

CECOM

CENTER FOR SOFTWARE ENGINEERING
ADVANCED SOFTWARE TECHNOLOGY

AD-A223 086

DTIC
ELECTE
JUN 21 1990
S E D
Co

Subject: **Final Report - Guideline to Select,
Configure, and Use an Ada Runtime
Environment**

CLEARED
FOR OPEN PUBLICATION
MAY 2 - 1990
DIRECTORATE FOR FREEDOM OF INFORMATION
AND SECURITY REVIEW (OASD-PA)
DEPARTMENT OF DEFENSE

CIN: C02 092LA 0001

15 FEBRUARY 1989

This document has been approved
for public release and its
distribution is unlimited.

90 002010

90 06 21046

GUIDELINE TO SELECT, CONFIGURE, AND USE

AN ADA RUNTIME ENVIRONMENT

FINAL REPORT



PREPARED FOR:

U.S. Army HQ CECOM
Center for Software Engineering
Advanced Software Technology
Fort Monmouth, NJ 07703-5000

PREPARED BY:

LabTek Corporation
8 Lunar Drive
Woodbridge, CT 06525

DATE:

30 September 1988

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/ _____	
Availability Codes	
Dist	Avail and/or Special
A1	

Ada-86 is a trademark of SofTech Inc.

ARTK is a trademark of Alsys.

DDC-I Ada Compiler System and DACS-80x86 are trademarks of DDC-I, Inc.

DEC, VAX VME, VAXELN, MicroVAX and VMS are trademarks of Digital Equipment Corp.

IBM is a trademark of International Business Machines Corp.

Intel and iSBC, ASM86, LIB86, LINK86, and LOC86 are trademarks of Intel Corp.

M68000, MC68881, MC68010, and MC68020 are trademarks of Motorola Corp.

Sun Workstation is a registered trademark of Sun Microsystems, Inc.

TeleSOFT and TeleGen 2 are trademarks of TeleSOFT.

UNIX is a trademark of Bell Laboratories.

VADS is a registered trademark of VERDIX Corp.

VRTX, VRTX32 are trademarks of Ready Systems.

(4)

EXECUTIVE SUMMARY

The Ada Language has incorporated many features such as tasking, dynamic storage management, and exception handling that require substantial execution-time support. Most of these features were not previously available in commonly used real-time languages, but were instead provided by an separate "executive". The inclusion of these features into the language expands the possibility for transportable and reusable software, but complicates the software development process to some degree. Engineers that previously had familiarity with their own executives, now are forced to accept the code of a compiler vendor for the execution-time support. This guide has been written to help software developers in the difficult task of selecting, configuring, and using a runtime that will meet the needs of their application.

The wide variety of applications for which Ada is used necessitates considerable flexibility within the implementation of the runtime code. Different algorithms for tasking, storage management, interrupt handling, and exception propagation can radically effect the behavior of real-time programs. Variations among compilers for the same processor can be as great as a factor of six in runtime size and a factor of eleven in tasking performance. It is therefore essential that software developers completely understand the characteristics of the available runtimes prior to selecting one for use on a project. Due to the cost and time involved in a compiler procurement (runtime source code can cost as much as \$250,000), it is often difficult to change to a new compiler implementation after a poor choice has been made. Unfortunately, a compiler that is good for one application may not necessarily be proper for other applications. Therefore, it is more a matter of matching a compiler implementation to an application rather than simply finding "the best compiler".

158

This guide lists all of the known (validated) Ada compilers that are developed for use in embedded applications. For each compiler, the supplier was contacted and asked specific questions about their implementation. As much information as possible was obtained from the suppliers to be summarized in the report. Performance benchmarks are included as a rough efficiency comparison among many of the implementations. The difficulty in obtaining this information cannot be overstated. Frequent letters with follow-up phone calls were necessary to obtain answers to even a few questions. Just as with the compiler implementations, substantial variation exists among compiler vendors in their willingness to provide detailed literature. The effort that went into collecting this information convinced the researchers that such a guide was worthwhile. For each individual project to go through a collection effort is a tremendous expenditure of effort and is unlikely to be as complete as this guide.

Finally, guidance is provided on how to proceed with the selection process, what questions to ask once the choice of compilers is narrowed down to one or two, and what to do after the compiler and runtime have been selected. Special attention is paid to areas that experience has shown to be particularly troublesome. These include:

- 1.) Maintaining configuration control over variations in the runtime; insuring that new device drivers which are configured into the runtime do not violate runtime conventions, especially with the processor state (privilege, interrupt level, memory management registers, etc.); and,
- 2.) Taking care not to extend the worst-case interrupt latency by allowing interrupts to be disabled for an extended period.

These types of problem usually do not manifest themselves in obvious ways, but rather result in working but unreliable systems. They may pass the acceptance testing and operate properly for months only to fail in a catastrophic fashion during a critical moment.

It is hoped that this guide will assist software developers through some of the problems in adopting Ada for real-time embedded projects. By providing information on how Ada implementations operate, there will be a reduction in the uncertainty associated with switching from assembly language executives, where every aspect is provided in minute detail, to Ada where the executive functions appear as a black box (or magic).

Table of Contents

1. Introduction	1
1.1 Background	1
1.2 Purpose and Intent	1
1.3 Definitions	1
1.4 Organization of Document	3
1.5 How to Best Utilize This Document	4
2. Approach	6
3. Ada Runtime Features	8
3.1 Dynamic Memory Management	8
3.2 Processor Management	8
3.3 Interrupt Management	9
3.4 Time Management	9
3.5 Exception Management	9
3.6 Rendezvous Management	9
3.7 Task Activation	9
3.8 Task Termination	9
3.9 I/O Management	10
3.10 Commonly Called Code Sequences	10
3.11 Target Housekeeping Functions	10
4. Bare Machine Targets	11
4.1 PIWG Benchmarks	13
4.2 Vendor Address Listing	17
Table I. Bare Machine Targeted Compilers by Processor	18
1750A	18

Table of Contents

80x86	19
680x0	22
32032	25
VAX processors	25
CAPS/AAMP	26
PowerNode	26
Table II. Bare Machine Targeted Compilers by Vendors	27
Vendor Supplied Information	28
AITech Software Engineering, Ltd.	28
Alsys	36
CAP Industry, Ltd.	59
DDC-I	65
Digital Equipment Corp.	81
Gould, Inc.	83
Intermetrics, Inc.	85
Rational	91
SofTech, Inc.	126
System Designers Software, Inc.	148
Tartan Laboratories, Inc.	162
TeleSoft, Inc.	174
TLD Systems, Ltd.	189
Verdix Corp.	196
Vendor Information Not Available	219
Advanced Computer Techniques Corp.	219
Harris Corp.	220
Rockwell International	221

Table of Contents

System Designers Software, Inc.	222
TeleSoft/Intel Corp./TeleLOGIC	223
TeleSoft, Inc.	224
Verdix Corp.	227
5. Application Characteristics	229
5.1 Electronic Warfare	230
5.1.1 Radar Systems	230
5.1.2 Electronic Counter Measures (ECM)	231
5.1.3 Signal Processing	231
5.2 Weapon Guidance	231
5.3 Fire Control	232
5.4 Simulation Systems	232
5.5 C3I Systems	232
5.6 Operating Systems	232
5.7 Navigation Systems	233
5.8 Artificial Intelligence	233
5.9 Robotics/Process Control	233
6. Guidelines	234
6.1 To Select a Runtime Environment	234
6.1.1 Documentation	234
6.1.2 Degree of Configurability	234
6.1.3 Chapter 13	235
6.1.4 Appendix F	235
6.1.5 Target Dependent Information	236
6.1.6 Target Initialization	236
6.1.7 Target I/O	236

Table of Contents

6.1.8 Target Timer	237
6.1.9 Data Representation	237
6.1.10 Implementation of Tasking	238
6.1.11 Interrupt/Handler/Interrupt Vectors	239
6.1.12 Storage Management	239
6.1.13 Subroutine Call and Parameter Passing Conventions	240
6.1.14 Saving Machine State During a Context Switch	240
6.1.15 Exception Handling	240
6.1.16 Unhandled Exceptions	241
6.1.17 Generics	241
6.1.18 I/O Interfaces	242
6.1.19 Compiler Capacity and Tool Availability	242
6.2 To Configure a Runtime Environment	243
6.2.1 Bootstrapping	243
6.2.2 Interrupt Vector	244
6.2.3 User-Configurable Module Dependencies	244
6.2.4 Timer Interrupt	244
6.2.5 Linker Options	244
6.3 To Use a Runtime Environment	245
7. Effects of Runtime Issues on the Development of Reusable Software	247
8. Summary	248
9. References	249
10. Appendix A	252

List of Figures

1. Ada Runtime Environment (RTE)	3
2. Runtime Environment Components	8
3. The Application Domain	230

List of Tables

1. Bare Machine Targeted Compilers (listed by target)	18
2. Bare Machine Targeted Compilers (listed by vendor)	27

Guideline to Select, Configure, and Use an Ada Runtime Environment

1. Introduction

1.1 Background

An extensive effort is underway by the DoD to transition Ada technology into the real-time embedded application domain. Much work has been done to determine *why* the transition to the Ada programming language is not, in actuality, as smooth as originally anticipated. A primary reason for the difficulty, cited in LabTek's 1987 report, titled "*Software Engineering Issues on Ada Technology Insertion for Real-time Embedded Systems*", is the incorporation of a substantial **runtime environment** into the compilation system.

An Ada compilation system, in addition to generating the code for the semantics of the Ada language, also supplies the code that was previously provided by a separate executive or operating system. It provides an extensive *runtime* which other traditional compilers did not. Therefore, application developers, who previously built their own executives, have to sacrifice some of the ability to configure the executive to suit the application when transitioning to Ada. This report will detail the extent of configurability available in Ada runtime environments today.

1.2 Purpose and Intent

The runtime environment of the Ada compilation system must always comply with the rules of the Ada language as defined by the Ada standard, ANSI-MIL-STD-1815A-1983. [5] Yet the Ada standard provides significant flexibility in how the runtime environments support the language definition. The runtime environment is thus allowed to exhibit different performance characteristics (that may reflect the needs of the application) for the same features or combination of features. In fact, Ada provides the pragma construct as one method to help the Ada compilation system determine the performance characteristics that the runtime environment should provide for an application. Thus, the runtime environment of an Ada compilation system may be able to accommodate an arbitrary number of interpretations of an application in Ada that comply with the Ada language standard. These interpretations can be guided by the pragma construct or by other mechanisms provided by the Ada compilation system.

It is the purpose and intent of this report to produce a guideline to select, configure and use an Ada runtime environment. It will detail the options available to application developers who must contend with Ada runtime environments. A view of the current state of the technology for bare machines will be presented.

1.3 Definitions

Following are the definitions for terms found throughout this report.

ARTEWG: The Ada RunTime Environment Working Group, is a group sponsored by the Association for Computing Machinery (ACM), Special Interest Group for Ada (SIGAda), whose purpose is to address the problems encountered in Ada runtime environments.

