded DSSI bus adapters, supporting up to seven other DSSI devices. Two KFQSA adaptors can be added to the Q–bus backplane, making it possible to configure up to 28 DSSI devices.
- Q22–bus architecture, allowing use of a wide range of I/O devices from Digital and third-party suppliers
- Model 600 uses an XMI-based system interconnect, memory subsystems, I/O subsystems, networking, and communications technology.

- VAXELN runtime target license included, with DECnet end-node and TCP/IP networking functionality (no additional networking license required)
- Allows development of realtime applications using the powerful CASE tools of VMS and the VAXELN Toolkit and in high-level languages (Ada, C, FORTRAN, and Pascal)
- Optionally comes with factory-installed NAS 200, 300, or 400 software.

24.6.3 rtVAX 6000 Family of Realtime Systems

rtVAX 6000 systems, based on the VAX 6000 Series multiprocessors, provide an expandable set of midrange realtime systems in a single cabinet. Two sets of models comprise the most recent rtVAX 6000 Series, the rtVAX 6000 Model 400 systems, and the rtVAX 6000 Model 500 systems. These models are the follow-on products for the earlier rtVAX 6000 Models 200 and 300 and in the future, can be upgraded with just a board swap.

rtVAX 6000 systems are based on the XMI bus, while offering both XMI and VAXBI busses for I/O. The efficient multiprocessor architecture and multiple-bus design of rtVAX 6000 systems speed up system throughput. The VAXELN operating environment balances the load, ensuring that the extensive processor, memory, and I/O resources are shared among multiple processes.

The rtVAX 6000 Model 400 Series offers from one to six VAX processors. The model designations are Model 410, 420, 430, 440, 450, and 460, respectively. As with the Model 200 and 300 Series, models with two or more processors run under the VAXELN Toolkit's tightly coupled multiprocessing.

The rtVAX 6000 Model 500 Series offers from one to six VAX processors. The model designations are Model 510, 520, 530, 540, 550, and 560, respectively. As with the Model 200, 300, and 400 Series, models with two or more processors run under the VAXELN Toolkit's tightly coupled multiprocessing.

Any rtVAX 6000 system also can participate in a closely coupled multiprocessing configuration, in which one or more KA800 processor boards are connected to the system's I/O bus. Typically, the KA800 executes a VAXELN system image whose purpose is to offload I/O processing from the rtVAX 6000 system.

VAXELN applications can span multiple systems across an Ethernet local area network (LAN), thus allowing an application to use the computing resources of a number of realtime processors. Processes executing in parallel on separate systems can exchange information rapidly across the Ethernet using a VAXELN message-passing mechanism.

Features of the rtVAX 6000

The rtVAX 6000 systems include these features:

- From one to six CMOS VAX processors, providing high-speed processing throughput ranging from 2.8 VUPs (rtVAX 6000 Model 210) to 72 VUPs (rtVAX 6000 Model 500)
- Integral floating point
- Main memory expandable to a maximum determined by the type and number of processors (from 32 to 512 Mbytes), employing MS62 or MS65A memory
- Up to six Ethernet controllers, for direct connection to Ethernet
- 296-Mbyte TK70 cartridge tape drive
- Mass storage expandable up to 58.2 Gbytes
- Modular components and efficient multiprocessor architecture, allowing for easy, economical expansion within the same cabinet and rapid system upgrades
- XMI bus available for native-mode I/O, supporting devices such as the DEC LANcontroller 400 (DEMNA) Ethernet controller
- Up to six VAXBI busses for I/O, with 10 backplane slots per bus, providing fast aggregate and single-device I/O rates, up to a maximum I/O throughput rate of 60 Mbytes per second
- Choice of XBIA or XBIA+ VAXBI bus adapters
- VAXELN runtime target license included, with DECnet end-node and TCP/IP networking functionality (no additional networking license required)
- Allows development of realtime applications using the powerful CASE tools of VMS and the VAXELN Toolkit and in high-level languages (Ada, C, FORTRAN, and Pascal)

24.6.4 rtVAX 9000 Model 110 and 310 Systems

The rtVAX 9000 system combines supercomputing performance with a mainframe architecture to meet the needs of the high-performance realtime user. Two models comprise the rtVAX 9000 Series: the rtVAX 9000 Model 110; and the rtVAX 9000 Model 310. A unique crossbar switch design, the large capacity XMI bus structure and powerful CPU allow for truely balanced system performance needed for the most demanding realtime applications, such as space flight simulations, medical imaging, and strategic government and military programs. A powerful CPU gives the rtVAX 9000 the highest single-stream performance of any Digital realtime system.

The I/O subsystem uses the XMI bus to support the corporate Backplane Interconnect (BI), Network Interconnect (NI), and Digital Storage Architecture (DSA) interconnect architectures through the use of the DWMBB, DEMNA, and KDM70 respectively. Both the rtVAX 9000 Models 110 and 310 come with one XMI I/O bus standard, but the 310 can be expanded to two if more I/O capacity is needed.

VAXELN supports all non-vector, single processor VAX 9000 systems with 128 or 256 Mbytes of memory.

Features of the rtVAX 9000

The rtVAX 9000 systems include these features:

- Single processor provides 40 VUPs of high-speed throughput
- Integral floating point
- Main Cache of 128 KB with 8 KB virtual instruction cache
- Up to two XMI I/O buses support devices such as, DEMNA Ethernet adapters, KDM70 disk/tape controllers, and the VAXBI
- Includes one DEMNA Ethernet adapter, expandable to two
- Supports up to four KDM70 disk/tape controllers
- Supports up to four VAXBI channels
- Mass storage expandable to 48 Gbytes
- All systems include VAXELN runtime target license with DECnet end-node and TCP/IP networking functionality

24.7 VAX and MIPS-Based Systems

This section covers server systems for VAX and MIPS-based systems.

This section describes the following:

- The DECsystem 5900 Server, Section 24.7.1
- VAX 6000 Systems, Section 24.7.2

24.7.1 The DECsystem 5900 Server

The DECsystem 5900 is a UNIX-based RISC system supporting a wide range of networking and storage options. The system is housed in a 67-inch cabinet (19-inch rack-based) with five modular drawers for upgradability. At the same time it protects your investment in Digital's storage options.

The DECsystem 5900 is the ideal server for data-intensive applications, large UNIX-based workgroups, commercial applications needing high volumes of storage, database systems, and as a software development CASE platform. As a data server it offers superior backup, management, and a low cost per user. The high-performance I/O minimizes bottlenecks and provides maximum server productivity.

Features of the DECsystem 5900

The DECsystem 5900 has these features:

- Balanced performance through high-performance CPU, I/O, and memory
- Exceptional price/performance and application performance
- Includes Prestoserve, a high-performance file system accelerator, for use with applications using synchronous writes; improves overall file system performance by up to 300 percent; Prestoserve can be enabled and disabled
- 32.4 SPECmark rating, 42.9 integer MIPS (Dhrystone)
- Wide variety of peripheral support—SCSI, TURBOchannel, FDDI, VME, 802.3/Ethernet, CI Interface, and third party peripherals through the Digital TRI/ADD Program
- Provides superior CPU performance through the MIPS R3000A/R3010A CPU/FPU chipset operating at 40 MHz (upgradeable to a future R4000 CPU daughter card)
- 448 Mbytes ECC (Error Correcting Code) memory
- Maximum internal storage capacity—to 35 Gbytes (SCSI)
- 128 Kbytes system cache
- Network support—TCP/IP and NFS standard; DECnet–ULTRIX optional

DECsystem servers have superior productivity because every system balances CPU power, high-speed memory, and high I/O to maximize server capabilities. The Prestoserve accelerator delivers a significant performance increase in an NFS environment.

24.7.2 VAX 6000 Systems

The focus of the VAX 6000 systems is to provide datacenter solutions across a wide range of applications.

VAX 6000 systems provide the superior, cost-effective, nondisruptive growth required by datacenter customers. With the VAX 6000 Model 600, you can add more processors, memory modules, XMI-based I/O adapters, optional VAXBI connections, and battery backup for availability.

Any VAX 6000 system can be upgraded to any higher VAX 6000 system through simple board swaps. VAX 6000 Model 600 is the follow-on product to VAX 6000 Models 520 through 560. However, the vector processor option is not available on the VAX 6000 Model 600 systems. Customers with the VAX–11/7xx and most VAX 8xxx series systems can also upgrade to the new VAX 6000 Model 600.

Features of the VAX 6000 Model 600

The VAX 6000 Model 600 includes these features:

- Provides up to 150 VUPS
- Based on multiple high-speed DSSI channels, up to 96 Gbytes of RF73 local storage can be supported
- Maximum physical storage 1,024 Mbytes
- Up to 83 transactions per second
- The optional XMI-to-VME transport system widens the choice of I/O expansion options
- VAX 6000 cabinet can house TF85 subsystem and up to 8 RF73 drives
- Open access for a wide range of I/O expansion options
- Volume shadowing technology provides site redundancy in a multi-datacenter facility
- NAS 200, 300, and 400 deliver NAS through focused packages

25 Realtime Options

This chapter presents realtime options available from Digital for the following bus architectures:

- Q22–bus, Section 25.1
- SCSI Bus, Section 25.2
- TURBOchannel Bus, Section 25.3
- VAXBI Bus, Section 25.4
- VMEbus, Section 25.5

In addition, two series of industrialized terminals offered by Digital are described in Section 25.6. These terminals can be used in realtime industrial settings.

For more detailed information and configuration guidelines, refer to the *Digital Systems and Options Catalog for Realtime* and the *Digital Systems and Options Catalog*.

25.1 Q22–bus Options

This section describes twelve Q22–bus options in alphabetical order:

- AAV11–C Digital-to-Analog Converter Module, Section 25.1.1
- AAV11–DA Digital-to-Analog Converter Module, Section 25.1.2
- ADQ32 Analog-to-Digital Converter Module, Section 25.1.3
- ADV11–C Analog-to-Digital Converter Module, Section 25.1.4
- ADV11–DA Analog-to-Digital Converter Module, Section 25.1.5
- AXV11 Analog Input/Output Module, Section 25.1.6
- DECscan Products for BITBUS Interconnection, Section 25.1.7
- DRQ3B Parallel-Line Interface Module, Section 25.1.8

- DRV1J and DRV11J Parallel-Line Interface Modules, Section 25.1.9
- DRV11–WA and DRV1W Parallel-Line Interface Modules, Section 25.1.10
- IEQ11–AB,AD,AF IEC/IEEE–488 Bus Interface Modules, Section 25.1.11
- KWV11–C Programmable Realtime Clock Module, Section 25.1.12
- Universal Data Interface Panels, Section 25.1.13

25.1.1 AAV11–C Digital-to-Analog Converter Module

The AAV11–C is a dual-height, multichannel, analog output module designed to interface analog instrumentation to Q22–bus-based MicroPDP–11 or MicroVAX systems.

The AAV11 is supported by DEC Realtime Integrator.

AAV11–C Digital-to-Analog Converter Module Features

The AAV11–C module includes:

- Four individually addressable digital-to-analog converter circuits
- 12-bit digital resolution
- Read/write, word, or byte-addressable registers
- Jumpers to permit selection of analog output voltage range for each register and its operating mode (either unipolar or bipolar)
- Output voltage range selection ±10 V or 0 to 10 V
- 4-bit digital output for cathode-ray tube (CRT) to control signal intensity, blank, unblank, and erase

AAV11–C Digital-to-Analog Converter Module Use

The AAV11–C module is used in applications requiring a computer-generated signal in the ±10 V range, such as acoustics, process control, and flight simulation.

25.1.2 AAV11–DA Digital-to-Analog Converter Module

The AAV11–DA is a high-speed, direct memory access (DMA) digital-to-analog converter designed to control experiments and analog instrumentation with the MicroVAX II, VAXstation II, or any Q22–bus-based Digital system.