AVO: The Ada Validation Organization provides administrative and technical support to ensure that Ada compilers faithfully implement the Ada programming language standard (ANSI/MIL-STD-1815A-1983). [4]

Guideline to Select, Configure, and Use an Ada Runtime Environment

Base Compiler: An Ada compiler for which a current validation certificate exists. [4]

Base Configuration: The specific configuration on which the base compiler is tested by an Ada Validation Facility (AVF) as part of the validation process. [4]

Ada Compiler: A system (in a loadable or executable code form) which translates Ada source programs into object code that, when loaded with the target run-time system, executes on a target computer in a manner that is in compliance with the Ada programming language. [4] Throughout this report the phrase *compilation system* will be used synonymously.

Configure an RTE: To configure an RTE is the ability to select various software components when building the application software. Components may be selected from the following categories: dynamic memory management, processor management, interrupt management, time management, exception management, rendezvous management, task activation, task termination, I/O management, and miscellaneous support functions. Configuring an RTE is different than tailoring an RTE (see tailor an RTE).

Derived Compiler: One of the following:

1. A base compiler on an equivalent configuration.
2. A maintained compiler on a base configuration.
3. A maintained compiler on an equivalent configuration, where any of these pairs originates from a base compiler and base configuration pair. [4]

Equivalent Configuration: Any configuration of the same computer architecture(s) and operating system for which compliance is achievable using the same ACVC (Ada Compiler Validation Capability) version used in the validation of the base compiler on the base configuration. [4]

Host Architecture: The computer architecture on which the compiler resides.

Maintained Compiler: A base compiler which has been changed in any way generally accepted by the software profession to constitute "maintenance" - usually meaning minor change. Complete replacement or addition of some major component of a base compiler is not considered "maintenance". [4]

PIWG: The Performance Issues Working Group, is a group sponsored by the Association for Computing Machinery (ACM), Special Interest Group for Ada (SIGAda), whose purpose is to write benchmark programs which can be executed on different Ada compilation systems and provide performance information.

Runtime Environment (RTE): Consists of three functional areas: abstract data structures, code sequences, and predefined subroutines. It includes all of the runtime support routines, the conventions between the runtime routines and the compiler, and the underlying virtual machine of the target computer. "Virtual" is used in the sense that it may be a machine with layered software (a host operating system). An RTE does not include the application itself, but includes everything the application can interact with. Each layer has a protocol between it and the layer underneath it for interfacing. In the event that there isn't any

Guideline to Select, Configure, and Use an Ada Runtime Environment

operating system layer (the *bare-machine target*), the runtime includes those low-level functions found in an operating system. See Figure 1.

Tailor an RTE: To tailor an RTE is the actual modification of the source code to achieve the requirements of the application.

(Target) Runtime System or Runtime System (RTS): The set of subprograms, which may be invoked by linking, loading, and executing object code generated by an Ada compiler. If these subprograms use or depend upon the services of an operating system, then the target runtime system includes those portions of that operating system. [4] These predefined subroutines are chosen from the *Runtime Library* for that Ada compilation system.

Target Architecture: The computer architecture used for execution of object code generated by an Ada compiler. [4]

VAXELN is a real-time operating system for DEC VAX line of computers.

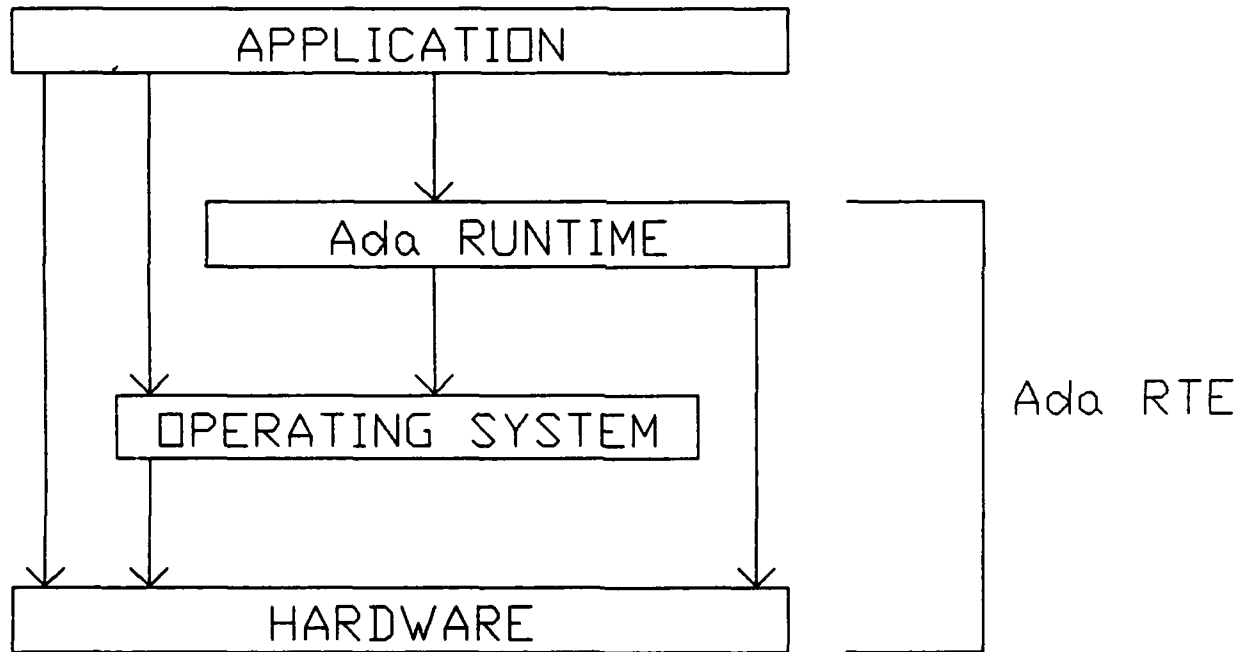


Figure 1. Ada Runtime Environment (RTE)

1.4 Organization of Document

Section one of this report contains the introductory information as well as the definitions of terms found in the report.

Section two of this report details the approach used to gather the information and the criteria used for its evaluation.

Section three of this report details the components of a runtime environment. The reader is referred to the document "*A Framework for Describing Ada Runtime Environments*",

Guideline to Select, Configure, and Use an Ada Runtime Environment

proposed by the ARTEWG, October 15, 1987. [2] This document details the evolution of runtime environments, and provides a taxonomy of the components of a runtime environment.

Section four of this report provides a complete list of the Ada compilation systems that were available for bare machine targets at the time of this writing. For each implementation it contains: 1) the degree of configurability of the runtime, 2) storage requirements of the runtime system, and 3) efficiency information.

Section five of this report categorizes the application domain into distinct areas. For example, a C3I application requires different runtime features than a signal processing application. The purpose of section five is to subdivide the application domain and detail the runtime features needed for each subdivision.

Section six of this report provides detailed guidance for selecting, configuring and using Ada runtime environments.

Section seven details the effects that runtime issues will have on the development of reusable software for Mission Critical Computer Resources (MCCR) applications.

Section eight contains a summary of lessons learned.

Section nine contains the reference materials used in the creation of this report.

Appendix A of this report contains two versions of the *"Survey of Runtime Environment Components"*. These surveys were used to obtain information about the bare machine target compilation systems from the compiler vendors. Throughout the period of performance of this contract the survey was fine-tuned, thus producing a second version.

1.5 How to Best Utilize this Document

This guideline can be utilized as a reference guide or as a process for selecting an Ada runtime environment.

For quick reference guide usage, turn to Table I, titled "Bare Machine Targeted Compilers". This table details what compilers are available for the target of interest. Refer to the pages listed in the right column for details on those implementations. A table of compiler vendor names and addresses (with phone numbers) is also provided in section 4.2. Please consult with the compiler vendors to answer any additional question you may have regarding a specific implementation.

To use this guideline as a process for selecting an Ada runtime environment, the following is a suggested method:

- 1.) Determine your system requirements. Review section five of this report, titled "Application Characteristics" for a general description of the requirements that can be imposed upon the application software.
- 2.) Review section three of this report, titled "Ada Runtime Features". For the most part, runtime environments can be broken down into these components and it provides a basis for further discussion.

Guideline to Select, Configure, and Use an Ada Runtime Environment

- 3.) Review section six, titled "Guideline to Select, Configure, and Use a Runtime Environment". This section contains the questions to ask before selecting a specific implementation. The list should be fine-tuned for the particular application.
- 4.) Review section four for details on specific compiler implementations.
- 5.) Contact the compiler vendor (see section 4.2 for phone numbers) to resolve any additional questions you may have regarding a specific implementation. If appropriate, purchase the documentation only for the compiler of interest and review it before making a commitment to use a particular runtime implementation.

Guideline to Select, Configure, and Use an Ada Runtime Environment

2. Approach

The approach used to obtain the information in this report was:

1. The current literature, especially the ARTEWG documents, was reviewed for material relevant to this task. [2], [3]
2. A comprehensive list of the validated bare machine target compilers was produced. [6], [18], [22]
3. A *Survey of Runtime Environment Components* was prepared to obtain pertinent runtime information for the compilers of interest (determined in step two above). A copy of this survey can be found in Appendix A. The survey was updated on an iterative basis. As vendors/users responded, it was fine-tuned and used from that point onward. The final version (V2.0) can also be found in Appendix A.
4. The compiler vendors were contacted and asked to respond to the survey produced in step 3 above.

The survey was concerned with obtaining the following information: a.) degree of configurability, b.) the storage requirements (overhead) associated with using a particular runtime feature, and c.) performance information.

Since most compiler vendors had the PIWG benchmarks available for their products, they were asked to supply the results, along with the speed of the processor and wait-state of the memory. The intent was to provide performance information that could be compared and contrasted.

5. The AVO was contacted in order to obtain copies of the validation report summaries for each bare machine target. Of particular interest was the Implementation Dependent Characteristics (Appendix F of the Ada Reference Manual) and the Language Features Supported section. This turned out not to be as useful as originally expected, for two reasons: 1.) It was not easy to obtain copies of the validation reports. It had to be ordered through NTIS (National Technical Information Services), which was backlogged, and did not (at the time) have copies of the recently validated compilers ready for distribution. Typically there was a one year lag between the time a compiler was validated, and the time the validation report was available through NTIS. 2.) The validation reports do not contain *configurability* information.

6. Compiler documentation, for a few selected compilers (Tartan Laboratories, Systems Designers Software, DDC-I, and Verdix), was reviewed for usefulness and completeness. Special attention was paid to the sections describing runtime configurability. Some vendors were not selected because a.) they would not sell the documentation separately without licensing the compiler, or b.) the cost for documentation exceeded our guidelines for purchase of it.

7. The Info-Ada bulletin was utilized to obtain a database of users who could provide the necessary information when gaps existed in vendor supplied information.