The AAV11 is supported by DEC Realtime Integrator.

AAV11–DA Digital-to-Analog Converter Module Features

The AAV11–DA module features:

- High-performance, DMA, analog output system with support for 22-bit memory addressing and four-level interrupts
- Two independent, high-speed, 12-bit, digital-to-analog channels with Z-axis control and ranges of ±10 V, ±5 V, and 0–10 V
- Data conversion with 12-bit resolution under programmed I/O or DMA mode
- Two data conversion start schemes, two channel selection modes, and two data conversion modes
- Four lines of digital output
- Continuous (no gap) output from disk, software drivers, and subroutines available
- High aggregate data output rate in DMA mode of up to 200,000 values per second in single-channel mode or 300,000 values per second in dual-channel mode

AAV11–DA Digital-to-Analog Converter Module Use

The AAV11 module is used in realtime applications requiring high-speed signal output or high-speed instrument control in the 10 V range, such as aerospace research and testing, materials testing, structural analysis, speech processing, physiological monitoring, and electronic testing.

25.1.3 ADQ32 Analog-to-Digital Converter Module

The ADQ32 is a high-speed, direct memory access (DMA) analog-to-digital converter for Q22–bus systems.

ADQ32 Analog-to-Digital Converter Module Features

The ADQ32 module features:

- 32 single-ended or 16 differential-input channels of input data converted to 12-bit digital data
- Maximum (single-channel) throughput of 200 kHz
- Programmable gains of 1, 2, 4, and 8
- Programmable, single-ended, or differential operation per conversion
- Vernier digital-to-analog converters for programmable calibration of the analog-to-digital converters
- On-board, 512-word FIFO (first in/first out) to use the bus more effectively

- On-board parameter list for random channel, gain, and single-ended or differential-input selection
- On-board realtime clock that can provide a wide range of triggering modes
- Switch-selectable, four-level interrupts
- Pre- and post-stimulus triggering
- Continuous data collection through block mode DMA engine with dual channels and buffer chaining

ADQ32 Analog-to-Digital Converter Module Use

The ADQ32 module provides a range of sampling and triggering mechanisms to meet the requirements of most research and laboratory applications. This device can be used to interface transducers and laboratory instruments to the computer. Designed for the efficient transfer of large blocks of data at high speeds, the ADQ32 can be used in speech processing, electronic and material testing, aerospace research, physiological monitoring, and vibrational analysis.

The ADQ32 is supported by DEC Realtime Integrator.

25.1.4 ADV11–C Analog-to-Digital Converter Module

The ADV11–C is a dual-height, analog-to-digital converter for Q22–bus-compatible MicroPDP–11 and MicroVAX systems.

The ADV11 (without continuous DMA) is supported by DEC Realtime Integrator. The ADV11 (continuous DMA) is supported by DEC Realtime Integrator, but only through the LIO subroutine library calls.

ADV11–C Analog-to-Digital Converter Module Features

The ADV11–C module features:

- 16 single-ended or 8 differential-input jumper-selectable analog input channels
- 12-bit digital resolution
- Maximum throughput of 25,000 samples per second
- Software-programmable gain amplifier with gains of 1, 2, 4, and 8
- Initiation of data conversion by program start, realtime clock input, or external trigger input
- Polled or interrupt-driven I/O

ADV11–C Analog-to-Digital Converter Module Use

The ADV11–C can be used in applications involving data acquisition of analog signals in the range of 10 V and for interfacing analytical instruments such as gas chromatographs to Q22–bus computers.

25.1.5 ADV11–DA Analog-to-Digital Converter Module

The ADV11–DA is a high-speed, direct memory access (DMA) analog-to-digital converter designed to interface instrumentation to any Q22–bus-based MicroPDP–11 and MicroVAX system.

The ADV11 (without continuous DMA) is supported by DEC Realtime Integrator. The ADV11 (continuous DMA) is supported by DEC Realtime Integrator, but only through the LIO subroutine library calls.

ADV11–DA Analog-to-Digital Converter Module Features

The ADV11–DA module features:

- 16 single-ended or 8 differential-input jumper-selectable input channels
- 12-bit digital resolution
- 50-kHz analog-to-digital converter
- Support for 22-bit memory addressing
- Support for four-level interrupts
- Data transfer by programmed I/O or DMA
- Two data-acquisition start schemes:
  - Software start
  - Realtime clock start
- Two channel-selection schemes:
  - Single channel
  - Automatic increment from specified start channel
- Two types of data conversions per start:
  - Single conversion per start
  - Multiple conversion per start

ADV11–DA Analog-to-Digital Converter Module Use

The ADV11 module can be used in realtime applications involving high-speed instrument interfacing or signal acquisition. Applications that require transferring large blocks of data at high speed will benefit from the device's DMA capabilities, in such fields as aerospace research and testing, structural analysis, speech processing, and electronics testing.

25.1.6 AXV11 Analog Input/Output Module

The AXV11 is a low-cost, high-performance, Q22–bus-compatible analog input/output module for MicroPDP–11 or MicroVAX systems.

The AXV11 is supported by DEC Realtime Integrator, but only through the LIO subroutine library calls.

AXV11 Analog Input/Output Module Features

The AXV11 module has these features:

- 16 single-ended or 8 differential-input channels
- 12-bit digital resolution
- 25-kHz analog-to-digital converter, interrupt driven
- Two-channel, 12-bit, digital-to-analog converter, set under program control
- Software-programmable gains of 1, 2, 4, and 8
- Analog-to-digital conversions can be started by:
  - Software program
  - External trigger
  - Realtime clock
- Analog-to-digital conversion results can be received by:
  - Programmed I/O transfer
  - Servicing an interrupt request
- Output data notation in binary, offset binary, or two's complement format

AXV11 Analog Input/Output Module Use

The AXV11 can be used for data acquisition and output of analog signals in the range of 10 V, in applications such as acoustics, process control, and flight simulation.

The AXV11 can also interface analytical instruments such as gas chromatographs to Q22–bus systems.

25.1.7 DECscan Products for BITBUS Interconnection

Digital's DECscan products provide realtime access to data from I/O devices on the factory floor. Using Intel Corporation's BITBUS interconnect and application software running on a Q22–bus-based VAX system, the DECscan products allow communication to any BITBUS-compatible measurement and control devices. The DECscan interconnect products suit a wide variety of monitoring, control, and data-acquisition applications in many industries.

DECscan links measurement and control devices to a master control system from as many as 250 slave nodes interconnected over a physically distributed domain ranging from 30 meters to thousands of meters. The BITBUS uses twisted-pair cable to send and receive data based on the RS-485 specification. The communications protocol is a modified synchronous datalink control (SDLC) protocol.

DECscan Product Features

The DECscan BITBUS interconnect products:

- Connect realtime interfaces and devices to Digital computers
- Provide a realtime data management subsystem
- Provide an open architecture that allows for system and application expansion as needs grow
- Allow for scanning of I/O data for status information and responding in realtime to alarm conditions, current I/O values, and external events

DECscan Product Use

The open architecture of the DECscan products and BITBUS let you tie together I/O equipment from multiple vendors and connect this equipment to a single MicroVAX or VAXstation host system. This flexibility makes DECscan products ideal for many applications in both process and discrete manufacturing.

You can set up monitoring and control activity (without any coding) in applications such as factory floor monitoring and automation, product handling, environmental control, and remote, high-speed data acquisition and control.

The DECscan products have three components, which are described in the sections that follow:

- IBQ01 DECscan BITBUS Controller Module, Section 25.1.7.1
- QA–VCJAA DECscan Driver Package, Section 25.1.7.2
- QA–VCSAA VAX DECscan VMS Toolkit, Section 25.1.7.3

25.1.7.1 IBQ01 DECscan BITBUS Controller Module

The IBQ01 DECscan BITBUS controller provides the hardware and firmware support for a MicroVAX or VAXstation Q22–bus-to-BITBUS interface. The IBQ01 module is the master controller in a BITBUS system. It handles all message transactions between the supervisory host system and the industrial process I/O devices that are distributed along a BITBUS.

Operational control of slave nodes is based on a user-generated program that can be written in any VMS-supported language such as FORTRAN, Macro, or Pascal.

The IBQ01 controller supports transmission rates of:

- 2.4 Mbits per second (at 30 meters)
- 375 Kbits per second (at 900 meters)
- 62.5 Kbits per second (at 13,200 meters)

25.1.7.2 QA–VCJAA DECscan Driver Package

QA–VCJAA, the DECscan Driver Package, contains a standard VMS and VAXELN driver interface to BITBUS compatible I/O devices.

25.1.7.3 QA–VCSAA VAX DECscan VMS Toolkit

QA–VCSAA, the VAX DECscan Toolkit:

- Allows for monitor and control of BITBUS I/O through simple menu interaction
- Provides callable runtime routines for developing DECscan VMS industrial applications
- Allows for data acquisition
- Allows for alarm detection with controlling actions and reporting
- Allows for action sets (sequence control)
- Allows for closed loop regulatory control
- Has I/O vendor routines to support both Honeywell Micro Switch and Phoenix Contact BITBUS I/O devices
- Allows for generating and dynamically adding more I/O vendor support
- Includes a graphics interface between the DECscan Toolkit realtime I/O database and the DataViews (DV-Draw and DV-Tools) industrial graphics package sold by V.I. Corporation, Northhampton, MA

25.1.8 DRQ3B Parallel-Line Interface Module

The DRQ3B is a high-performance, direct memory access (DMA), parallel, Q22–bus I/O interface designed for realtime data collection or for high-speed interprocessor communications between MicroVAX II systems.

The DRQ3B is supported by DEC Realtime Integrator.

DRQ3B Parallel-Line Interface Module Features

The DRQ3B interface module features:

- Quad-height module
- Two independent, unidirectional I/O channels
- 16-bit parallel data transfers (in and out)
- Single-cycle burst-mode and block-mode DMA or program-controlled transfers
- 512-word FIFO (first in/first out) buffer on each channel
- 2.6 Mbytes per second maximum transfer rate
- Continuous data transfer capability
- Buffer chaining

DRQ3B Parallel-Line Interface Module Use

Typical applications of the DRQ3B interface module include high-speed graphics display and imaging, data telemetry, simulation, structural and performance testing, and seismic data collection.

25.1.9 DRV1J and DRV11J Parallel-Line Interface Modules

The DRV1J and DRV11J are general-purpose, 16-bit, parallel, programmable I/O interfaces for realtime data collection.

DRV1J and DRV11J Parallel-Line Interface Modules Features

The DRV1J and DRV11J interface modules feature:

- Four 3-state 16-bit parallel ports
- User-assigned device addresses
- Accepts up to 16 external interrupt requests
- Programmable interrupt vector addresses
- Program-controlled I/O operations
- Programable operating modes (interrupt controller, priority, and vector address modes)

- Can drive up to 25 feet (7.6 meters) of shielded cable and 6 feet (1.8 meters) of unshielded flat or round cable

DRV1J and DRV11J Parallel-Line Interface Modules Use

The DRV1J and DRV11J interface modules can accept up to 16 external interrupt requests, making them ideal in sensor I/O applications.

25.1.10 DRV11–WA and DRV1W Parallel-Line Interface Modules

The DRV11–WA and DRV1W are general-purpose, 16-bit, parallel, direct memory access (DMA) interface units for Q22–bus systems.

The DRV11–W is supported by DEC Realtime Integrator, but only through the LIO subroutine library calls.

DRV11–WA and DRV1W Parallel-Line Interface Modules Features

The DRV11–WA and DRV1W interface modules feature:

- Dual-height module (DRV11–WA) or quad-height module (DRV1W)
- Backward-compatible with DRV11–B
- 16-bit, bidirectional, parallel I/O port
- Switch-selectable 18- or 22-bit addressing mode
- Switch-selectable device and vector addresses
- Burst mode, byte addressing, and read-modify-write operation support
- Data transfer rates up to 500 KB per second in single-cycle mode and up to 800 KB per second in burst mode

DRV11–WA and DRV1W DMA Parallel-Line Interface Modules Use

The DRV11–WA and DRV1W interface modules are useful for connecting to high-speed user devices or an interprocessor communications link between Q22–bus, VAX, or PDP–11 systems. In typical applications, the DRV11–WA and DRV1W interface with such hardware as high-speed graphics terminals, data acquisition devices, and machine tool controls to the host processor.

25.1.11 IEQ11–AB,AD,AF IEC/IEEE–488 Bus Interface Modules

The IEQ11 is a direct memory access (DMA) controller that interfaces a Q22–bus system and two independent instrument busses. The instrument busses conform to both the European Standard IEC 625–1 and the United States IEEE–488 Standard (1978).

The IEQ11 is supported by DEC Realtime Integrator.

IEQ11–AB,AD,AF IEC/IEEE–488 Bus Interface Modules Features

The IEQ11 interface module features:

- One quad module to support two independent General-Purpose Interface Busses (GPIBs)
- 100-kHz DMA transfer rate available for each GPIB
- Full IEEE–488 Standard (1978) interface support
- 14 devices supported plus the controller itself
- End or identify (EOI) byte count or matched characters data transfer termination
- Ability to serve as a device interface between a MicroPDP–11 or MicroVAX system and microprocessor-controlled instrumentation

IEQ11–AB,AD,AF IEC/IEEE–488 Bus Interface Modules Use

The IEQ11 module provides an interface to laboratory instrumentation devices, such as programmable power supplies, signal generators, digital multimeters, function generators, and spectrum analyzers.

25.1.12 KWV11–C Programmable Realtime Clock Module

The KWV11–C is a programmable realtime clock module for Q22–bus-compatible MicroPDP–11 and MicroVAX systems.

KWV11–C Programmable Realtime Clock Module Features

The KWV11–C clock module includes:

- Five internal base frequencies: 1 MHz, 100 kHz, 10 kHz, 1 kHz, and 100 Hz
- 16-bit clock divider
- 1 microsecond minimum time base
- Four operational modes
- Two Schmitt triggers (level and edge adjustable)
- Line frequency input from BEVNT bus signal (50/60 Hz)
- Ability to trigger the ADV11, the AAV11, and the AXV11–*xx*

KWV11–C Programmable Realtime Clock Module Use

Typical uses of the KWV11–C clock module include measuring time intervals, measuring frequency, counting events, triggering analog data collection, triggering analog-to-digital or digital-to-analog converter modules, and interrupting the CPU at predetermined intervals.

25.1.13 Universal Data Interface Panels

Universal data interface panels (UDIPs) are special termination panel connections to laboratory instruments and transducers.

Each analog-to-digital, digital-to-analog, and realtime clock option has its own special UDIP. The options mount into the UDIP–BA box, which holds three single-width UDIPs or one single-width and one double-width UDIP. The UDIP–BA can be mounted in three ways:

- Inside either or both of the top two disk bays of a BA123
- Inside the UDIP–TA tabletop expansion box
- In a pair (two UDIP–BAs) inside the UDIP–RA rackmount expansion box

25.2 SCSI Bus Options

For most of its VAXstation, DECstation, and DECsystem products, Digital offers SCSI (Small Computer Systems Interface) disks and tapes. Digital's RZ disk drive interface products conform to the SCSI-2 industry-standard specification, which allows for high performance, high capacity, small form factor, and fast, simple interface features.

A SCSI bus can interface with up to eight SCSI devices, seven of which can be target devices.

25.2.1 IEZ11 SCSI-to-IEEE–488 Bus Converter Module

The IEZ11 Small Computer System Interface (SCSI) IEEE–488 bus converter provides IEEE–488 General-Purpose Interface Bus (GPIB) monitoring and control capabilities to VAXstation 3100 and MicroVAX 3100 systems. This converter allows bidirectional communication between system SCSI ports and IEEE–488 devices.

The IEZ11 bus converter is well suited for electronic test and measurement applications, where low cost and high performance are critical selection criteria.

DEC Realtime Integrator supports the IEZ11. DEC Realtime Integrator manages all SCSI and IEEE–488 bus interactions, allowing the user to communicate directly with the instruments. DEC Realtime Integrator for ULTRIX includes the software driver and icon support for this option.

IEZ11 SCSI-to-IEEE–488 Bus Converter Module Features

The IEZ11 converter module:

- Supports up to 14 IEEE–488 instruments per IEZ11 module
- Achieves a maximum data throughput rate of over 500 KB per second
- Provides all major IEEE–488 interface functions specified in the IEEE–488.1 Standard
- Meets the ANSI X3T9.2 (SCSI-2) Standard
- Can connect up to four IEZ11 converter modules to a VAXstation 3100 or MicroVAX 3100 system, thus providing an interface to communication networks
- Can modify IEQ11 application programs for use with the IEZ11 converter
- Consists of a SCSI–to–IEEE–488 converter module in a tabletop or rackmount enclosure
- Is packaged with this software:
  - VMS system-based SCSI-class driver
  - Installation, test, and demonstration software (ITADS)
- Is supported by the DEC Realtime Integrator for VMS and ULTRIX

IEZ11 SCSI-to-IEEE–488 Bus Converter Module Use

The IEZ11 converter is used to monitor and control standard test instruments, such as signal generators, voltmeters, programmable power supplies, transient recorders, and network analyzers.

Also, the IEZ11 converter can be connected to IEEE–488-compatible peripherals, such as plotters, printers, video terminals, disks, and tapes.

25.3 TURBOchannel Bus Options

TURBOchannel is an open, high-performance I/O interconnect designed by Digital for desktop computers and servers. The TURBOchannel architecture departs from previous bus architectures in that the TURBOchannel control signals have a radial point-to-point topology. A TURBOchannel-based system provides separate control lines for each of several peripheral slots, resulting in high-bandwidth DMA block transfer, low-latency transactions, and simplified option module design.

TURBOchannel is a synchronous channel, transferring one 32-bit word of data or protocol overhead in each cycle. TURBOchannel protocol supports either DMA or I/O transactions. With DMA transactions, data is transferred directly into buffers and system memory in block sizes determined by the option, up to a system-defined maximum.

TURBOchannel I/O transactions transfer single words of data between CPU registers and memory locations and may be used for data transfers by options lacking DMA buffers and logic.

Digital made the TURBOchannel specifications available outside of Digital to encourage the development of additional high-performance options for the DECsystem and DECstation computers.

Today, more than 200 vendors offer, or are developing, TURBOchannel options. TURBOchannel adapters, such as the TURBOchannel-to-VME adapter let you connect to other interconnect devices.

This section features TURBOchannel adapters offered by Digital that can be used in realtime industrial settings:

- DWTVX and DWTVA TURBOchannel-to-VME Adapters, Section 25.3.1
- IET11 TURBOchannel-to-IEEE–488 Adapter, Section 25.3.2

25.3.1 DWTVX and DWTVA TURBOchannel-to-VME Adapters

TURBOchannel-to-VME adapter options provide the ability to interface various standard and specialized independently supplied VMEbus (Versa Module Eurocard) devices to the VAXstation 4000 Models 60 and 90, DECstation, DECsystem 5000 Series, and DECsystem 5900 systems. The TURBOchannel-to_VME adapter makes systems suitable for a wider range of applications. TURBOchannel-to-VME adapters conforms to ANSI IEEE 1014–1987 VMEbus specifications and are compatible with over 4000 VMEbus compliant products.

TURBOchannel-to-VME adapters allow customers to expand the DECstation 500 Series with a variety of standard and specialized peripheral devices, which makes the DECStation 5000 Series suitable for a wider range of applications.

The adapter consists of a two-board set (DWTVX) and optionally an expansion card cage (DWTVA). One board is mounted in a single TURBOchannel slot; the other requires a 6Ux160mm VMS slot in a VME card cage. These modules are interconnected by a cable supplied with the modules. The software supported is included with ULTRIX Version 4.2 (or later) or DEC OSF/1. Byte ordering is accommodated via programmable hardware circuitry to minimize impact on performance.

A bus is open to the extent that third parties claim it is open and commit to port their products to it. Digital is actively working with VMEbus third party companies to port their products to Digital systems. Industry standard third-party option support is available. Also, Digital's TRI/ADD Program provides technical and marketing support worldwide to third-party vendors using the SCSI, TURBOchannel, VME, ACCESSbus, and Futurebus+ interconnects to develop products on open systems.

DWTVX and DWTVA TURBOchannel-to-VME Adapter Features

The TURBOchannel-to-VME adapter features:

- Programmable hardware
- Block Mode DMA Read of 21.0 Mbytes per second and DMA Write of 17 Mbytes per second
- Number of mapped pages in VME address space is 892
- Pagesize of 4096
- Total of 3.5 Mbytes
- Conformance to ANSI IEEE 1014–1987 VMEbus specifications
- Compatibility with over 4000 VMEbus-compliant products
- Available as a standalone module set or with VME expansion chassis
- Industry-standard, third-party option support

The TURBOchannel-to-VME option addresses the needs of technical OEMs, Government prime contractors, aerospace companies, and others needing specialized interface hardware to VAX and MIPS-based systems running applications such as realtime, data acquisition, process control, and image-processing applications. TURBOchannel expansion slots allow use of 3D, true-color, and high-resolution graphics, live video, specialized networking options (including FDDI networks), serial line expansion, and I/O options.

25.3.2 IET11 TURBOchannel-to-IEEE–488 Adapter

The IET11 is a combined hardware/software solution that includes the TURBOchannel-to-IEEE–488 bus converter, ULTRIX-based driver software and documentation. The software consists of a loadable multitasking device driver and programs, which allows a DECstation 5000 workstation to act as an IEEE–488 controller with complete communication and management capabilities. In addition, the device driver is LIO-compliant; that is, the IET11 is supported by the latest version of the DEC Realtime Integrator software platform. The software runs under ULTRIX, version 4.2 and compatability will be maintained with future ULTRIX releases.

The IET11 TURBOchannel-to-IEEE–488 converter provides a high-performance TURBOchannel I/O connection with the IEEE–488 bus for realtime computing. The IET11 provides IEEE–488 bus capabilities for the DECstation 5000 series workstations. It connects instruments and measuring devices communicating via the IEEE–488 bus to the TURBOchannel, Digital's integral I/O interconnect for workstations.

The IET11 converter allows standardized, bi-directional communication between the TURBOchannel and the widely used IEEE–488 bus, also known as GPIB bus or IEC bus. It features all major IEEE–488 interface functions specified in the IEEE–488.1 and IEEE–488 488.2 standard, and ensures data transfer rates of 1 Mbytes per second for both read and write operations by providing enhanced DMA transfer capability.

A DECstation 5000 workstation equipped with the IET11 converter is well suited for instrument control applications where low-cost and high-performance requirements are important selection criteria. These applications can be found in research and development, service laboratories, education and government. Typical applications include data acquisition, automatic test equipment, and numerous graphic applications including digitizing analog video signals for CAD/CAM, high-speed laser printers, and color scanners.

The IET11 provides full IEEE–488 interface functionality including, controller, talker and extended talker, listener and extended listener, service request, parallel poll, etc. It features high-speed data transfers with a sustainable transfer data rate capability of 1 Mbyte per second for both read and write operations.

DEC Realtime Integrator for ULTRIX supports the IET11. The ULTRIX device driver for the IET11 is only sold with the IET11 module. DEC Realtime Integrator for ULTRIX, however, provides icon support for the IET11 module through an interface layer between the IET11 driver and DEC Realtime Integrator's generic IEEE–488 icon.