8. A mailing to a large group of people (approximately 360) concerned with Ada runtimes was performed. The purpose was to see if anyone had specific information regarding a bare

Guideline to Select, Configure, and Use an Ada Runtime Environment

machine implementation and could provide input into this report. Those who respond favorably were contacted either by electronic mail or phone and sent a survey.

9. The ARTEWG meetings and the SIGAda meetings, which fell during the period of performance of this contract, were attended by LabTek personnel. Informal interviews were held, contacts were made and surveys were distributed.

10. The input material obtained from steps 1-9 above was analyzed, and this report was produced.

Guideline to Select, Configure, and Use an Ada Runtime Environment

3. Ada Runtime Features

This section contains a taxonomy of Ada runtime environment components (see Figure 2.) with a description of each. Again, the reader is referred to the ARTEWG document, *A Framework for Describing Ada Runtime Environments*. The taxonomy is provided here to clarify the components referenced in the size breakdown of each runtime in section 4.

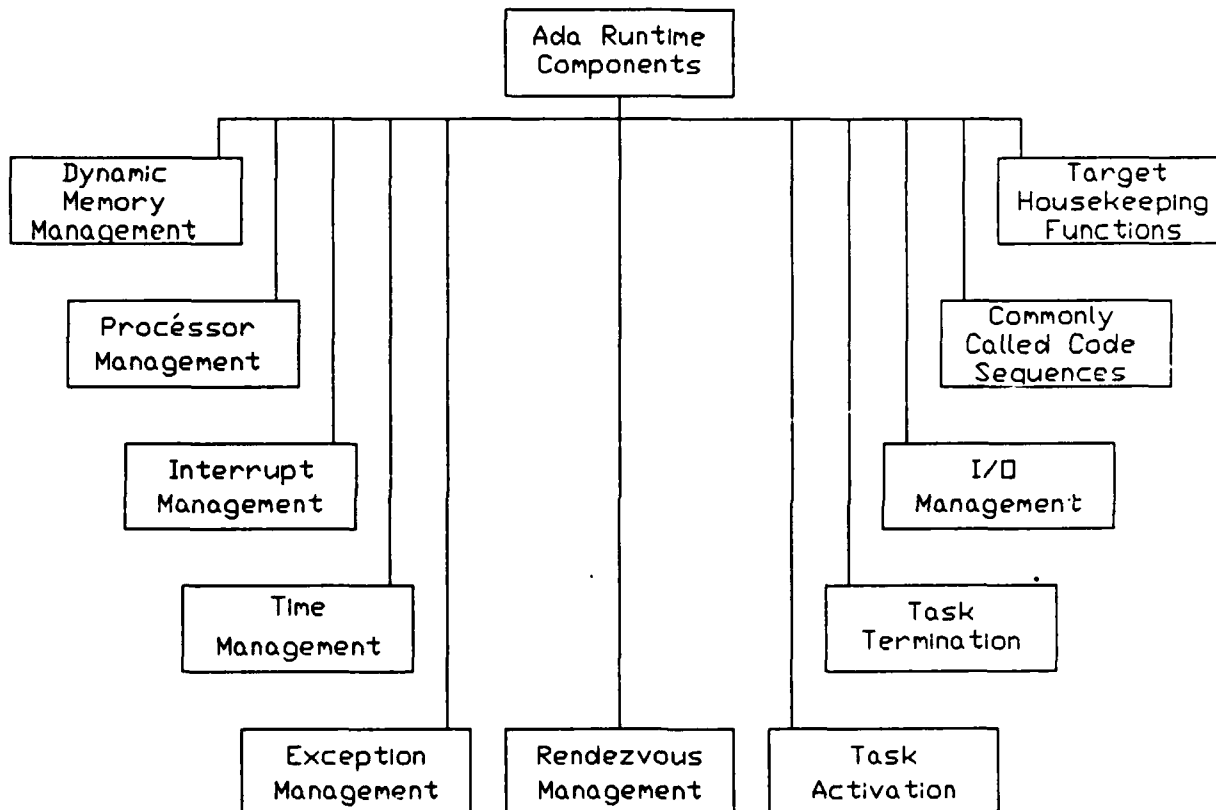


Figure 2. Runtime Environment Components

3.1 Dynamic Memory Management

Dynamic Memory Management is responsible for allocation and deallocation of storage at runtime. It also detects when a request for storage cannot be fulfilled, and for raising the exception `STORAGE_ERROR` as appropriate.

3.2 Processor Management

Processor Management implements the assignment of the CPU (or CPUs) to tasks that are "logically executing". The processor management function is invoked by other components of the runtime environment, in order to block and unblock tasks. It keeps a list of those

Guideline to Select, Configure, and Use an Ada Runtime Environment

tasks which are "logically executing" and uses this list, in conjunction with the priorities of tasks, to select which task (or tasks) should physically execute. This component is often called the "scheduler".

3.3 Interrupt Management

Interrupt Management is responsible for initialization of the interrupt mechanism of the underlying computing resource, and it is also responsible for resetting that mechanism after an interrupt has occurred, if the architecture of the underlying computing resource requires such resetting.

3.4 Time Management

Time Management consists of all those portions of the runtime environment that will support the predefined package CALENDAR and the implementation of delay statements. If the underlying computing resource offers enough functionality, the support of package CALENDAR is trivial.

3.5 Exception Management

Exception Management implements Ada semantics for exceptions: that is, it determines whether there is a matching handler for the exception at hand, and if there is one, it transfers control to the handler. If there is no matching handler, it invokes the Task Termination function to terminate the task at hand or the main program.

3.6 Rendezvous Management

Rendezvous Management implements the semantics of the Ada rendezvous model. In order to do so, it utilizes variables that are internal to the runtime environments. These variables reflect, among other things, which tasks are blocked because they are waiting to rendezvous with other tasks, and what the exact circumstances of these wait states are. The rendezvous management function cooperates with the interrupt management function in the implementation of interrupt rendezvous, if the interrupt rendezvous is supported by the runtime environment.

3.7 Task Activation

At some point after the task object has been created, the execution of the new task has to be started. This is effected by the task activation function. This function is invoked by the creator of a new task in order to start the new task's activation (which is defined as the execution of the declarative part of the task's body). It may also be invoked by the new task in order to signal the completion of that task's activation.

3.8 Task Termination

Task Termination implements the set of rules for the completion, termination, and abortion of tasks.

Guideline to Select, Configure, and Use an Ada Runtime Environment

3.9 I/O Management

I/O Management consists of all those portions of the runtime environment that are provided for the support of input and output. This includes in particular all those functions that support predefined packages from Chapter 14 of the Ada Reference Manual.

3.10 Commonly Called Code Sequences

Commonly Called Code Sequences is a "catchall" category. It includes runtime routines in the classical sense: commonly called sequences of code. Typical examples are operation for multi-word arithmetic, block moves and string operations. Ada attribute calculations also fall into this category.

3.11 Target Housekeeping Functions

Target Housekeeping Functions are associated with the start up and termination of the execution environment of an Ada program. Such actions include determination of the particular hardware and software execution environment, setting of variables identifying same, processor and interrupt initializations, and so on. Similarly, if a program terminates, control is typically returned to some surrounding software whose state must be reset upon program exit.

Guideline to Select, Configure, and Use an Ada Runtime Environment

4. Bare Machine Targets

There are currently seventy-one validated Ada compilers which generate code for the bare machine target. These compilers are produced by eighteen vendors.

The information in this chapter will be provided in two formats:

- 1.) by Target Processor Type (Table I). This section, *indexed by processor type*, contains a page reference to Table II, where the detailed information for that compilation system can be found.
- 2.) by Compiler Vendor (Table II). This section, *indexed by compiler vendor*, details the host/target combination, degree of configurability, PIWG benchmark results, storage requirements in graphical form, Package SYSTEM, Package STANDARD, and the vendor responses to pertinent questions that were considered critical to real-time programming. The information contained in Table II is explained more fully beginning on page 27.

In addition to the two tables described above, chapter 4 contains a brief description of the PIWG benchmarks (4.1), and a list of vendor contacts (4.2).

Guideline to Select, Configure, and Use an Ada Runtime Environment

This page intentionally left blank.

Guideline to Select, Configure, and Use an Ada Runtime Environment

4.1 PIWG Benchmarks

Benchmarks are used as part of the compiler selection process. Therefore, the Performance Issues Working Group (PIWG) benchmarks are supplied in Table II. The following is a description of the PIWG benchmarks taken from the PIWG test suite itself.

TEST	DESCRIPTION
A000091	DHRYSTONE Benchmark. Contains Ada statements in a distribution considered representative: 53% assignments, 32% control statements, 15% procedures, function calls. 100 statements are dynamically executed. The program is balanced with respect to the three aspects: statement type, operand type (for simple data types), and operand access (operand global, local, parameter, or constant). The combination of these three aspects is balanced only approximately. All variables have a value assigned to them before they are used as a source operand.
A000093	WHETSTONE Benchmark. Ada version of the Whetstone Benchmark Program. Reference: "Computer Journal", February 1976, pages 43-49 for description of benchmark and ALGOL60 version. Note: Procedure POUT is omitted.
C000001	Task create and terminate measurement, with one task, no entries, when task is in a procedure, using a task type in a package, no select statement, no loop.
C000002	Task create and terminate time measurement, with one task, no entries when task is in a procedure, task defined and used in procedure, no select statement, no loop.
C000003	Task create and terminate measurement. Task is in declare block of main procedure, one task, no entries, task is in the loop.
D000001	Dynamic array allocation, use and deallocation time measurement. Dynamic array elaboration, 1000 integers in a procedure, get space and free it in the procedure on each call.
D000002	Dynamic array elaboration and initialization time measurement, allocation, initialization, use and deallocation, 1000 integers initialized by others greater than 0, equal to one.
D000003	Dynamic record allocation, and deallocation time measurement, elaborating, allocating and deallocating record containing dynamic array of 1000 integers.
D000004	Dynamic record allocation, and deallocation time measurement, elaborating, initializing by (Dynamic_Size,(others = > 1)) record containing a dynamic array of 1000 integers.

Guideline to Select, Configure, and Use an Ada Runtime Environment

PIWG BENCHMARKS (Continued)

TEST	DESCRIPTION
E000001	Time to raise and handle an exception. The exception is defined locally and handled locally.
E000002	Exception raise and handle timing measurement when exception is in a procedure in a package.
E000003	Exception raise and handle timing measurement, when exception is raised nested three deep in procedure calls.
E000004	Exception raise and handle timing measurement, when exception is nested four deep in procedures.
E000005	Exception raise and handle timing measurement when exception is in a rendezvous. both the task and the caller must handle the exception.
F000001	Time to set a boolean flag using a logical equation. A local and a global integer are compared. compare this test with F000002.
F000002	Time to set a boolean flag using an "if" test. A local and global integer are compared. Compare this test with F000001.
G000005	TEXT_IO.Get an INTEGER from a local string, timing measurement. Use TEXT_IO to convert 1..100 to a string, then use TEXT_IO.GET to get the number back.
G000006	TEXT_IO.Get getting a floating point fraction from a local string. Timing measurement on .001 to .01 range of numbers. Compare, approximately, to G000005 for INTEGER vs. FLOAT.
H000001	Time to perform standard BOOLEAN operations on arrays of BOOLEAN. For this test the arrays are PACKED with the pragma PACK. The operations are performed on the entire array.
H000002	Time to perform standard BOOLEAN operations on arrays of BOOLEAN. The arrays are not PACKED with pragma PACK. The operations are performed on the entire array.
H000003	Time to perform standard BOOLEAN operations on arrays of BOOLEAN. The arrays are PACKED with the pragma PACK. The operations are performed on components in a loop.
H000004	Time to perform standard BOOLEAN operations on arrays of BOOLEAN. The arrays are not PACKED with the pragma PACK. The operations are performed on components in a loop.

Guideline to Select, Configure, and Use an Ada Runtime Environment

PIWG BENCHMARKS (Continued)

TEST	DESCRIPTION
H000005	Time to move one INTEGER object to another INTEGER object using UNCHECKED_CONVERSION. This may be zero with good optimization.
H000006	Time to move 10 floating point array objects to a 10 component floating point record using UNCHECKED_CONVERSION.
H000007	The time to store and extract bit fields that are defined by representation clauses using both BOOLEAN and INTEGER record components. Consists of twelve accesses, five stores, one record copy.
L000001	Simple "for" loop time. For I in 1..100 loop. Time is reported for once through loop.
L000002	Simple "while" loop time. While I is less than or equal to 100 loop. Time is reported for once through the loop.
L000003	Simple "exit" loop time. Loop I:=I + 1; exit when I greater than 100; end loop; Time is reported for once through the loop.
L000004	Measure the compilers' choice to UNWRAP a small loop of five iterations when given the pragma OPTIMIZE(Time). An execution time less than .05 microseconds indicates the unwrap occurred.
L000005	Measure the compilers' choice to UNWRAP a small loop of five iterations when given the pragma OPTIMIZE(Space). An execution speed less than .05 microseconds indicates the unwrap occurred.
P000001	Procedure call and return time (may be zero in automatic inlining). Procedure is local with no parameters.
P000002	Procedure call and return time. Procedure is local with no parameters, when procedure is not inlineable.
P000003	Procedure call and return time measurement. Procedure is in a separately compiled package. Compare to P000002.
P000004	Procedure call and return time measurement. Procedure is in a separately compiled package. Pragma INLINE used. Compare to P000001.
P000005	Procedure call and return time measurement. Procedure is in a separately compiled package. One parameter, in INTEGER.

Guideline to Select, Configure, and Use an Ada Runtime Environment

PIWG BENCHMARKS (Continued)

TEST	DESCRIPTION
P000006	Procedure call and return time measurement. Procedure is in a separately compiled package. One parameter, out INTEGER.
P000007	Procedure call and return time measurement. Procedure is in a separately compiled package. One parameter, in out INTEGER.
P000010	Procedure call and return time measurement. Ten parameters, in INTEGER. Compare to P000005.
P000011	Procedure call and return time measurement. Twenty parameters, in INTEGER. Compare to P000005, P000010.
P000012	Procedure call and return time measurement. Ten parameters, in MY_RECORD, a three component record. Compare with P000010 (discrete vs. composite parameters).
P000013	Procedure call and return time measurement. Twenty composite 'in' parameters, the composite type is a three component record.
T000001	Minimum rendezvous, entry call and return time measurement. One task, 1 entry, task inside procedure, no select.
T000002	Task entry call and return time measurement. One task active, one entry in task, task in a package, no select statement.
T000003	Task entry call and return time measured. Two tasks active, one entry per task, tasks are in a package. No select statement used.
T000004	Task entry call and return time measured. One task active, two entries, tasks in a package, using select statement.
T000005	Task entry call and return time measured. Ten tasks active, one entry per task, tasks in a package, no select statement.
T000006	Task entry call and return time measurement. One task with ten entries, task in a package, one select statement, compare to T000005.
T000007	Minimum rendezvous, entry call and return time measurement, using one task, one entry, and no select statement.
T000008	Measures the average time to pass an integer from a producer task through a buffer task to a consumer task.

Guideline to Select, Configure, and Use an Ada Runtime Environment

4.2 Vendor Address Listing

Advanced Computer Techniques Corp.
(InterACT)
16 East 32nd Street
New York New York 10016
(212) 696 - 3600

AITech Software Engineering Ltd.
1250 Oakmead Parkway
Suite 210
Sunnyvale California 94086
(408) 720 - 9400

Alsys, Inc
1432 Main Street
Waltham, Massachusetts 02154
(617) 890 - 0030

CAP Industry Ltd.
Trafalgar House
Richfield Avenue
Reading Berkshire RG18QA
England
+ 44 734 508961

DDC-I, Inc.
P.O. Box 32220
11024 North 28th Drive
Suite 200
Phoenix, Arizona 85064
(602) 863 - 6910

Digital Equipment Corporation
40 Old Bolton Road
Stow, Massachusetts 01775
(617) 496 - 8740

Gould, Inc.
Computer Systems Division
6901 West Sunrise Boulevard
P.O. Box 9148
Fort Lauderdale, Florida 22210 - 9148
(305) 797 - 5509

Harris Corporation
2101 West Cyress Creek Road
Fort Lauderdale, Florida 33309
(305) 974 - 1700

Intermetrics, Inc.
733 Concord Avenue
Cambridge, Massachusetts 02138
(617) 661 - 1840

Rational
3320 Scott Boulevard
Santa Clara, California 95054-3197
(408) 496-3600

Rockwell International
400 Collins Road North East
Cedar Rapids, Iowa 52498
(319) 395 - 1729

SofTech Inc.
460 Totten Pond Road
Waltham, Massachusetts 02154 - 1960
(617) 890 - 6900

System Designers Software Inc.
101 Main Street
Cambridge, Massachusetts 02142
(617) 499 - 2000

Tartan Laboratories Inc.
461 Melwood Avenue
Pittsburgh, Pennsylvania 15213
(412) 621 - 2210

TeleSoft, Inc
5959 Cornerstone Court West
San Diego, California 92121 - 9891
(619) 457 - 2700

TLD
21235 Hawthorne Boulevard
Suite 204
Torrance, California 90503
(213) 316 - 1516

Verdix Corporation
Sullyfield Business Park
14130 - A Sullyfield Circle
Chantilly, VA 22021
(703) 378 - 7600

Guideline to Select, Configure, and Use an Ada Runtime Environment

Table I. Bare Machine Targeted Compilers

(Listed By Target Processor)

The following table is a list of bare machine targeted compilers which are listed in order by target processor (1750, 80x86, 680x0, 32032, etc.). The table includes the host processor that the compiler executes on and the vendor who produces the compiler. Listed under each vendor are reference pages which refers the reader to the proper pages in Table II concerning the detailed configuration, runtime size, and benchmark information for the corresponding compiler.

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
1750A, ECSPO RAID simulator CX-04.001 (bare machine)	VAX-11/785 (under VMS 4.2)	Intermetrics, Inc. (Ref. pages 85 - 90)
1750A, ECSPO RAID MIL-STD-1750A simulator version 4.0 executing on the host (bare machine)	MicroVAX II (under VMS, version 4.6)	TeleSoft, Inc. (Ref pages 174 - 180)
1750A, Fairchild 9450/1750A in a HP 64000 workstation (bare machine)	VAX-11/785 (under VMS 4.4)	Advanced Computer Techniques Corp. (Ref. page 219)
1750A, Fairchild F9450 (bare machine)	VAX-11/750 (under VMS 4.1)	Tartan Laboratories, Inc. (Ref. pages 162 - 173)
1750A, Fairchild 9450 under Tektronics emulation (bare machine)	MicroVAX II (under VMS Version 4.7)	Verdix Corp. (Ref. pages 196 - 201)
1750A, Ferranti Computer System 100A (bare machine) * Derived *	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.5 or MicroVMS 4.5)	Systems Designers Software, Inc. (Ref. pages 148 - 149)
1750A, Mikros MKS1750/SO (bare machine)	VAX-11/750 (under VMS 4.1)	Tartan Laboratories, Inc. (Ref. pages 162 - 173)
1750A, MIL-STD-1750A (bare machine)	Rational 1000	Rational (Ref. pages 91 - 109)

Guideline to Select, Configure, and Use an Ada Runtime Environment

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
1750A MIL-STD-1750A (bare machine)	VAX-11 VMS	TLD Systems Ltd. (Ref. pages 189 - 195)
1750A MIL-STD-1750A (bare machine)	HP9000 - 350	TLD Systems Ltd. (Ref. pages 189 - 195)
1750A MIL-STD-1750A (bare machine)	DG AOS/VS	TLD Systems Ltd. (Ref. pages 189 - 195)
1750A, Tektronix 8540A (bare machine)	Harris HCX-7 series (under HCX/UX, V.2.2)	Harris Corporation (Ref. page 220)
1750A, Tektronix 8540A (bare machine)	Harris H1200 (under VOS, 6.1)	Harris Corporation (Ref. page 220)
1750A, Unisys S1636- MIL-STD-1750A (bare machine)	VAX-11/750 (under VMS 4.1)	Tartan Laboratories, Inc. (Ref. pages 162 - 173)
8086, Intel iSBC 86/05A (bare machine)	DEC MicroVAX II (under MicroVMS 4.4)	DDC-I (Ref. pages 65 - 80)
8086, Intel iSBC 86/35 (bare machine) *Derived*	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.6 or MicroVMS 4.6)	DDC-I (Ref. pages 65 - 80)
8086, Intel iAPX 8086 (bare machine)	VAX-11/780 and VAX 11/785 (under VAX/VMS 4.5)	SofTech, Inc. (Ref. pages 126 - 147)
8086, Titan SECS 86/20 (bare machine)	DEC MicroVAX II (under MicroVMS 4.4)	DDC-I (Ref. pages 65 - 80)
80186, Intel iAPX 80186 (bare machine)	VAX-11/780 and VAX 11/785 (under VAX/VMS 4.5)	SofTech, Inc. (Ref. pages 126 - 147)
80186, Intel iSBC 186/03A (bare machine)	DEC MicroVAX II (under MicroVMS 4.4)	DDC-I (Ref. pages 65 - 80)
80186, Intel iSBC 186/03A (bare machine) *Derived*	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.6 or MicroVMS 4.6)	DDC-I (Ref. pages 65 - 80)