IET11 TURBOchannel-to-IEEE–488 Adapter Features

The TURBOchannel-to-IEEE–488 adapter features:

- Provides IEEE–488 capability to DECstation 5000 series workstations
- Low-cost, high-performance IEEE–488 bus converter which sustains transfer rates of 1 Mbyte per second
- Installs in any single-width TURBOchannel slot
- Provides the highest attainable performance specified by the IEEE–488 standard
- Supports up to 14 IEEE–488 instruments or devices

- Compliance with all revisions of the IEEE–488 standard, including IEEE–488.2–1987 and *Standard Commands for Programmable Instrumentation*
- Complete IEEE–488 Controller/Talker/Listener capability
- ULTRIX-based software driver including diagnostic software
- Supported by DEC Realtime Integrator software

25.4 VAXBI Bus Options

The following VAXBI bus options are described in this section:

- DRB32–M,W,E Parallel-Line Interface Modules, Section 25.4.1
- KA800–M Processor Board, Section 25.4.2
- MS820–CA Memory Module, Section 25.4.3

25.4.1 DRB32–M,W,E Parallel-Line Interface Modules

The DRB32 is a high-speed, asynchronous, direct memory access (DMA) parallel interface to the VAXBI bus.

DRB32–M,W,E Parallel-Line Interface Modules Features

The DRB32 interface module features:

- 32-bit, half-duplex, bidirectional I/O path to transfer data to and from the user device
- Two 8-bit unidirectional paths for control data
- Limitless data transfers provided by a dual register set
- DMA block-mode transfer rates up to 6.7 Mbytes per second
- Page tables supported so that buffers do not need to be physically contiguous
- DR11 (16-bit) emulation mode available
- DRB32 VMS software included in one package
- Supported by VAXELN Toolkit, Version 3.1 or higher

The DRB32 is offered in three versions:

- DRB32–M Parallel-Line Interface Module, Section 25.4.1.1
- DRB32–W Parallel-Line Interface Module, Section 25.4.1.2
- DRB32–E Parallel-Line Interface Module, Section 25.4.1.3

DRB32–M,W,E Parallel-Line Interface Modules Use

The DRB32 parallel port is useful for realtime applications that collect data and control realtime devices using parallel lines. For example, in closely coupled symmetric multiprocessing configurations, VAXELN KA800 systems can use DRB32 devices to communicate with user devices. VAXELN KA800 systems can directly control the DRB32 parallel port for high interrupt response time.

25.4.1.1 DRB32–M Parallel-Line Interface Module

The DRB32–M is the VAXBI bus parallel interface. The DRB32 connects to a user device within the same cabinet system as the VAXBI system.

25.4.1.2 DRB32–W Parallel-Line Interface Module

The DRB32–W is the module with a subset of DR11–W functionality (providing a UNIBUS migration path to the VAXBI bus) plus DRB32–M. The DRB32–W connects to a user device within the same cabinet system as the VAXBI system.

25.4.1.3 DRB32–E Parallel-Line Interface Module

The DRB32–E is the long-line module plus DRB32–M. The DRB32–E connects to a user device or another DRB32–E up to 40 feet outside the system cabinet.

25.4.2 KA800–M Processor Board

The KA800 is a MicroVAX II class processor board in a VAXBI form factor, which you connect to the VAXBI bus of a VAX 6000 or 8000 system. The KA800 runs VAXELN software and serves as a secondary processor, typically dedicated to I/O handling tasks, in a closely coupled multiprocessor configuration.

KA800–M Processor Board Features

The KA800 module has these features:

- MicroVAX II-based CPU
- 1 Mbyte of on-board memory
- Private memory interconnect (PMI), used for all instruction fetches and references to local data
- Ability to access all physical address space available on the VAXBI bus to which it is connected, including both memory space and I/O space
- A slave port that gives other processors and I/O devices on the VAXBI bus access to its local memory

- Useable in any system with a VAXBI bus, including VAX 6000 systems, VAX 8200, VAX 8250, VAX 8300, VAX 8350,[1] VAX 8500, VAX 8530, VAX 8550, VAX 8700, and VAX 8800

KA800–M Processor Board Use

The KA800 module is used as a secondary processor in closely coupled symmetric VAX multiprocessor configurations. The KA800 processor may be dedicated to the control of critical, realtime devices to deliver optimal system response. The VAX 6000 or 8000 system in which the KA800 is installed—the primary system—may itself run either VMS or VAXELN. The KA800 does not support VMS locally.

25.4.3 MS820–CA Memory Module

The MS820 is a 16-Mbyte, dynamic random access memory (DRAM) high-speed memory module.

MS820–CA Memory Module Features

The MS820 module has these features:

- 16-Mbyte, DRAM high-speed memory module
- Useable in any system with a VAXBI bus, including VAX 6000 systems, VAX 8200, VAX 8250, VAX 8300, VAX 8350, VAX 8500, VAX 8530, VAX 8550, VAX 8700, and VAX 8800
- Public accessibility over the VAXBI bus

25.5 XMI-to-VMEbus Option

VMEbus is an industry-standard, high-performance, open I/O interconnect. VME is currently the most popular industry standard bus with more options than any other bus, and is well understood by systems and peripherals vendors. In an open systems environment, the XMI-to-VME option provides a transport from realtime, laboratory, and data-acquisition devices to a VAX 6000 datacenter system or server.

The VAX 6000 VME Adapter option is the interface between the XMI system bus of the VAX 6000 Models 200, 300, 400, 500, and 600 systems and the VMEbus. You can use this option to interface with any of the commercially available VMS add-in or custom modules.

[1] The VAX 8350 and KA800 configuration is supported only with VAXELN running on the VAX 8350 (primary) system, not with VMS.

The VMS operating system provides for the third party support with VMS device driver subroutines that allow you to write VME device drivers under supported systems. Both programmed I/O (PIO) and direct memory access (DMA) support routines are included. Setup, device interrupt routines, and "byte swapping" routines are includes for additional compatibility.

XMI-to-VMEbus Option Features

The XMI-to-VMEbus Option has these features:

- Supported on all VAX 6000 models
- Byte-swapping modes include: byte, word, longword, and no swap
- DMA support increases performance

25.6 Industrial Terminals

Digital offers a family of industrialized products that provide tolerance for manufacturing environments: heavy- to light-industrial, rugged hardware including terminals, keyboards, processors, air-conditioned computer cabinets, application software, and analog and digital I/O interfaces.

This section features two series of industrialized terminals (and accompanying keyboard options) offered by Digital that can be used in realtime industrial settings:

- IT330 and IT340 Industrial Terminals, Section 25.6.1
- VT33N and VT34N Industrial Terminals, Section 25.6.2

Additional product and configuration information can be found in the *Digital Systems and Options Catalog for Realtime.*

25.6.1 IT330 and IT340 Industrial Terminals

The IT330 and IT340 are Digital's industrialized terminals for use in heavy-industrial environments. Housed in sturdy, NEMA-12, cast-aluminum enclosures, these terminals give all the features and capabilities of the VT330 and VT340 terminals out on the factory floor. They have twice the resolution of the VT240 and VT241 terminals, and their graphics capabilities are up to five times faster.

IT330 and IT340 Industrial Terminal Features

The IT330 and IT340 industrial terminals feature:

- VT330/VT340 terminal functionality
- Tough, sealed, NEMA–12 enclosures to protect the terminals from vibration or rough handling, and a high-impact shield to protect the monitors against breakage
- Unique, passive-cooling design that lets the terminals withstand higher operating temperatures than ordinary office terminals
- IT3*nn*–AA membrane keyboard and IT3*nn*–AB tactile keyboard, also sealed to NEMA–12 standards, available for both the IT330 and IT340
- Optional tilt/swivel base and NEMA–12 wallmount and rackmount kits available for both IT330 and IT340 terminals and IT3*nn* keyboards

IT330 and IT340 Industrial Terminal Use

The IT330 and IT340 industrial terminals are ideal general-purpose terminals for inventory management and factory data collection on the shop floor. Because these terminals are sealed to the NEMA–12 industrial standard, they can operate in any harsh environment where dirt, airborne particles, and other noncorrosive liquids and leaking oil or coolants would damage an unprotected terminal or might cause traditional keyboards to stick and fail.

25.6.2 VT33N and VT34N Industrial Terminals

The VTN series of industrial terminals features Digital's VT330 and VT340 terminals packaged in NEMA–2 enclosures for light-industrial areas.

VT33N and VT34N Industrial Terminal Features

The VT33N and VT34N industrial terminals feature:

- Conformance to the NEMA–2 standard to provide protection from falling dirt and splashing, noncorrosive liquids while the rugged enclosures protect from rough handling.
- Optional LK207 compact keyboard. The LK207 can be rackmounted and is compatible with the LK201 standard keyboard for Digital's VT330 and VT340 terminals.
- For VTN series tabletop terminals, an optional keyboard garage. The VTN series terminal sits on the keyboard garage.
- Optional highly rugged keyboards (IT3*nn*–AA and IT3*nn*–AB) compatible with the VTN terminals. These may be ordered with the VTN series terminal alone.

- Optional rackmount kit that allows the VTN terminals to be mounted in industry-standard, 19-inch racks or cabinets. Alternatively, they can sit on a tabletop.

VT33N and VT34N Industrial Terminal Use

The VTN terminals are designed to operate in light-industrial environments where office-grade equipment would fail. These terminals can be located next to an assembly area, manufacturing process, or loading dock to provide convenient graphics capability.

A
Associated Documents

Appendix A lists supplementary readings on the products described in this technical summary. These associated documents have been grouped by product or functional category as follows:

- CASE tools
- DEC Ada, VAX Ada, and VAXELN Ada
- DECelx Toolkit
- DECmessageQ
- DEC OSF/1 realtime kernel
- DEC Realtime Integrator
- rtVAX 300 realtime processor board
- VAXELN Toolkit
- VAXELN Window Server (EWS)
- VMS POSIX
- XD Ada
- Additional documents

CASE Tools

Additional discussion on VMS and ULTRIX CASE tools is presented in the following manuals:

- *The Digital COHESION Environment for CASE.* A survey of Digital VMS and ULTRIX CASE tool offerings and associated computing environments. Some popular third-party CASE tools are also described.
- *Using DECset in the DEC COHESION Environment.* A tutorial for using DECset tools in the COHESION environment. This manual supersedes *Using VAXset.*

- *DEC FUSE Handbook*. A description of the FUSE software as used for software development, analysis, and maintenance of application programs.
- *DEC FUSE EnCASE Manual*. A tutorial for integrating tools into the DEC FUSE environment.

DEC Ada, VAX Ada, and VAXELN Ada

The following manuals document the VAX Ada and VAXELN Ada products:

- *Developing Ada Programs on VMS Systems*. A description of how to use the VAX Ada compiler, program library manager, and VAX Debugger to compile, link, run, and debug VAX Ada programs. This book also describes how to set up and maintain program libraries and how to use the Ada language-specific features of optional tools such as the VAX Language-Sensitive Editor and VAX Source Code Analyzer.
- *Reference Manual for the Ada Programming Language*. A presentation of the full text of the Ada standard, ANSI/MIL–STD–1815A–1983, as specified by the U.S. Government's Ada Joint Program Office.
- *VAX Ada and VAXELN Ada Technical Summary*. A technical summary of the VAX Ada and VAXELN Ada products. Topics covered include the VAX Ada and VAXELN Ada application development environments, execution environments, and VAX Ada compiler performance and capacity (on the host) and generated target code performance.
- *VAX Ada Language Reference Manual*. A presentation of the full text of the Ada standard, ANSI/MIL–STD–1815A–1983, as specified by the U.S. Government's Ada Joint Program Office, together with VAX Ada language-specific supplements. (Insertions and additions are printed in a different color.)
- *VAX Ada Runtime Reference Manual*. A collection of system-related material on topics such as VAX Ada storage allocation and object representations. This book also explains how to use VMS operating system components external to the language (for example, VMS system services). Furthermore, it explains how to use VMS operating system-related Ada features (such as multitasking and input/output) and how to call code written in other VAX languages from an Ada program.
- *VAX Ada Installation Guide*. Step-by-step instructions for installing the VAX Ada product, including information and recommendations on resource requirements.

- *VAXELN Ada User's Manual.* A description of the VAXELN Ada product features. This book also explains how to compile, link, build, execute, and debug VAXELN Ada applications. Furthermore, it describes the differences between the VAXELN Ada and VAX Ada products.
- *VAXELN Ada Installation Guide.* Step-by-step instructions for installing the VAXELN Ada product.

The following manuals document the DEC Ada on ULTRIX systems:

- *Developing Ada Programs on ULTRIX Systems.* A description of how to use the DEC Ada compiler, program library manager, and VAX dbx debugger to compile, link, run, and debug DEC Ada programs. This book also describes how to set up and maintain program libraries and how to use the Ada language-specific features of optional tools such as the DEC Language-Sensitive Editor and DEC Source Code Analyzer.
- *DEC Ada Language Reference Manual.* A presentation of the full text of the Ada standard, ANSI/MIL–STD–1815A–1983, as specified by the U.S. Government's Ada Joint Program Office, together with DEC Ada language-specific supplements.
- *Run-Time Reference Manual for ULTRIX Systems.* A collection of system-related material on topics such as DEC Ada storage allocation and object representations. This book also explains how to use ULTRIX operating system components external to the language.
- *DEC Ada Installation Guide for ULTRIX Systems.* Step-by-step instructions for installing the DEC Ada product, including information and recommendations on resource requirements.

DECelx Toolkit

The following manuals further detail the DECelx product:

- *Introduction to DECelx.* A brief overview of the DECelx Realtime Tools for ULTRIX product and DECelx development environment for realtime applications.
- *DECelx Reference Manual.* A description of the functions provided in the DECelx callable interface, including the POSIX functions.
- *DECelx Programming Guide.* A description of the components of the DECelx system and how these components interrelate.
- Hardware Supplements. These supplements to the *DECelx Reference Manual* provide reference information for coding DECelx routine calls that are specific to the processor. There is one supplement for each of the supported processors.

- *DECelx Guide to ElxGDB*. There is one guide for each of the architectures supported by DECelx. This guide describes ElxGdb, the source-level debugger for debugging DECelx applications.
- *DECelx Guide to the GNU Toolkit*. This guide describes the customized versions of the GNU C compiler, assembler, loader, and object-file utilities for cross-development to Motorola processors (68K architecture).
- *DECelx Board Support Porting Guide*. This guide describes the how to port DECelx board support to an otherwise unsupported 680*n*0 or R3000-based target platform.

DECmessageQ

For descriptions of using the DECmessageQ software, refer to the following manuals:

- *DECmessageQ Introduction to Messaging*. A description of how DECmessageQ interprocess message queuing software integrates distributed applications in a multivendor environment.
- *DECmessageQ Guide to Application Integration*. An explanation of how to integrate distributed applications using DECmessageQ software.
- *DECmessageQ Programmer's Reference*. Reference material for programmers who use the DECmessageQ application programming interface.
- DECmessageQ Management and Configuration Guides. Provide task-oriented information for DECmessageQ system managers and explains how to configure and troubleshoot DECmessageQ software.
- DECmessageQ Installation Guides. Instructions for installing DECmessageQ.

DEC OSF/1 Realtime Kernel

To enhance your understanding of the developing realtime applications using the DEC OSF/1 operating system, refer to these manuals:

- *DEC OSF/1 Guide to Realtime Programming*. A description of how to use P1003.4/D11 functions in combination with other DEC OSF/1 system and library functions to write realtime applications.
- *DEC OSF/1 Programmer's Guide*. A description of the programming environment for the DEC OSF/1 operating system with an emphasis on the C programming language. This manual covers program development, shared libraries, build and debugging programs, and improving performance.

- *DEC OSF/1 Guide to Programming Support Tools*. A description of commands and utilities in the DEC OSF/1 system, including facilities for text manipulation, macro and program generation, source file management, and software kit installation and creation.

DEC Realtime Integrator

For descriptions of using the DEC Realtime Integrator software for application development, refer to the following manuals:

- *DEC Realtime Integrator Icon Developer's Guide*. A guide to developing icons for use in DEC RT Integrator applications.
- *DEC Realtime Integrator Installation Guide*. A guide to installing the DEC RT Integrator Development Kit and Runtime Kit on RISC ULTRIX and VMS systems.
- *DEC Realtime Integrator Application Developer's Guide*. A guide to developing and running applications with the DEC RT Integrator graphical user interface.

For descriptions of using only the DEC Realtime Integrator graphical user interface, refer to the following manuals:

- *Getting Started with the DEC RT Integrator Subroutine Libraries*. A description of the subroutine library components of the product.
- *DEC RT Integrator Laboratory Graphics Package (LPG) Reference Manual*. A reference manual describing how to plot realtime data or data produced by calculations in two dimensions, three dimensions, and 2-dimensional contours from a 3-dimensional view.
- *DEC RT Integrator Laboratory Input/Output (LIO) Reference Manual for ULTRIX*. A reference manual describing how to initiate, control, process, and terminate I/O to and from I/O devices under the ULTRIX operating system.
- *DEC RT Integrator Laboratory Input/Output (LIO) Reference Manual for VMS*. A reference manual describing how to initiate, control, process, and terminate I/O to and from I/O devices under the VMS operating system.
- *DEC RT Integrator Laboratory Signal Processing (LSP) Reference Manual for VMS*. A reference manual describing how to use the signal-processing routines to perform Fourier transforms, correlation functions, and data filtering.

rtVAX 300 Realtime Processor Board

The next manuals further discuss the rtVAX 300 realtime processor board:

- *rtVAX 300 Hardware User's Guide.* A collection of rtVAX 300 technical and physical specifications and details necessary for configuring this processor into a host and target configuration. Specifically, this guide describes the following interfaces: memory system, console and boot ROM, network interconnect, and I/O device.
- *rtVAX 300 Programmer's Guide.* A guide to writing device drivers that interface the rtVAX 300 processor with realtime devices.
- *VAXELN rtVAX 300 Supplement.* A description of the steps that may be required to tailor the VAXELN kernel image to run in a specialized rtVAX 300 target system configuration. Such tailoring is performed only when the VAXELN Toolkit's basic support for an rtVAX 300 target and associated integral-bus devices must be augmented, for example, to support an external I/O bus and associated devices not supported by the VAXELN Toolkit.

VAXELN Toolkit

The following manuals in the VAXELN documentation set will enhance your understanding of the VAXELN Toolkit:

- *Introduction to VAXELN.* A manual that surveys the features of the VAXELN Toolkit, introduces VAXELN concepts and practices, and illustrates the design, coding, building, and running of a sample VAXELN application.
- *VAXELN Ada Programming Guide.* A guide to using the VAX Ada language to program VAXELN applications. This guide explains how to develop and debug VAXELN Ada application programs. The guide also discusses I/O, tasking, VAXELN data types, the creation of routine bindings, device handling, and exception handling.
- *VAXELN Application Design Guide.* A guide that uses sample VAXELN applications to illustrate VAXELN application design issues, such as the use of asynchronous I/O, multiple-cicuit servers, and device drivers.
- *VAXELN C Programming Guide.* A guide to using the VAX C language to program VAXELN applications. This guide explains how to develop VAXELN C application programs. The guide also discusses the C interface to the VAXELN kernel, I/O, device handling, and exception handling. Descriptions of C runtime library functions and macros are also included.

- *VAXELN Device Drivers Guide*. A guide to using the VAXELN Toolkit device drivers. This guide discusses the Ethernet, disk, tape, printer, terminal, Small Computer System Interface (SCSI), and realtime device drivers that are supplied in the toolkit. The guide also explains how to use the programming interfaces that are available for some of these drivers.
- *VAXELN Device Drivers Reference Manual*. A reference manual that describes VAXELN device driver interface routines. Routine descriptions include an overview, language-specific format information, argument descriptions, and status values.
- *VAXELN File, Network, and Security Services Guide*. A guide to using the VAXELN Toolkit file, network, and security services. This guide discusses and explains how to use the programming interfaces for the File Service, Ethernet/IEEE 802 Datagram Service, DECnet Service, Internet Services, Local Area Terminal (LAT) Host Services, and Authorization Service.
- *VAXELN File, Network, and Security Services Reference Manual*. A reference manual that describes VAXELN file, network, and security service interface routines. Routine descriptions include an overview, language-specific format information, argument descriptions, and status values.
- *VAXELN FORTRAN Programming Guide*. A guide to using the VAX FORTRAN language to program VAXELN applications. This guide explains how to develop VAXELN FORTRAN application programs. The guide also discusses programming considerations, device handling, and exception handling.
- *VAXELN Guide to DECwindows*. A guide to programming and building dedicated, realtime applications that integrate VAXELN and DECwindows software. This guide provides a VAXELN DECwindows overview and explains how to build the VAXELN DECwindows Server into VAXELN applications, program VAXELN DECwindows applications, and use VAXELN DECwindows user environment components. The guide also walks you through the development of a sample VAXELN DECwindows application. You use this guide as a supplement to the VMS DECwindows documentation.
- *VAXELN Guide to Using POSIX*. A guide to programming VAXELN POSIX applications. This guide explains how to develop VAXELN POSIX application programs and discusses issues concerning the use of POSIX functions in VAXELN application programs. The guide also explains how to use process primitives, the process environment, I/O primitives, device and class-specific functions, the system database, binary semaphores, clocks,

and timers. Memory usage, priority scheduling, and message passing are also discussed.

- *VAXELN KAV30 Programming Guide.* A guide that describes the KAV30 software and hardware and how to develop realtime applications for the KAV30 using the VAXELN Toolkit software.
- *VAXELN Messages Manual.* A reference manual that describes the messages produced by the VAXELN Toolkit system development and runtime software. Each description includes an explanation and, where applicable, a suggested recovery procedure.
- *VAXELN Pascal Programming Guide.* A guide to using the VAXELN Pascal language to program VAXELN applications. This guide describes the VAXELN Pascal program structure and the language components, which include declarations, data types, constants, variables, expressions and operators, statements, procedures and functions, and predefined routines. The guide explains how to use language-defined elements for queues and exception handling. The guide also explains how to develop VAXELN Pascal programs, including how to use the VAXELN Pascal compiler.
- *VAXELN POSIX Callable Interface Reference Manual.* A reference manual that describes the functions provided by the VAXELN POSIX callable interface. Function descriptions include an overview, declaration formats, argument descriptions, and status values.
- *POSIX Conformance Information for VAXELN (Std 1003.1).* A manual that discusses details of the VAXELN implementation of the IEEE 1003.1-1990 standard (POSIX.1). This information includes limit values, implementation defined features, and features of VAXELN POSIX that do not conform to the POSIX.1 standard.
- *POSIX Conformance Information Concerning Draft 11 of P1003.4.* A manual that discusses details of the VAXELN implementation of POSIX.4 Draft 11 (POSIX.4/D11). This information includes limit values, implementation defined features, and features of VAXELN POSIX that do not comply with POSIX.4/D11.
- *VAXELN POSIX Information Concerning Draft 4 of P1003.4a.* A manual that discusses details of the VAXELN implementation of POSIX.4a Draft 4 (POSIX.4a/D4). This information includes limit values, implementation defined features, and features of VAXELN POSIX threads that do not comply with POSIX.4a/D4.