Guideline to Select, Configure, and Use an Ada Runtime Environment

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
80286 , Intel iAPX 80286 protected mode (bare machine)	MicroVAX II (under MicroVMS 4.6)	CAP Industry, Ltd. (Ref. pages 59 - 64)
80286 , Intel iAPX 80286 real mode (bare machine)	VAX-11/780 and VAX 11/785 (under VAX/VMS 4.5)	SofTech, Inc. (Ref. pages 126 - 147)
80286 , Intel iAPX 80286 protected mode (bare machine)	VAX-11/780 and VAX 11/785 (under VAX/VMS 4.5)	SofTech, Inc. (Ref. pages 126 - 147)
80286 , Intel iSBC 286/14 (bare machine)	IBM PC/AT (under PC/DOS 3.2)	Alsys (Ref. pages 36 - 49)
80286 , Intel iSBC 286/12 (bare machine) *Derived*	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.6 or MicroVMS 4.6)	DDC-I (Ref. pages 65 - 80)
80286 , Intel iSBC 286/12 Protected mode (bare machine) *Derived*	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.6 or MicroVMS 4.6)	DDC-I (Ref. pages 65 - 80)
80286 , Titan SECS 286/20 (bare machine)	DEC MicroVAX II (under MicroVMS 4.4)	DDC-I (Ref. pages 65 - 80)
80386 , Intel 80386 on an Intel 386-100 board (bare machine)	VAX 8530 (under VMS, version 4.6)	TeleSoft/Intel Corp./TeleLOGIC (Ref. page 223)
80386 , Intel iAPX 80386 compatibility mode (bare machine)	VAX-11/780 and VAX 11/785 (under VAX/VMS 4.5)	SofTech, Inc. (Ref. pages 126 - 147)

Guideline to Select, Configure, and Use an Ada Runtime Environment

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
80386, Intel iSBC 386/21 (bare machine)	DEC MicroVAX II (under MicroVMS 4.4)	DDC-I (Ref. pages 65 - 80)
80386, Intel iSBC 386/21 (bare machine) *Derived*	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.6 or MicroVMS 4.6)	DDC-I (Ref. pages 65 - 80)
80386, Intel iSBC 386/21 Protected mode (bare machine) *Derived*	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.6 or MicroVMS 4.6)	DDC-I (Ref. pages 65 - 80)
80386, Intel iSBC 386/20P using file-server support from the Host (bare machine)	MicroVAX II (under MicroVMS, Version 4.4)	Verdix Corp. (Ref. pages 202 - 206)
80386, Intel iSBC 386/20P using file-server support from the Host (bare machine)	MicroVAX II (under MicroVMS, Version 4.7)	Verdix Corp. (Ref. pages 202 - 206)
80386, Intel iSBC 386/20 (bare machine)	Intel system 320 (under UNIX system version release 3.0)	Verdix Corp. (Ref. page 227)
80386, Intel iSBC 386/20P using file-server support from the Host (bare machine) * Derived *	VAX 8800, 87000 8650, 8600, 8500, 8300, 8200 VAX 11/785, 782, 780, 750, 730, & MicroVAX II (under VMS 4.4)	Verdix Corp. (Ref. pages 202 - 206)
80386, Intel iSBC 386/20P using file-server support from the Host (bare machine)	Sequent Symmetry S-27(under DYNIX, release 3.0)	Verdix Corp. (Ref. page 227)
80386, Force CPU-386 VMEbus (bare machine)	DEC MicroVAX II (under MicroVMS 4.4)	DDC-I (Ref. pages 65 - 80)

Guideline to Select, Configure, and Use an Ada Runtime Environment

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
680x0, MC680x0 (bare machine)	IBM PC/AT, Compaq 386, SUN-3, HP-300, VAX/VMS	Alsys (Ref. pages 50 - 58)
68000, MC68000/10 implemented on the MVME 117-3FP board (bare machine) * Derived *	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.5 or MicroVMS 4.5)	Systems Designers Software, Inc. (Ref. page 222)
68000, MC68000/10 implemented on the MVME 117-3FP board (bare machine) * Derived *	DEC VAX-11/7xx, VAX 8xxx, VAX Station (under VMS 4.6) and MicroVAX Series (under MicroVMS 4.5)	Systems Designers Software, Inc. (Ref. pages 222)
68000, MC68000 implemented on a Motorola MVME 101 board (bare machine)	Sun Microsystems Sun-3/280 Workstation (under Sun UNIX version 4.2, release 3.2)	TeleSoft, Inc. (Ref. pages 224 - 226)
68000, MC68000 implemented on a Motorola MVME 101 board (bare machine) *Derived*	Sun Microsystems Sun-3 Workstations, Models: 260, 180, 160, 150, 140, 110, 75, 60, 50 and 52 (with soft- ware floating point); SOME and 52 + 152A (with MC68881 FPC) (under Sun UNIX version 4.2, Releases 3.2 & 3.4)	TeleSoft, Inc. (Ref. pages 224 - 226)
68000, MC68000 implemented on a Motorola MVME 101 board (bare machine)	MicroVAX II (under VMS, version 4.6)	TeleSoft, Inc. (Ref. pages 224 - 226)
68000, MC68000 implemented on a Motorola MVME 101 board (bare machine) * Derived*	DEC VAX family (MicroVAX, VAX station, VAX server, VAX 8xxx, & VAX-11 models) (under VMS 4.5 and 4.6)	TeleSoft, Inc. (Ref. pages 224 - 226)

Guideline to Select, Configure, and Use an Ada Runtime Environment

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
68010, MC68010, implemented on the MVME 117-3FP board (bare machine)	DEC VAX 8600 (under VMS 4.5)	Systems Designers Software, Inc. (Ref. pages 150 - 153)
68010, MC68010 implemented on a Motorola MVME 117-4 board (bare machine)	Sun Microsystems Sun-3/280 Workstation (under Sun UNIX version 4.2, release 3.2)	TeleSoft, Inc. (Ref. pages 224 - 226)
68010, MC68010 implemented on a Motorola MVME 117-4 board (bare machine) *Derived*	Sun Microsystems Sun-3 Workstations, Models: 260, 180, 160, 150, 140, 110, 75, 60, 50 and 52 (with soft- ware floating point); SOME and 52 + 152A (with MC68881 FPC) (under Sun UNIX version 4.2, Releases 3.2 & 3.4)	TeleSoft, Inc. (Ref. pages 224 - 226)
68010, MC68010 implemented on a Motorola MVME 117-4 board (bare machine)	MicroVAX II (under VMS, version 4.6)	TeleSoft, Inc. (Ref. pages 225 - 226)
68010, MC68010 implemented on a Motorola MVME 117-4 board (bare machine) * Derived*	DEC VAX family (MicroVAX, VAX station, VAX server, VAX 8xxx, & VAX-11 models) (under VMS 4.5 and 4.6)	TeleSoft, Inc. (Ref. pages 225 - 226)
68010, MC68010 implemented on a Motorola MVME 133A-20 board with a MC6881 floating point coprocessor (bare machine) *derived*	Sun Microsystems Sun-3 Workstations, Models: 260, 180, 160, 150, 140, 110, 75, 60, 50 and 52 (with soft- ware floating point); SOME and 52 + 152A (with MC68881 FPC) (under Sun UNIX version 4.2, Releases 3.2 & 3.4)	TeleSoft, Inc. (Ref. pages 225 - 226)

Guideline to Select, Configure, and Use an Ada Runtime Environment

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
68020, MC68020 Motorola MVME 133 board (bare machine)	Micro VAX II (under MicroVMS v.4.5)	AITech Software Engineering Ltd. (Ref. pages 28 - 35)
68020, MC68020 (bare machine)	Rational 1000	Rational (Ref. pages 110 - 125)
68020, MC68020, implemented on the MVME 133 board with a MC68881 floating point coprocessor (bare machine)	DEC VAX 8600 (under VMS 4.5)	Systems Designers Software, Inc. (Ref. pages 154 - 161)
68020, MC68020, implemented on the MVME 133 board with a MC68881 floating point coprocessor (bare machine)	DEC VAX-11/7xx VAX 8xxx, VAX station, and Micro VAX series (under VAX/VMS 4.5 or MicroVMS 4.5)	Systems Designers Software, Inc. (Ref. pages 154 - 161)
68020, MC68020, implemented on the MVME 133 board with a MC68881 floating point coprocessor (bare machine)	DEC VAX-11/7xx VAX 8xxx, VAX station (under VMS 4.6), MicoVAX series (under MicroVMS 4.5)	Systems Designers Software, Inc. (Ref. pages 154 - 161)
68020, MC68020 implemented on a Motorola MVME 133A-20 board with a MC68881 floating point coprocessor	DEC VAX family (MicroVAX VAX station VAX server. VAX 8xxx models) (under VMS 4.5 and 4.6)	TeleSoft, Inc. (Ref. pages 181 - 188)
68020, MC68020 implemented on a Motorola MVME 133A-20 board with a MC68881 floating point coprocessor	MicroVAX II (under VMS 4.6)	TeleSoft, Inc. (Ref. pages 181 - 188)

Guideline to Select, Configure, and Use an Ada Runtime Environment

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
68020, MC68020 implemented on a Motorola MVME 133-A-20 board with a MC68881 floating point coprocessor (bare machine)	Sun Microsystems Sun-3/280 Workstation (under Sun UNIX version 4.2, release 3.2)	TeleSoft, Inc. (Ref. pages 225 - 226)
68020, Microbar GBC68020 (bare machine) using file-server support from the Host	Sun Microsystems Sun-3/160 (under Sun UNIX 4.2, Release 3.2)	Verdix Corp. (Ref. pages 207 - 218)
68020, Microbar GPC-68020 (bare machine)	MicroVAX II (under UNIX 4.2 BSD)	Verdix Corp. (Ref. pages 207 - 218)
68020, Microbar GPC-68020 (bare machine)	MicroVAX II (under MicroVMS 4.4)	Verdix Corp. (Ref. pages 207 - 218)
32032, National DB32000 (NS32032) (bare machine) using file-server support from the Host	MicroVAX II (under MicroVMS, Version 4.4)	Verdix Corp. (Ref. page 228)
32032, National DB32000 (NS32032) (bare machine) using file-server support from the Host * Derived *	VAX 8800, 87000 8650, 8600, 8500, 8300, 8200 VAX 11/785, 782, 780, 750, 730, & MicroVAX II (under VMS 4.4)	Verdix Corp. (Ref. page 228)
32032, National DB32000 (NS32032) (bare machine) using file-server support from the Host	SYS32/20 (under Opus5 (UNIX SYS V), release 2.0)	Verdix Corp. (Ref. page 228)
MicroVAX II (under VAXELN Toolkit, Version 3.0 in Combination with VAXELN Ada, Version 1.2)	VAX 8800 (under VAX/VMS, Version 4.7)	Digital Equipment Corp. (Ref. pages 81 - 82)

Guideline to Select, Configure, and Use an Ada Runtime Environment

TARGET PROCESSOR	HOST PROCESSOR	COMPILER VENDOR
Any of the following configurations: MicroVAX I & II; rtVAX 1000; KA620 (rtVAX 1000 processor board); MicroVAX 3500 & 3600; VAX-11/730 & 750 and VAX 8500, 8530, 8550, 8700, & 8800 (under VAXELN Toolkit, version 3.0 in combination with VAXELN Ada version 1.2) * Derived *	All members of the VAX family: MicroVAX I, VAXstation I, MicroVAX II, VAXstation II, VAXstation 2000 (under MicroVMS, version 4.7); MicroVAX 3500 & 3600; VAXserver 3500, 3600, & 3602; and VAXstation 3200, 3500 (under VAX/VMS version 4.7A); VAX-11/730, 750, 780, 782, 785, VAX 8200, 8250, 8300, 8350, 8530, 8550, 8600, 8650, 8700, and 8800 (under VAX/VMS, version 4.7)	Digital Equipment Corp. (Ref. pages 81 - 82)
CAPS/AAMP (bare machine)	VAX-11/8650 (under VMS, Version 4.5)	Rockwell Int'l. (Ref. page 221)
CAPS/AAMP (bare machine)	DEC VAX 8650 (under VMS, Version 4.7)	Rockwell Int'l. (Ref. page 221)
Gould PowerNode Model 6080 (or SelConnection) (bare machine)	Gould PowerNode Model 9080 (under UTX/32 Version 2.0)	Gould, Inc. (Ref. pages 83 - 84)

Guideline to Select, Configure, and Use an Ada Runtime Environment

Table II. Bare Machine Targeted Compilers

The following table is a list of bare machine compilation systems listed in alphabetical order by vendor. The compiler vendor, host processor, target processor (grouped by machine family), and compiler version are presented in a banner heading each new host/target combination page. Following each banner is the following information:

1.) Degree of Configurability - The survey found in Appendix A was used to obtain this information. The sources of information were: the compiler vendors, the appropriate compiler documentation, and users. Item VI. under this section describes the source(s) of information for the information reported. Some vendors provided a technical summary and this was included as appropriate.

2.) PIWG Benchmarks - These benchmarks were included to provide some feeling for the efficiency of the implementation. Each compiler vendor had their own subset of the PIWG benchmarks which they supplied as input. A description of the tests can be found in section 4.1. Processor speed and wait-state of the memory is provided to properly compare the results. Users are encouraged to contact the PIWG directly for the benchmark results of new compiler releases.

3.) Histogram - This is a graphical display of the runtime component sizes, and most of the runtime sizing information was supplied by the vendors. Since this information did not conform to a standard format the data represented on the graphs is displayed in the same format as it was received it from the vendor. To change this information to conform to a standard format (see survey in Appendix A) might have resulted in misrepresentation of the compiler/vendor.

IT SHOULD BE NOTED that it is NOT advisable to compare the different compilers with just the aid of the graphs themselves. For an accurate overview of each compiler, ALL documentation provided for a particular compiler must be taken in account.

The graphs DO NOT show inter-dependencies of each component with respect to the other components. The sizes shown represent each individual component alone and do not take in account the fact that in order to use one component (component-A), two other components (components-B & C) might also have to be loaded to make component-A functional. This obviously adds to the size needed to use the component.

The graphs display maximums (and minimums when supplied) and does not express granularity of the components. The actual sizes may depend upon the features of the application code. For example, component-A's maximum size may be 10,000 bytes, but depending upon the language construct used, only part of the component may be loaded.

The graphics package that was used to draw the graphs scaled the information to fit within a fixed size window. Therefore, depending on the amount of information to be displayed within this fixed size window, some graphs appear smaller than others. This has nothing to do with the vendors or their products. It's simply a function of the graphics package used.

4.) Package SYSTEM and Package STANDARD specification, when provided the vendor.

Guideline to Select, Configure, and Use an Ada Runtime Environment

5.) Response to Critical Questions. As the surveys (see Appendix A) were returned, they were fine-tuned. A later survey (V2.0, also found in Appendix A), asks the vendors/users to respond to additional questions that are important to know if one is developing real-time software. These were limited to 10 questions because a limit had to be placed on the length of the survey to realistically expect people to answer it. The responses received are included.

Note that while most compiler vendors were willing to provide at least some of the requested data, not all responded to the survey. When this was the case, it was so stated.

Finally, a vendor's validation certificate for a particular compilation system remains in effect for one year. The period of performance for this contract spanned nine months and during that time some compiler versions became obsolete. The information presented is as up to date and accurate as possible for the compiler version presented. Because the implementations are changing so rapidly, it is suggested that a potential customer contact the vendor for newer release information. However, this report provides a substantial amount of basic information that probably would not change drastically from release to release.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
AITech Software Engineering Ltd. Compiler version: AI-ADA/020 V2.1	VAX/VMS	MC68020, Motorola (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Any part of a library unit being required loads the entire unit.

II. Customization of the Runtime:

- By linker switches
- By modifying the source to the entire runtime (after purchasing it)

III. Documentation provided to help user configure runtime:

- This information was not supplied by the vendor.

IV. Services to customize the runtime:

- Provided by AITech
- Cost: Engineering services are provided, subject to negotiation between the customer and AITech.

V. Cost of runtime source code:

- The price of the runtime source code is subject to negotiation.

VI. Source of Information: Vendor Input.

Guidelines to Select, Configure and Use an Ada Runtime Environment

AITech Systems Ltd. PIWG results for MVME133 board, 68020 + 68881. Clock : 20MHz, one wait state, and cache enabled.

PIWG Test Name	Description	Micro - seconds
C000001	Task creation/terminate, task type declared in package.	546.9
C000002	Task creation/terminate, task type declared in procedure.	550.8
C000003	Task creation/terminate, task type declared in block.	554.7
T000001	Minimum rendezvous, entry call and return.	183.6
T000002	Task entry call and return (one task, one entry).	177.8
T000003	Task entry call and return (two tasks, one entry each).	201.2
T000004	Task entry call and return (one task, two entries).	193.4
T000005	Active entry and return (ten tasks, one entry each).	187.5
T000006	Task entry call and return (one task, ten entries).	226.6
T000007	Minimum rendezvous, entry call and return.	139.6
T000008	Parameter pass from producer task through buffer task to	464.9

Runtime System Overhead Measurements

Measurement	Description	Micro - seconds
Interrupt Response Time	from interrupt signal to first instruction in the rendezvous body	81
Interrupt Response for Null Rendezvous	from interrupt signal until a user task (the interrupted one or another) is resumed	67
Rendezvous Initiation Overhead	from arrival of second partner to first instruction in the rendezvous body	65
Rendezvous Termination Overhead	from end of rendezvous body until the user task is resumed	77
Clock Interrupt Overhead	time spent handling one clock one clock interrupt	3.8
Context Switch	from last instruction in one user task to "first instruction in another user task (measured on the statement: delay 0.0;)"	44

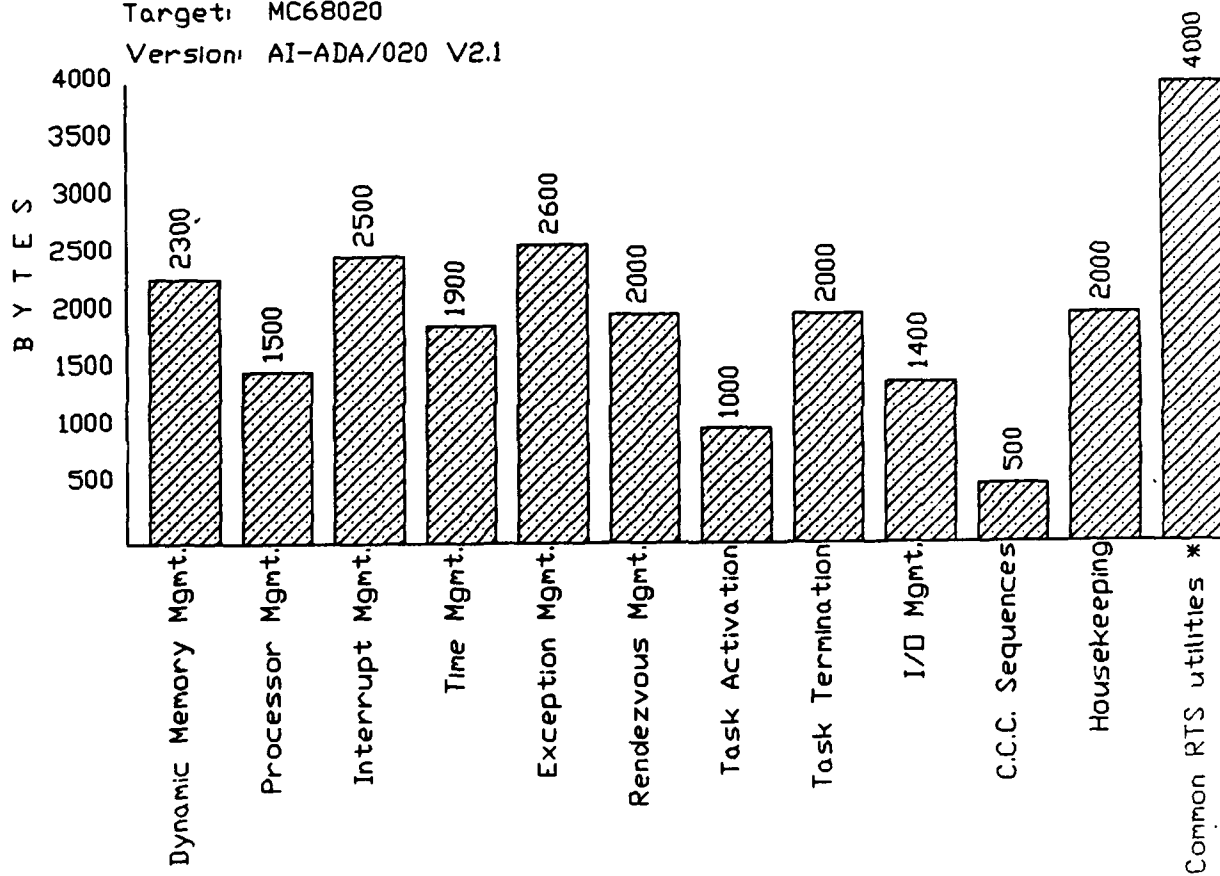
Guidelines to Select, Configure and Use an Ada Runtime Environment

Aitech Software Engineering Ltd.

Host: VAX / VMS

Target: MC68020

Version: AI-ADA/020 V2.1



- Sum of All Components = 23,700 bytes

* Component supplied by vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for `delay` statements?

A1: Currently 10 microseconds, but it can be configured to suit the application.

Q2: How long, and for what reasons are interrupts disabled?

A2: Maximum of 100 Microseconds when the RTS is handling some global data structures, and during context switches.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: The following are rendezvous mechanisms that are highly optimized:

- 1.) Efficient handling of the select statement.
- 2.) Special handling of synchronization rendezvous.
- 3.) Very low context switch overhead.

See PIWG results.

Q4: What are the restrictions for representation clauses?

A4: Representation clauses will be supported in the next version.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Priority driven preemptive scheduling with optional time slicing among tasks of equal priority.

Q6: What are the restrictions on `pragma INLINE`?

A6: Version 2.1 does not support `pragma INLINE`.

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: No.

Q9: What object types are supported by `pragma SHARED`?

A9: Only scalar objects (integers, real numbers, etc.).