- *VAXELN Quick Reference*. A document that summarizes components of the VAXELN Toolkit and serves as a convenient, fast reference tool as you program and develop VAXELN applications. This reference document lists language-specific formats for the VAXELN kernel and utility routines and summarizes commands, qualifiers, and other information pertaining to program development and the use of VAXELN Toolkit utilities.
- *VAXELN Release Notes*. A document that discusses release-specific enhancements, corrections, restrictions, and documentation errors and omissions. This document also provides VAXELN performance data.
- *VAXELN System Development Guide*. A guide that explains how to develop a VAXELN system image. This guide explains how to build program images, build a system image, and load and boot the system image on target processors.
- *VAXELN System Services Guide*. A guide to using the VAXELN Toolkit system services. This guide describes VAXELN kernel objects and data structures. The guide also explains how to use those objects and structures with VAXELN kernel and utility routines to manage jobs, processes, and memory; synchronize process execution; program communication between processes and jobs; and handle device interrupts and exceptions.
- *VAXELN System Services Reference Manual*. A reference manual that describes VAXELN system service routines. The system service routines include kernel routines, kernel-related utility routines, VAX instruction routines, and other general-purpose routines for VAXELN programming. The routine descriptions include an overview, language-specific format information, argument descriptions, and status values.
- *VAXELN Utilities Guide*. A guide that discusses the VAXELN utlities. This guide explains how to use and describes the commands for the debugger, Performance Utility, Display Utility, Command Language, LAT Control Program, outbound Remote Terminal Utility, and Error Logging Services.

These manuals are not part of the VAXELN documentation set and can be ordered separately:

- *VAXELN Internals and Data Structures*. A description of the internal data structures and operations of the VAXELN kernel.
- *VAXELN rtVAX 300 Supplement*. A description of the steps that may be required to tailor the VAXELN kernel image to run in a specialized rtVAX 300 target system configuration. Such tailoring is performed only when the VAXELN Toolkit's basic support for an rtVAX 300 target and associated integral-bus devices must be augmented, for example, to support

an external I/O bus and associated devices not supported by the VAXELN Toolkit.

VAXELN Window Server

Additional information on installing and using the VAXELN Window Server software is supplied in the next manuals:

- *ULTRIX Worksystem Software DECwindows Desktop Applications Guide.* A guide that explains how to use a variety of desktop applications in the ULTRIX Worksystem Software DECwindows environment.
- *ULTRIX Worksystem Software DECwindows User's Guide.* A guide that explains how to use the ULTRIX Worksystem Software DECwindows software.
- *VAXELN Window Server Installation Guide for ULTRIX Systems.* A guide that explains how to install the VAXELN Window Server software on the ULTRIX operating system.
- *VAXELN Window Server Installation Guide for VMS Systems.* A guide that explains how to install the VAXELN Window Server software on the VMS operating system.
- *VAXELN Window Server User's Manual.* A manual that furnishes information about using workstations and terminals that run VAXELN Window Server software.

VMS POSIX

These manuals furnish further details on the VMS POSIX product:

- *Guide to Using VMS POSIX.* A guide that explains how to use POSIX 1003.1, P1003.2/D10, and P1003.4/D9 functions in combination to write portable applications using VMS POSIX.
- *VMS POSIX Reference Manual: Shell and Utilities.* A description of the POSIX shell and utilities as presented in P1003.2/D10 and P1003.2a standards. This manual also contains descriptions of VMS POSIX utilities not covered in the standard.
- *VMS POSIX Reference Manual: Callable Interface.* A description of the callable interface and realtime functions as presented in POSIX 1003.1, P1003.2/D10, and P1003.4/D9 standards or draft standards. This manual is designed to be used in conjunction with the *Guide to Using VMS POSIX.*

XD Ada

The following manuals further detail the XD Ada cross-development system:

- *Developing XD Ada Programs on VMS Systems for the MC68020.* A description of how the XD Ada compiler, program library manager, and debugger differ from their VAX Ada counterparts. Also described is how to use the XD Ada toolset for developing applications that will run on Motorola MC68020 microprocessor target systems.
- *XD Ada MC68020 Assembly Language Reference Manual.* A description of how to use the assembly language that is supplied as part of the MC68020 version of the XD Ada toolset.
- *XD Ada MC68020 Installation Guide.* Step-by-step instructions for installing XD Ada for MC68020 target processor application development.
- *XD Ada MC68020 Runtime Reference Manual.* A description of the XD Ada MC68020 target system hardware. This book also informs you about program execution on the target, including memory allocation, compiler optimization, interfacing with assembly language, and writing interrupt handlers.
- *XD Ada MC68020 Supplement to the Ada Language Reference Manual.* A collection of supplements to the Ada standard ANSI/MIL–STD–1815A–1983. This book describes the XD Ada interpretation of the MC68020 target processor-dependent language features and permitted implementation-dependent additions to the language.
- *Developing XD Ada Programs on VMS Systems for the MIL-STD-1750A.* A description of how the XD Ada compiler, program library manager, and debugger differ from their VAX Ada counterparts. Also described is how to use the XD Ada toolset for developing applications that will run on MIL–STD–1750A microprocessor target systems.
- *XD Ada MIL-STD-1750A Assembly Language Reference Manual.* A description of how to use the assembly language that is supplied as part of the MIL–STD–1750A version of the XD Ada toolset.
- *XD Ada MIL-STD-1750A Installation Guide.* Step-by-step instructions for installing XD Ada for MIL–STD–1750A target processor application development.
- *XD Ada MIL-STD-1750A Runtime Reference Manual.* A description of the XD Ada MIL–STD–1750A target system hardware. Also, this book discusses program execution on the target, including memory allocation, compiler optimization, interfacing with assembly language, and writing interrupt handlers.

- *XD Ada MIL-STD-1750A Supplement to the Ada Language Reference Manual.* A collection of supplements to the Ada standard ANSI/MIL–STD–1815A–1983. This book describes the XD Ada interpretation of the MIL–STD–1750A target processor-dependent language features and permitted implementation-dependent additions to the language.
- *XD Ada Technical Summary.* A technical overview of the XD Ada cross-development system. This book describes software and hardware configurations for host development and target runtime environments, the host program support environment, Ada language-related issues, host-to-target communications, XD Ada cross-compiler performance and capacity (on the host) and generated target code performance, software validation, and support services. It also includes some sample code listings.

Additional Documents

Some of the products described in this book are discussed in the following documents:

- *Digital Systems and Options Catalog for Realtime.* The most current descriptions, ordering, and configuration information available on Digital software and hardware products for use in realtime applications.
- *Realtime User's Guide.* A technical description of the realtime features of Digital VAX hardware and software systems used in scientific and industrial settings. This guide assists in configuring Digital VAX systems for realtime applications and in programming these applications.
- *VAX Architecture Reference Manual.* A detailed technical description of the VAX architecture, including virtual address translation, data representations, instruction formats, addressing modes, interrupt schemes, and memory management.
- *VAX Hardware Handbook.* A collection of general technical material for the VAX hardware product line. This book includes descriptions and specifications for the VAX processors, data storage systems and devices, VAXcluster configurations, and communication products.
- *Digital Systems and Options Catalog.* A collection of descriptions, ordering, and configuration information on Digital products, including VAX Systems, DECsystems, communications hardware, disks and tapes, terminals and printers, integrated personal computing solutions, and software and service offerings.
- *VMS DCL Dictionary.* A collection of detailed discussions and examples of VMS Digital Command Language (DCL) commands and lexical functions.

- *VMS Linker Utility Manual.* A description of how the VMS Linker works and how to use it.
- *VMS Network Control Program Reference Manual.* A description of how to use the Network Control Program (NCP) to manage DECnet–VAX networks.

B Trademarks

Appendix B lists Digital and third-party trademarks referenced within this technical summary.

The cover photo for this manual appears courtesy of the National Aeronautics and Space Administration, Houston, Texas.

The following are trademarks of Digital Equipment Corporation: ACCESSbus, BASEstar, CDA, CDD/PLUS, CDD/Repository, CI, COHESION, DDIF, DEC Ada, DEC C, DEC CMS, DECdevice, DEC Fortran, DEC FUSE, DEC GKS, DEC LSE/SCA, DEC MMS, DEC OSF/1, DEC Pascal, DEC PCA, DEC PHIGS, DEC Realtime Integrator, DEC Test Manager, DEC@aGlance, DECdesign, DECelx, DECelx BSP, DECelx Runtime, DECelx Realtime, DECforms, DECgraph, DECimage, DECmessageQ, DECmessageQ Message Bus, DECnet, DECnet–ULTRIX, DECnet–VAX, DEComni, DECosap, DECpresent, DECscan, DECserver, DECset, DECstation, DECwindows, DECwrite, Digital, DRB32, DRQ, DSSI, DTIF, EDCS, FUSE, INTERNET, LA50, LK, Local Area VAXcluster, LN03 PLUS, LVP16, MicroPDP–11, MicroPower/Pascal, MicroVAX, MS820, PAMS, PATHWORKS, PDP–11, PixelStamp, Q–bus, Q22–bus, RSX–11, RT–11, rtVAX, rtVAXstation, SQL, ThinWire, TK, TURBOchannel, ULTRIX, UNIBUS, VAX, VAX Ada, VAXBI, VAX BASIC, VAX C, VAX DIBOL, VAX DOCUMENT, VAX FORTRAN, VAX Notes, VAX Pascal, VAX Performance Advisor, VAX Rdb/VMS, VAX SCAN, VAX SPM, VAXBI, VAXcamps, VAXcluster, VAXELN, VAXELN Pascal, VAXELN Toolkit, VAXlab, VAXserver, VAXset, VAXstation, VAXft, VAX/VMS Connection (UCX), VMS, VT, VT1300, XD Ada, XUI, and the DIGITAL logo.

Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

Apple and Macintosh are registered trademarks of Apple Computer, Inc.

AT&T is a registered trademark of American Telephone and Telegraph.

Bell is a trademark of Bell Telephone Companies.

BSD is a trademark of the University of California, Berkeley.

CRAY is a trademark of Cray Research, Inc.

DataViews is a registered trademark of Visual Intelligence Corporation.

Domain is a registered trademark of Apollo Computer, Inc., a subsidiary of Hewlett-Packard Company.

EXCELERATOR is a registered trademark of Index Technology Corporation.

Futurebus+ is a trademark of Force Computers GMBH, Federal Republic of Germany.

Glockenspiel C++ is a registered trademark of Glockenspiel, Ltd.

HP and HP-UX are trademarks of Hewlett-Packard Company, Inc.

IBM and OS/2 are registered trademarks of International Business Machines Corporation.

Intel is a trademark of Intel Corporation.

Macintosh and MacX are registered trademarks of Apple Computer, Inc.

Micro Switch is a trademark of Honeywell, Inc.

Microsoft and MS–DOS are registered trademarks of Microsoft Corporation.

Motorola is a registered trademark of Motorola, Inc.

MS and MS–DOS are registered trademarks of Microsoft Corporation.

Network File System and NFS are trademarks of Sun Microsystems, Inc.

Open Software Foundation, OSF, OSF/1, and OSF/DCE are trademarks of the Open Software Foundation, Inc.

OSF/Motif and Motif are registered trademarks of the Open Software Foundation, Inc.

PostScript is a registered trademark of Adobe Systems, Inc.

Radstone is a trademark of Radstone Technology PLC.

RS/1 is a registered trademark of BBN Software Products Corporation.

SAS is a registered trademark of SAS Institute, Inc.

SINEC, SINUMERIK, SICOMP, and SIROTEC are trademarks of Siemens AG.

Software through Pictures is a trademark of Interactive Development Environments, Inc.

SONIC is a trademark of National Semiconductor.

Star MVP/R3000 is a trademark of Lockheed Sanders.

Statemate is a registered trademark of i-Logix, Inc.

SUN, SunOS, and SUN/SPARC are trademarks of Sun Microsystems, Inc.