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

- Maximum number of tasks (No limit)
- Task time slice default
- Timer resolution
- Exception trace
- Default stack sizes
- Fast interrupt entry
- Default task priority
- Terminal I/O

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the MC68020

package SYSTEM is

type ADDRESS is private;

subtype PRIORITY is INTEGER range 0 .. 23;

-- Priority 0 is reserved for the Null Task

-- Priority 24 is reserved for System Tasks

-- Priorities 25 .. 31 are for interrupts

type NAME is (M68020, M68000);

SYSTEM_NAME : constant NAME := M68020;

STORAGE_UNIT : constant := 16;

MEMORY_SIZE : constant := 2048 * 1024;

MIN_INT : constant := -2_147_483_647 - 1;

MAX_INT : constant := 2_147_483_647;

MAX_DIGITS : constant := 18;

MAX_MANTISSA : constant := 31;

FINE_DELTA : constant := 2#1.0#E-31;

TICK : constant := 0.000_001;

private

type ADDRESS is new LONG_INTEGER;

end SYSTEM;

Guidelines to Select, Configure and Use an Ada Runtime Environment

Description of Package STANDARD for AITech MC68020 Bare Machine Target

Integer Types

Three predefined integer types are implemented, SHORT_INTEGER, INTEGER, and LONG_INTEGER.

They have the following attributes:

```
SHORT_INTEGER'FIRST = -128
SHORT_INTEGER'LAST  = 127
SHORT_INTEGER'SIZE   = 8

INTEGER'FIRST        = -32_768
INTEGER'LAST         = 32_767
INTEGER'SIZE         = 16

LONG_INTEGER'FIRST   = -2_147_483_648
LONG_INTEGER'LAST    = 2_147_483_647
LONG_INTEGER'SIZE    = 32
```

Floating Point Types

Three predefined floating point types are implemented, SHORT_FLOAT, FLOAT, and LONG_FLOAT. They have the following attributes:

```
SHORT_FLOAT'DIGITS      = 6
SHORT_FLOAT'EPSILON      =
SHORT_FLOAT'FIRST       = -16#0.FFFF_FF#E32
SHORT_FLOAT'LARGE       =
SHORT_FLOAT'LAST        = 16#0.FFFF_FF#E32
SHORT_FLOAT'MACHINE_EMAX = 127
SHORT_FLOAT'MACHINE_EMIN = -126
SHORT_FLOAT'MACHINE_MANTISSA = 23
SHORT_FLOAT'MACHINE_OVERFLOWS = TRUE
SHORT_FLOAT'MACHINE_RADIX = 2
SHORT_FLOAT'MACHINE_ROUNDS = TRUE
SHORT_FLOAT'MANTISSA     =
SHORT_FLOAT'SAFE_EMAX    = 125
SHORT_FLOAT'SAFE_LARGE   = 42535275582707704281251401981719740416.0
SHORT_FLOAT'SAFE_SMALL   = 0.1175494350822287507968736537222245677E-37
SHORT_FLOAT'SIZE         = 32
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for AITech MC68020 Bare Machine Target (Continued)

```

FLOAT'DIGITS           = 15
FLOAT'EPSILON          =
FLOAT'FIRST            = -16#0.FFFF_FFFF_FFFF_F8#E256
FLOAT'LARGE            =
FLOAT'LAST             = 16#FFFF_FFFF_FFFF_F8#E256
FLOAT'MACHINE_EMAX     = 1023
FLOAT'MACHINE_EMIN     = -1022
FLOAT'MACHINE_MANTISSA = 52
FLOAT'MACHINE_OVERFLOW = TRUE
FLOAT'MACHINE_RADIX    = 2
FLOAT'MACHINE_ROUNDS   = TRUE
FLOAT'MANTISSA         =
FLOAT'SAFE_EMAX        = 1021
FLOAT'SAFE_LARGE       = 224711641857789388674147672112637508883611472.0E263
FLOAT'SAFE_SMALL       = 0.2225073858507201383090232717332404064219216E-307
FLOAT'SIZE             = 64

LONG_FLOAT'DIGITS      = 18
LONG_FLOAT'EPSILON     =
LONG_FLOAT'FIRST       = -16158503035655503648605529934797844443001542.0E573
LONG_FLOAT'LARGE       =
LONG_FLOAT'LAST        = 16158503035655503648605529934797844443001542.0E573
LONG_FLOAT'MACHINE_EMAX = 16383
LONG_FLOAT'MACHINE_EMIN = -16382
LONG_FLOAT'MACHINE_MANTISSA = 63
LONG_FLOAT'MACHINE_OVERFLOW = TRUE
LONG_FLOAT'MACHINE_RADIX = 2
LONG_FLOAT'MACHINE_ROUNDS = TRUE
LONG_FLOAT'MANTISSA     =
LONG_FLOAT'SAFE_EMAX    = 2047
LONG_FLOAT'SAFE_LARGE   = 1615850303565550364334980470618644983134.0E573
LONG_FLOAT'SAFE_SMALL   = 0.3094346047382578275480183369971197853892E-616
LONG_FLOAT'SIZE         = 80
```

Fixed Point Types

Three kinds of anonymous predefined fixed point types are implemented, named `SHORT_FIXED`, `FIXED`, and `LONG_FIXED`. Note that these names are not defined in package `STANDARD`, but only used here for reference.

8 bits are used for the representation of `SHORT_FIXED` types, 16 bits are used for the representation of `FIXED` types, and 32 bits are used for the representation of `LONG_FIXED` types.

For each of `SHORT_FIXED`, `FIXED` and `LONG_FIXED` there exists a virtual predefined type for each possible value of `SMALL`. The possible values of `SMALL` are the powers of two that are representable by a `LONG_FLOAT` value

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for AITech MC68020 Bare Machine Target (Continued)

The lower and upper bounds of these types are:

```
lower bound of SHORT_FIXED types = -128 * SMALL
upper bound of SHORT_FIXED types = 127 * SMALL
lower bound of FIXED types       = -32_768 * SMALL
upper bound of FIXED types       = 32_767 * SMALL
lower bound of LONG_FIXED types  = -2_147_483_648 * SMALL
upper bound of LONG_FIXED types  = 2_147_483_647 * SMALL
```

A user defined fixed point type is represented as that predefined SHORT_FIXED, FIXED, or LONG_FIXED type which has the largest value of SMALL not greater than the user-specified DELTA, and which has the smallest range that includes the user-specified range.

Any fixed point type T has the following attributes:

```
T'MACHINE_OVERFLOWS = TRUE
T'MACHINE_ROUNDS    = TRUE
```

The Type DURATION

The predefined fixed point type DURATION has the following attributes:

```
DURATION'AFT          = 5
DURATION'DELTA        = DURATION'SMALL
DURATION'FIRST        = -131_072.00000
DURATION'FORE         = 7
DURATION'LARGE        = 1.31071999938965E05
DURATION'LAST         = 131_071.00000
DURATION'MANTISSA     = 31
DURATION'SAFE_LARGE   = 1.31071999938965E05
DURATION'SAFE_SMALL   = DURATION'SMALL
DURATION'SIZE         = 32
DURATION'SMALL        = 6.10351562500000E-05 = 2#1.0#E-14
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Alsys Compiler version 3.21	IBM PC/AT (under PC/DOS 3.2)	80286, Intel iSBC 286/14 (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Subprograms are loaded if used. Package data is always loaded if the package is in a context clause.

II. Customization of the Runtime:

- By the use of compiler switches.
- By linker switches.
- Modifying/Replacing the source to selective runtime routines provided with the purchase of the compiler (i.e. Device Drivers).
- By modifying the source to the entire runtime (after purchasing it).

The target runtime system for the cross-compiler consists of the following sections:

Ada Runtime Executive - performs various high-level services (e.g., exception and interrupt handling). For typical operating system functions, the executive invokes the Bare-Machine Kernel. The Ada Runtime Executive is provided by Alsys.

Bare-Machine Kernel - invokes the Hardware Interface to perform any operations specific to the particular hardware in the system. The Bare-Machine Kernel is provided by Alsys.

Hardware Interface - is the interface to the specific hardware in the system. It is unique to each specific system and must therefore be supplied by the user. The user must provide configuration routines, configuration parameters and configuration tables. These are written in assembly language. They are linked into the runtime system.

Hardware Setup - contains the following routines and configuration parameters for performing hardware housekeeping operations of the system.

Configuration Routines:

An initialization routine which is used to initialize the peripheral chips on the board and install any specialized interrupt handlers.

A routine to return the processor to real mode.

Guidelines to Select, Configure and Use an Ada Runtime Environment

A routine to perform any necessary cleanup of the hardware or operating environment and return control to the operating system.

A routine to return in the DX:AX register pair the 32 bit address of the end of the memory area reserved for the heap.

Configuration Parameters:

The keyboard (or serial input channel) interrupt number.

The timer interrupt number (level).

The frequency at which the timer interrupts occur.

XON/XOFF protocol specification.

Input and Output Routines:

A routine to handle keyboard interrupts. It collects the incoming character and clears the interrupt from the interrupting device.

A routine to write a character to the console output device.

A routine to handle timer interrupts. It updates the real-time clock and clears the interrupt from the interrupting device.

Device Drivers - perform the various input/output functions. They are unique to each specific system and must therefore be supplied by the user. The user must provide configuration routines, configuration parameters and configuration tables. These are written in assembly language. They are linked into the runtime system.

Configuration Tables:

A table which contains the list of names corresponding to the devices in the system.

A table which contains a list of 16-bit values, one for each device in the above table. Each value specifies whether the corresponding device is the console input, console output, some other device, or a file.

A table containing the names of initialization routines, one for each device.

A table containing the names of necessary cleanup routines (such as flushing buffers), one for each device.

A table containing the names of routines which allow the user to maintain a file position so that direct I/O could be performed on the device, one for each device.

Guidelines to Select, Configure and Use an Ada Runtime Environment

A table containing the names of routines which allow a sequence of characters to be sent to the device, one for each device.

Configuration Routines:

A procedure must be specified for each routine named in the above tables.

Useful External Routines - These routines are provided for use in the Hardware Interface and Device Specifications.

A routine to put a character into the input buffer so that it can be fetched by the Ada program.

A routine to update the real-time clock.

A routine to get a character from the console-input device.

A routine to output a character to the console-output device.

A routine which will cause the program to be restarted (start of the bare-machine kernel).

Tools:

The Intel 8086 toolset is used, namely: ASM86, LINK86, LOC86, LIB86.

Transfer Tools - A tool which allows cross loading.

80x86AdaProbe - cross debugging version of AdaProbe. It is an IBM PC AT-hosted program viewer/debugger that works with other components of the cross-compiler to debug code that executes on any Intel i80x86-family processor.