System V, System V/88, and AT&T are registered trademarks of American Telephone & Telegraph Company in the U.S. and other countries.

TEAMWORK, TEAMWORK/Access, TEAMWORK/RT, TEAMWORK/SA, and TEAMWORK/SD are registered trademarks of Cadre Technologies, Inc.

TEKTRONIX is a registered trademark of Tektronix, Inc.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

VMEmodule and SYSTEM V/88 are trademarks of Motorola, Inc.

VR3000 is a trademark of Omnibyte.

X/Open is a trademark of X/Open Company Limited.

X Window System, Version 11 and its derivations (X, X11, X Version 11, X Window System) are trademarks of the Massachusetts Institute of Technology.

68000, 68020, and 68030 are trademarks of Motorola, Inc.

Glossary

application device

In a local area transport (LAT) configuration, a remote device attached to a terminal server, which offers a service to the VAXELN service node.

area

A region of physically contiguous memory accessible to all jobs executing on the same node in a local area network (LAN).

area-lock variable

A variable that resides in an area for synchronizing job access to that area. Using an area-lock variable, a process can lock an area to gain exclusive access.

AST (asynchronous system trap)

A VMS mechanism for furnishing a software interrupt when an external event occurs.

AST (asynchronous system trap) service routine

A user-written routine that receives control when an AST is delivered after being queued to a process.

asynchronous I/O

In a POSIX environment, a type of non-blocking I/O. The process making the I/O request immediately regains control of execution once the I/O operation is queued to the device.

authorization server

The node in a multinode VAXELN system that manages a shared Authorization Service database.

binary semaphore

A semaphore that enforces exclusive access to a shared resource.

boot server

In the VAXELN context, a VMS or ULTRIX host system that downline loads, over Ethernet, VAXELN system images to target systems.

booting

A technique that brings a device or processor to a state in which it can operate on its own.

CASE (Computer-Aided Software Engineering)

A software development environment that helps developers create and maintain software faster, with high-quality code.

circuit

See virtual circuit.

client

An application program connected to the server.

closely coupled multiprocessing

A computer configuration consisting of a VAX primary system running a VAXELN or VMS system and one or more KA800 processor boards. The KA800 processors are linked to the primary processor by a VAXBI bus. Each KA800 processor runs its own copy of a VAXELN system image and its own collection of jobs.

COHESION

Digital's COHESION environment is designed for the development, deployment, and management of software. COHESION includes a comprehensive CASE environment that supports all phases of the software life cycle and emphasizes efficiency and productivity throughout these phases.

communication region

An area of memory that an interrupt service routine (ISR) uses for communicating with device driver code while servicing a device.

compilation unit

A source file consisting of a program block or module that you can compile to produce an object module.

compiler

A program that translates a high-level language program into an object module of binary machine code.

Complementary Solutions Organization (CSO)

A leading third-party application supplier that allies with Digital through the Cooperative Marketing Program (CMP) and through joint marketing, joint product development, system reseller, or Complementary Software House (CSH) agreements. Digital and its CSOs share product goals and directions, supplying Digital's customers with the best integrated solutions on the market.

Compound Document Architecture (CDA)

An open, integrated, Digital architecture for creating, displaying, revising, processing, managing, and distributing compound documents throughout a networked, heterogeneous computing environment. CDA defines two primary formats: Digital Document Interchange Format (DDIF) and Digital Tabular Interchange Format (DTIF). The encoding of CDA documents is compatible with the American National Standards Institute (ANSI) standard. CDA also provides support for full two-way conversion to and from both International Standards Organization (ISO) standards: Office Document Architecture/Office Document Interchange Format (ODA/ODIF) and Standard Generalized Markup Language (SGML). *See also* Digital Document Interchange Format (DDIF) and Digital Tabular Interchange Format (DTIF).

connection

In a DECwindows context, the network path between a client and server.

console emulator

A program that displays a window that looks and functions like a console device, providing minimal terminal input/output (I/O) functionality.

control region

See P1 virtual address space.

counting semaphore

A semaphore that permits metered access, allowing a specified number of processes simultaneous access to units of a shared resource.

data access protocol (DAP)

A set of standardized formats and procedures that facilitate the creation, deletion, transfer, and access of files between a process and a file system in a network environment.

datagram

A message sent between two named ports in a VAXELN system. Datagrams do not require an explicit connection sequence.

debugger

A program that helps you find errors in application programs. For example, the VAXELN Debugger supports local and remote debugging environments.

DECmessageQ

A family of software products that provide interprocess communication for distributed applications through a message queuing system.

DECmessageQ Message Queuing Bus

Digital's product implementation of the message queuing technology.

DECset

A collection of Digital's CASE offerings for ULTRIX systems. DECset tools include a Motif interface for software development tools.

DEC OSF/1

DEC OSF/1 is Digital's implmentation of the OSF/1 operating system.

dedicated application

An application that uses a computer to solve a specific problem or a set of related problems.

development software

Programs that serve as tools for developing applications.

device driver

A program that controls communication between application programs and external devices.

device interrupt

See interrupt.

Digital Document Interchange Format (DDIF)

A standard Compound Document Architecture (CDA) interchange format for the complete life cycle of networked, electronic compound documents. DDIF compound documents include text, graphics, and images. Compliant applications can interchange any common data. *See also* Compound Document Architecture (CDA).

Digital Tabular Interchange Format (DTIF)

A standard Compound Document Architecture (CDA) interchange format for the complete life cycle of networked, electronic compound documents. DTIF compound documents include text and tabular or spreadsheet data. Compliant applications can interchange any common data. *See also* Compound Document Architecture (CDA).

distributed processing

A computer configuration in which computer resources are distributed among multiple processors linked by a common communication path, such as a local area network (LAN) or VAXBI bus.

downline loading

Transmitting a runtime image from a host VAX processor over an Ethernet logical link to a target VAX processor where the image is loaded and started.

emulator

A device or program that allows application execution on a different type of computer than the one on which an application was developed or for which it was written.

Ethernet

A local communication network that employs coaxial cable as a passive communications medium. The coaxial cable interconnects different kinds of computers, information processing products, and office equipment at a local business site without requiring switching logic or control by a central processor.

event

A flag that identifies the occurrence of a realtime event. Events synchronize process execution and access to shared data.

event response

The ability of a process to coordinate its activities with other processes.

exception

A hardware or software event that changes the normal flow of a program's execution synchronously or asynchronously. Synchronous exceptions occur at predictable points during a program's execution. Asynchronous exceptions result from unpredictable events, such as power failures.

exception handler

A routine or program function that traps exceptions and handles them through appropriate action.

executive

A program that controls a VAXELN system's hardware resources and the execution of the system's software.

Friendly Unified Software Environment (FUSE)

A collection of Digital's CASE offerings for ULTRIX and UNIX systems. FUSE software development tools are based on traditional UNIX tools. FUSE includes a consistent Motif interface, a database, and FUSE EnCASE, a tool for integrating additional tools into the FUSE environment.

function

A routine that defines executable code and data that executes from and returns control and a value to the calling routine.

gateway

A computer system that physically connects and transfers messages between networks.

global section

An area of physical memory that is contained within the virtual-address space of multiple processes.

hibernate state

In the VMS context, a synchronization mechanism (accessible through a VMS system service call) that allows a process to control when it becomes active. A process can only place itself into the hibernate state; that is, one process cannot put another into hibernation. Hibernation temporarily halts process execution. *See also* suspend state.

host

In an Internet environment, the source and destination computer systems of transferred data.

host processor

In the VAXELN context, a VAX computer, running the VMS operating system, on which you develop VAXELN applications by using VAXELN and VMS development tools.

IEC/IEEE–488 Standard (1978)

An industry-standard bus for digital interfacing to programmable instrumentation.

image file

An executable, shareable, or system-type file containing information that establishes the process context for a user program.

interactive terminal

In a local area transport (LAT) configuration, a user-controlled terminal device connected to a terminal server, which lets you establish a session with a service offered by the VAXELN service node.

Internet

A set of networks that are connected by hosts called gateways.

interrupt

An event other than an exception that changes the normal flow of instruction execution. Interrupts are external to the executing process and occur asynchronously to the currently executing instruction stream.

interrupt priority level (IPL)

The interrupt level at which a hardware component generates an interrupt. IPL ranges are system-dependent.

interrupt service routine (ISR)

A routine that responds to a device's interrupt signal and services the device's request.

interrupt service routine (ISR) latency

The amount of time required by a system to recognize an external interrupt, cease the current activity, save the current process state, and execute the first instruction in the ISR.

job

An instance of a VAXELN application program, consisting of a master process and zero or more subprocesses, that executes concurrently with other jobs and processes.

kernel

A small, realtime executive that controls target hardware resources and the execution of VAXELN system software. *See* UNIX kernel and preemptive kernel.

kernel object

A data structure that represents an ongoing activity (such as process execution) or a hardware or software resource (such as a device, memory region, event, or message).

local area network (LAN)

A privately owned data communications system that offers high-speed communications channels optimized for connecting information-processing equipment. The geographical area is usually limited to a section of a building, an entire building, or a group of closely situated buildings.

local area transport (LAT)

A communications protocol that lets system nodes running LAT host services communicate with dedicated terminal server nodes running LAT server services.

local area transport (LAT) network

The collection of system nodes and terminal server nodes in a local area network (LAN).

local data

Data that you declare within a routine.

local debugging environment

The VAXELN debugger environment that you can include as part of a VAXELN system image and use from a target system console.

local port name

A port name known only to processes and jobs on the node on which the name is created.

loosely coupled multiprocessing

A computer configuration in which computer resources are distributed among multiple processors that are linked by a common *external* communication or data path. For example, the configuration can consist of multiple VAX processors participating on the same Ethernet segment of a local area network (LAN).

MacX X Window System Server

A Macintosh application that implements an X Window System server using the industry-standard X Window System, Version 11.4 (X11) protocol. MacX X Window System Server is developed by Apple Computer and is furnished with the PATHWORKS for Macintosh software.

main memory

See physical memory.

master process

The process that represents the execution of a program's main section of code: the innermost procedure block for Ada programs, the main routine for C programs, the main program for FORTRAN programs, and the program block for VAXELN Pascal programs.

memory locking

In a POSIX environment, one method to ensure that a process stays in primary memory. UNIX functions explicitly lock processes into memory.

memory management

The system functions that use hardware to map a VAXELN system image into a processor's virtual address space.

message

A block of contiguous bytes of memory that is transmitted between processes in the same or different jobs, processes, or applications.

Message Recovery Services (MRS)

A set of DECmessageQ services which manage the automatic redelivery of critical messages.

metadata

Data definitions associated with actual data. In the VAX CDD/Plus repository context, metadata refers to data stored in the repository that keeps track of location, type, format, size, change history, and usage of the actual data. The metadata that is stored in the VAX CDD/Plus repository may be used by multiple software products. *See also* repository.

Manufacturing Message Specification (MMS)

Internationally accepted application-layer, communications protocol. MMS makes it possible to monitor and control disparate devices such as programmable sensors, robots, controllers, and numerical-control machines.

module-level data

Data that a job's processes can use to communicate. Module-level data includes constants, types, variables, procedures, functions, and process blocks that you declare outside routines.

multiprocessing

Concurrent processing of an application's parts on more than one processor.

multiprogramming

Concurrent processing of multiple jobs, including multitasking jobs.

multitasking

The division of an application program's labor into a set of smaller, focused tasks that execute concurrently.

mutex

A mutual exclusion semaphore that ensures that only one process at a time has exclusive access to a shared resource.

mutual exclusion

The ability of a process to gain sole access to a shared resource.

name

An entry in a name table that associates a character string name with a message port or process. Names can be local or universal.

name server

A target system that manages a multinode VAXELN application's universal name table.

name service

The Network Service component that provides kernel extensions so that jobs can access and maintain a universal name table.

network

A collection of computers physically connected by a communication medium, such as an Ethernet. *See also* Ethernet.

Network Application Services (NAS)

A comprehensive set of standards-based software that consists of well-defined programming interfaces, toolkits, and products to help developers build applications that are integrated and easily ported across a distributed, multi-vendor environment.

node

A computer system in a network that can communicate with other systems in the network.

NTSC (National Television Standards Committee)

A video signal standard used in the United States. It defines a composite video signal transmitted on a wire as 525 scan lines at a field frequency of 60 Hz displaying 30 frames per second.

object

See kernel object.

OSF (Open Software Foundation)

The OSF is open to all computer companies wishing to support a commitment to a standards-based operating system. OSF is pledged to POSIX compliance and has adopted Motif as its windowing standard.

OSF/1

OSF/1 is a UNIX operating system based on the MACH and BSD kernels.

PAL (phase-alternating line)

A video signal standard used (in several variants) throughout much of Europe, except France and the Soviet Union. It defines the color encoding of a composite video signal transmitted on a wire as 625 scan lines, a field frequency of 50 Hz, and displaying 25 frames per second.

PAMS

Process Activation and Message Support. PAMS is the original name for the DECmessageQ messaging system.

PC DECwindows Display Facility

An MS–DOS application that implements an X Window System server using the industry-standard X Window System, Version 11 (X11) protocol. PC DECwindows Display Facility is developed by Microsoft and is supplied with the PATHWORKS for DOS software.

P0 virtual address space

In the VAXELN context, the area of a target processor's virtual memory into which the kernel maps job program images, data, and message buffers.

P1 virtual address space

In the VAXELN context, the area of a target processor's virtual memory into which the kernel maps data associated with dynamically created processes. Each process in a job, including the master process, has its own P1 virtual address space.

physical address

The address of a program in terms of and limited by the amount of a computer's actual, physical memory.

physical memory

A computer's primary storage in which program instructions and data are stored and executed.

port

A system-maintained queue for messages waiting to be sent and received.

portability

The ability to compile an unaltered source program on several operating systems and machines.

port name

An entry in a name table that associates a character string name with a VAXELN message port or process. Port names can be local (known only on their own node) or universal (known on any node in the LAN).

position-independent code (PIC)

A coding technique that lets code execute anywhere in the virtual address space without alteration.

POSIX (Portable Operating System Interface for Computer Environments)

A collection of standards proposed by the POSIX working groups of the IEEE Computer Society. POSIX standards are developed through participation by representatives of many computer companies. POSIX standards define system interfaces to support the source portability of applications.

preemptive kernel

A realtime UNIX kernel that guarantees a deterministic response to realtime events by providing the ability to respond quickly to requests.

priority

An integer value assigned to a job or process to determine when that job or process can access system resources. Priority ranges are system-dependent.

procedure

A routine that defines executable code and data that executes from and returns control to a calling routine.

process

A functionally independent entity that provides the execution context for a program image or part of a program image.

process block

A routine that defines executable code and data available to one or more dynamically created processes.

process state

The current state of a process. While active, a process is always in one of four states: run, ready, wait, or suspend.

program

One or more compilation units, each of which is a source file that you can compile to produce an object module.

program block

The main section of a VAXELN Pascal program's code that you declare with the reserved word PROGRAM and that executes as a job. The C equivalent is a program's main routine.

program region

See P0 virtual address space.

pseudodevice

A virtual device. The VMS operating system supports drivers for pseudodevices, including the null device (NL:), network device (NET:), remote terminal device (RT:), and mailbox (MB:). You can assign channels to these devices and issue I/O requests just as though they were real devices.

QIO (queued input/output)

A VMS operating system term for queued input/output using the SYS$QIO system service routine.

queue

A list of items to be processed in a first-in-first-out (FIFO) order. In the VAXELN context, queues provide an efficient, highly structured means for a job's processes to exchange large packets of information.

ready state

The state during which a process is not executing but is ready to execute as soon as the VAXELN kernel scheduler allows.

realtime application

An application that provides interfaces for equipment, such as input/output (I/O) devices and busses that collects and, in some cases, controls the processing of data. Realtime applications must respond to events generated by equipment within a predetermined time limit.

realtime timers

In a POSIX environment, per-process timers for synchronizing and scheduling realtime events.

remote debugger environment

The VAXELN debugger environment that you can use from the remote host system if you have a DECnet license and the appropriate Ethernet hardware. This debugger environment provides a symbolic interface.

rendezvous

In the Ada context, the interaction that occurs between two parallel Ada tasks. During this interaction, one task calls an entry of the other task, and a corresponding **accept** statement is executed by the other task on behalf of the calling task.

rendezvous performance

In the Ada context, a measure of the time it takes for a rendezvous between two Ada tasks to complete. *See also* rendezvous.

repository

An evolutionary extension of data dictionary systems. A repository takes the facilities of the dictionary for data integration and adds the control services (that is, security, versioning, and object management) that facilitate control integration. A repository must allow access to all types of data and data definitions.

routine

A set of instructions that performs an operation.

RS/1 tables

A statistical analysis package used in applications such as research and quality analysis.

RS170

An Electronic Institute of America (EIS) video signal standard that defines a composite video signal as transmitted on a wire.

run state

The state during which a process has control of the processor and is currently executing.

runtime library

The group of common functions and macros that accompany the compiler. Access to the runtime library is by receiving a copy of the function module in the program image or by sharing the function image.

runtime software

In the VAXELN context, system and application programs that run as a realtime system on a VAX computer configuration.

S0 virtual address space

In the VAXELN context, the area of a target processor's virtual memory into which the kernel maps a VAXELN system image: the kernel, program, and shareable runtime images.

scheduling policy

In a POSIX environment, the set of rules that governs how the scheduler selects runnable processes, how processes are queued, and how much time each process is given to run. POSIX supports multiple, user-specified scheduling policies, including fixed-priority and round-robin.

semaphore

A synchronization gate that controls access to a shared resource. Semaphores can be binary or counting.

server

A program that controls specific devices. For example, a DECwindows server controls workstation devices such as screens, keyboards, and pointers.

service node

See VAXELN service node.

session

In the VAXELN context, a logical connection between a terminal device attached to a terminal server and a service offered by a VAXELN service node.

shared memory

Shared memory allows for fast communication between processes that share portions of their virtual address space.

SMP (symmetric multiprocessing)

See symmetric multiprocessing (SMP).

subprocess

A subsidiary process that a job's master process or another subprocess creates dynamically.

suspend state

In the VAXELN context, the state during which a process cannot reenter the ready state until another process in the same job reactivates the suspended process.

In the VMS context, a synchronization mechanism (accessible through a VMS system service call) that allows a process to control when it or another process becomes active. A process can place other processes into the suspend state. Suspension temporarily halts process execution. *See also* hibernate state.

symbolic debugging

The use of debug symbol table information provided by VMS compilers (such as variable names, labels, and source-line information) during a debugging session.

symmetric multiprocessing (SMP)

A computer configuration in which multiple processors can share common resources equally.

system image

See VAXELN system.

system region

See S0 virtual address space.

target processor

In the VAXELN context, the VAX computer on which a VAXELN system image runs.

TCP/IP

Transport Control Protocol/Internet Protocol. TCP/IP is a general-purpose networking standard.

terminal emulator

A program that displays a window that looks and functions like a terminal.

throughput

The rate at which a computer system computes data.

tightly coupled symmetric multiprocessing

A computer configuration in which multiple processors are linked by a CPU memory data path or bus.

timers

See realtime timers.

transparent networking

See distributed processing.

TRI/ADD Program

Third Parties with Add-On Products for RISC UNIX Platforms. The TRI/ADD Program provides technical and marketing support worldwide to third-party vendors using the SCSI, TURBOchannel, VME, ACCESSbus, and Futurebus+ interconnects to develop add-on products for open systems.

UIS (User Interface Services)

A library of shared routines and commands supported by VWS (VMS Workstation Software). These routines can be used for manipulating graphics using world coordinates.

UISX (User Interface Services X)

A library of routines that emulate UIS calls on an eight-plane VMS DECwindows workstation or X Window terminal.

universal port name

A port name known to processes and jobs on all nodes in the local area network (LAN). Universal names are the key to distributed applications.

UNIX kernel

The operating system on a UNIX machine. Can also be used to refer to ULTRIX, DEC OSF/1, or DECelx operating systems.

utility

A program that provides a set of related, general-purpose functions, such as a command language or error logging utility.

VAX calling standard

A software standard that enables full access to all of the VMS operating system's services and procedures through high-level language extensions.

VAXELN service node

A VAXELN system on which the local area transport (LAT) driver executes.

VAXELN system

A set of user and Digital program images that comprise a VAXELN application and execute on VAX target hardware.

virtual address

A memory address within the 4-gigabyte virtual address space available to a program.

virtual address space

A set of theoretical virtual addresses that a program can occupy, which is limited by the memory addressing hardware on which the program executes. The virtual address space for VAX computers is 4 gigabytes.

virtual circuit

A communication path between two connected ports over which messages are transmitted.

virtual memory

The set of storage locations in physical memory to which virtual addresses correspond. For example, in VAXELN, the size of a system's virtual memory depends on the amount of available physical memory.

virtual memory (VM) driver

A device driver program that lets you create a virtual disk structure in system memory and use the memory as you would an actual disk drive.

wait state

The state during which a process is waiting for a specified set of conditions to be satisfied. A process might be waiting for an amount of time to elapse, an event or series of events to occur, or the receipt of a message.

window server

A system that displays DECwindows application interfaces while taking advantage of the resources and power of the host system.

WYSIWYG

WYSIWYG (what you see is what you get) refers to an editor that allows the editing, formatting, and viewing of a document, on a monitor screen, exactly as it will appear when the document or screen is printed.

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825) and press 2 for technical assistance.

Electronic Orders

If you wish to place an order through your account at the Electronic Store, dial 800-234-1998, using a modem set to -2400 or -9600 baud. You must be using a VT terminal or terminal emulator set at 8 bits, no parity. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825) and ask for an Electronic Store specialist.

Telephone and Direct Mail Orders

From	Call	Write
U.S.A.	DECdirect Phone: 800-DIGITAL (800-344-4825) FAX: (603) 884-5597	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	Phone: (809) 781-0505 FAX: (809) 749-8377	Digital Equipment Carribean, Inc. 3 Digital Plaza, 1st Street Suite 200 Metro Office Park San Juan, Puerto Rico 00920
Canada	Phone: 800-267-6215 FAX: (613) 592-1946	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	————	Local Digital subsidiary or approved distributor
Internal Orders[1] (for software documentation)	DTN: 241-3023 (508) 874-3023	Software Supply Business (SSB) Digital Equipment Corporation 1 Digital Drive Westminster, MA 01473
Internal Orders (for hardware documentation)	DTN: 234-4325 (508) 351-4325 FAX: (508) 351-4467	Publishing & Circulation Services Digital Equipment Corporation NR02–2 444 Whitney Street Northboro, MA 01532

[1]Call to request an Internal Software Order Form (EN–01740–07).

Reader's Comments

Realtime Products Technical Summary
EK–RPTSS–TM–004

Your comments and suggestions help us improve the quality of our publications.
Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (product works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

What I like best about this manual is ______________________________

____

What I like least about this manual is ______________________________

____

I found the following errors in this manual:

Page	Description

Additional comments or suggestions to improve this manual:

____

____

____

For software manuals, please indicate which version of the software you are using: ______

____

Name/Title ______________________ Dept. ____________

Company ______________________ Date ____________

Mailing Address ______________________________

______________________ Phone ____________

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
If Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Information Design and Consulting
NUO1-1/G10
55 NORTHEASTERN BLVD.
NASHUA, NH 03062-9934

Do Not Tear - Fold Here