It supports all three execution modes provided by the cross-compiler:

- In cross mode the code executes on a remote target with no operating system. 80x86 AdaProbe runs on the host, communicating with the target over a serial link or equivalent.
- In simulated mode the code executes on the host machine but without making any calls to the operating system.
- In native mode the code runs on the host taking full advantage of DOS and BIOS.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Technical Summary for Alsys Cross Compilation Systems for Intel 80x86 (V. 3.21)

Alsys provides a document titled "Technical Summary for Alsys Cross Compilation Systems for Intel 80x86 (version 3.21)", dated July 11, 1988. [16] Its purpose is to aid prospective purchasers of Alsys cross compilation systems for Intel 80x86 microprocessors in their evaluations of the technical characteristics of the products. It addresses the more common concerns of real-time and embedded applications developers when selecting an Ada compilation system. Following is an index of the information contained in it.

- A. Components & Applicability of Product.**
 - 1. Compilation System Components.
 - 2. Documentation Set.
 - 3. Available Hosts.
 - 4. Supported Targets.
 - 5. Software and Hardware Requirements.
 - 6. Validation Status.
- B. Compilation System.**
 - 1. Capacities & Robustness.
 - 2. Speed/Throughput.
 - 3. Library Facilities.
 - 4. Code Quality.
 - 5. Error Messages.
 - 6. Ease of Use, Convenience, Flexibility.
 - 7. Implementation Dependent Features.
 - 8. Target Specific Features.
 - 9. Optimizations.
- C. Runtime System.**
 - 1. Capacities and Robustness.
 - 2. Performance/Resource Usage.
 - 3. Configurability Support.
 - 4. Tailorability.
 - 5. Extensions.
 - 6. Source Code Availability & Implementation Language.
 - 7. Certifiability.
 - 8. Features.
- D. Development Tools.**
 - 1. Debugger Support.
 - 2. Profiler.
 - 3. Support for Logic Analyzers and Emulators Independent of Debugger.
 - 4. Cross Referencer.
 - 5. Source Formatter.
 - 6. Language Sensitive Editor.
 - 7. Source Configuration Control.
 - 8. Communications Support.
 - 9. Test Case Generators.
 - 10. Support for Development of Independent Tools.
- E. Documentation.**

Guidelines to Select, Configure and Use an Ada Runtime Environment

1. Use of System.
2. Implementation Dependencies.
3. Configurability.
4. Installation Guide.
5. Project Development Guide.
6. Command Reference.
7. Index.
8. Applications Development Guide (tips, etc.).
9. Runtime System Guide.
10. Bug Lists.
11. Change Bars on Succeeding Revisions.

The following excerpts are from the section titled "Runtime System".

1. Capacities and robustness.

a. Max active tasks. Bounded by available memory. Task representation is approximately 160 bytes, plus 500 bytes stack overflow buffer, plus the designated task stack size. Programs have been executed with 500 simultaneously active tasks.

2. Performance/resource usage.

a. PIWGs and other benchmarks. Can be found following this section.

b. Size. The size of the runtime environment varies from approximately 14 K bytes to 28K bytes, depending on features used.

3. Configurability support. The compilation system offers comprehensive configurability support. Configuration customization of the runtime environment is effected via configuration files and user defined hook routines. The areas under user control include:

- Size and location of the Ada heap for dynamic memory management.
- Designation of default task stack size.
- User supplied routines to initialize and handle the timer, and designation of its period (the effective TICK of the application).
- Facilities for integrating I/O devices.

4. Tailorability. One aspect of tailorability, aside from the configuration facilities listed above, is that the system supports unused subprogram elimination for both user and RTE code.

5. Extensions. No runtime environment extensions are currently implemented.

6. Source code availability & implementation language. The compiler and a large part of the RTE are implemented in Ada. Some portions of the RTE are implemented in assembly language.

Source code for the RTE is available through a separate arrangement (AlsysARTE).

7. Certifiability. Alsys is willing to enter into special arrangements should a project require special certification of portions of the RTE.

Guidelines to Select, Configure and Use an Ada Runtime Environment

8. Features.

a. Interrupt support. Not supported in this release, but will be available beginning November 1988.

b. Scheduler Characteristics.

1) *Preemptive.* The scheduler is fully preemptive and interrupt driven. Scheduling actions are load-insensitive: the number of simultaneously active tasks does not affect the time to perform scheduling actions.

2) *Priority levels & treatment of undefined priority.* Priorities may be defined in the range 1 .. 16. An undefined priority is considered to be lower than any defined value.

3) *Consideration of hardware interrupt priority levels.* Interrupt entries may have up to 8 priority levels beyond type System.Priority.

4) *Time slicing.* Time slicing may be set via a binder command option. Granularity for specifying the quantum is 1 millisecond.

5) *Support for rate monotonic scheduling.* Rate monotonic scheduling is not implemented in the current version.

6) *Load sensitivity.* All scheduling and inter-task operations are load insensitive. They are not affected by the number of simultaneously active tasks.

7) *Deadlock or permanent blocking detection & support for actions on same.* When the RTE detects a permanent blocking situation, it terminates the application by transferring control to the user written termination hook.

c. Time support.

1) *Clock resolution.* Clock resolution is determined by the user definition of the basic real-time clock period.

2) *DURATION characteristics.*

DURATION'DELTA = .001 seconds

DURATION'SMALL = 2**-10 seconds

DURATION'FIRST = -2_097_152.0 seconds

DURATION'LAST = DURATION'LARGE = 2_097_151.999 seconds

3) *TICK SYSTEM.TICK* = 1/18 seconds, but is unused. The effective TICK is designated by the application builder in the configuration file.

4) *Clock call overhead.* Not currently determined.

5) *Typical time to reschedule when highest priority task times out.* On the order of 80 microseconds on an 8 MHz, 0 wait state 80286.

6) *Time operations overhead.* Not currently measured.

Guidelines to Select, Configure and Use an Ada Runtime Environment

7) Configurability. As described in the configurability section, the user designates the basic period of the real-time clock used to drive all time based operations. The user also provides timer initialization and interrupt handler routines, as well as initialization of date and time.

d. Dynamic memory management approach.

Several classes of objects are allocated on the heap. These include objects created by the execution of an Ada allocator, task stacks, arbitrarily large objects and compiler generated temporaries. Special representations within the heap are used when objects are 32 bits or smaller, and when dynamic objects have global scope.

When an access type is defined in a task or subprogram, all objects of the type are automatically deallocated when the scope defining the type is exited. This implementation has the same effect as explicitly applying pragma CONTROLLED to each access type in the application. Compiler generated temporaries are reclaimed as soon as they are no longer needed. A task's stack is reclaimed as soon as the task terminates. UNCHECKED_DEALLOCATION reclaims an access object immediately.

In order to provide better management of stacks and global data areas, which are currently limited to 64 K bytes each, binder options are provided to set threshold values for the maximum size objects to be allocated in each of these areas. If an object is larger than the pertinent threshold value, it is allocated on the heap instead.

e. Exception management approach.

The exception management implementation follows the philosophy described in the Ada Rationale document, which considers exceptions to be exceptional, and not a normal method for transferring flow of control. The language designers felt that there should be no overhead at subprogram linkage related to exception management. Alsys has followed this philosophy by using a table driven, interpretive approach to exception management, which does not penalize subprogram linkage sequences for the possibility of an exception being raised.

f. Support for multiprocessor configurations.

There is no explicit support for multiprocessor configurations in the current release, but it is possible to build systems with stand-alone Ada programs on each processor which communicate with each other.

g. Support for multiprogramming.

There is no explicit support for multiprogramming in the current release, but it should be possible for a user to build such a system, provided that careful attention is given to correct setup of interrupt vectors.

h. Rendezvous implementation.

The rendezvous implementation uses the "naive" approach to execution of accept bodies: on the stack of the callee, executed by the callee. This approach requires less overhead for nested rendezvous implementation than alternative approaches.

Guidelines to Select, Configure and Use an Ada Runtime Environment

The synchronization rendezvous case, where there is an empty statement list for the accept body, is optimized and involves no context switches.

The selective wait statement is made reasonably "fair" in selection of among multiple open entries by varying the starting point for processing of open alternatives for selection.

Rendezvous algorithms are load-insensitive: they do not depend on the number of active tasks.

III. Documentation provided to help user configure runtime:

- As part of a standard product: Cross Development Guide
- As RTE technology transfer (ALSYARTE), all design documentation, a week long course and consulting services.

IV. Services to customize the runtime:

- Not for particular applications, but Alsys occasionally does custom work for projects of sufficient scope.

V. Cost of runtime source code:

- Approximately \$250,000, but it is dependent upon the specific situation.

VI. Source of Information: Vendor input and relevant compiler documentation supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Alslys PIWG results for iSBC286. Clock : 8MHz, iSBC286/12, Multibus I, zero wait-states (tests were compiled with Checks off, Optimizations on). PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
A000093	Whetstone benchmarks*	160**
C000001	Task creation/terminate, task type declared in package.	1747.1
C000002	Task creation/terminate, task type declared in procedure.	1912.5
C000003	Task creation/terminate, task type declared in block.	1734.3
D000001	Dynamic array, use and deallocation.	116.4
D000002	Dynamic array elaboration and initialization.	7200.9
D000003	Dynamic record allocation and deallocation.	150.8
D000004	Dynamic record elaboration and initialization.	7598.9
E000001	Raise and handle an exception locally.	374.2
E000002	Raise and handle an exception in a package.	732.0
E000004	Raise and handle an exception nested 4 deep in procedures.	1289.2
F000001	Set a BOOLEAN flag using a logical equation.	3.6
F000002	Set a BOOLEAN flag using an "if" test.	4.1
G000005	TEXT_IO.Get an INTEGER from a local string.	540.8
G000006	TEXT_IO.Get a FLOAT from a local string.	1850.0
L000001	Simple "for" loop.	3.9
L000002	Simple "while" loop.	3.9
L000003	Simple "exit" loop.	3.9
P000001	Procedure call and return (inlineable), no parameters.	0.0
P000002	Procedure call and return (not inlineable), no parameters.	4.8
P000003	Procedure call and return (compiled separately).	6.0
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	6.6
P000006	Procedure call and return (one parameter, out INTEGER).	7.5
P000007	Procedure call and return (one parameter, in out INTEGER).	7.5
P000010	Procedure call and return (ten parameters, in INTEGER).	16.4
P000011	Procedure call and return (twenty parameters, in INTEGER).	26.0
P000012	Procedure call and return (ten parameters, in record_type).	29.1
P000013	Procedure call and return (twenty parameters, in record_type).	52.0
T000001	Minimum rendezvous, entry call and return.	416.4
T000002	Task entry call and return (one task, one entry).	411.7
T000003	Task entry call and return (two tasks, one entry each).	417.6

Guidelines to Select, Configure and Use an Ada Runtime Environment

Alsys PIWG results for iSBC286 (continued). Clock : 8MHz, iSBC286/12, Multibus I, zero wait-states (tests were compiled with Checks off, Optimizations on). PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
T000004	Task entry call and return (one task, two entries).	741.4
T000005	Active entry and return (ten tasks, one entry each).	411.9
T000006	Task entry call and return (one task, ten entries).	1905.3
T000007	Minimum rendezvous, entry call and return.	253.9

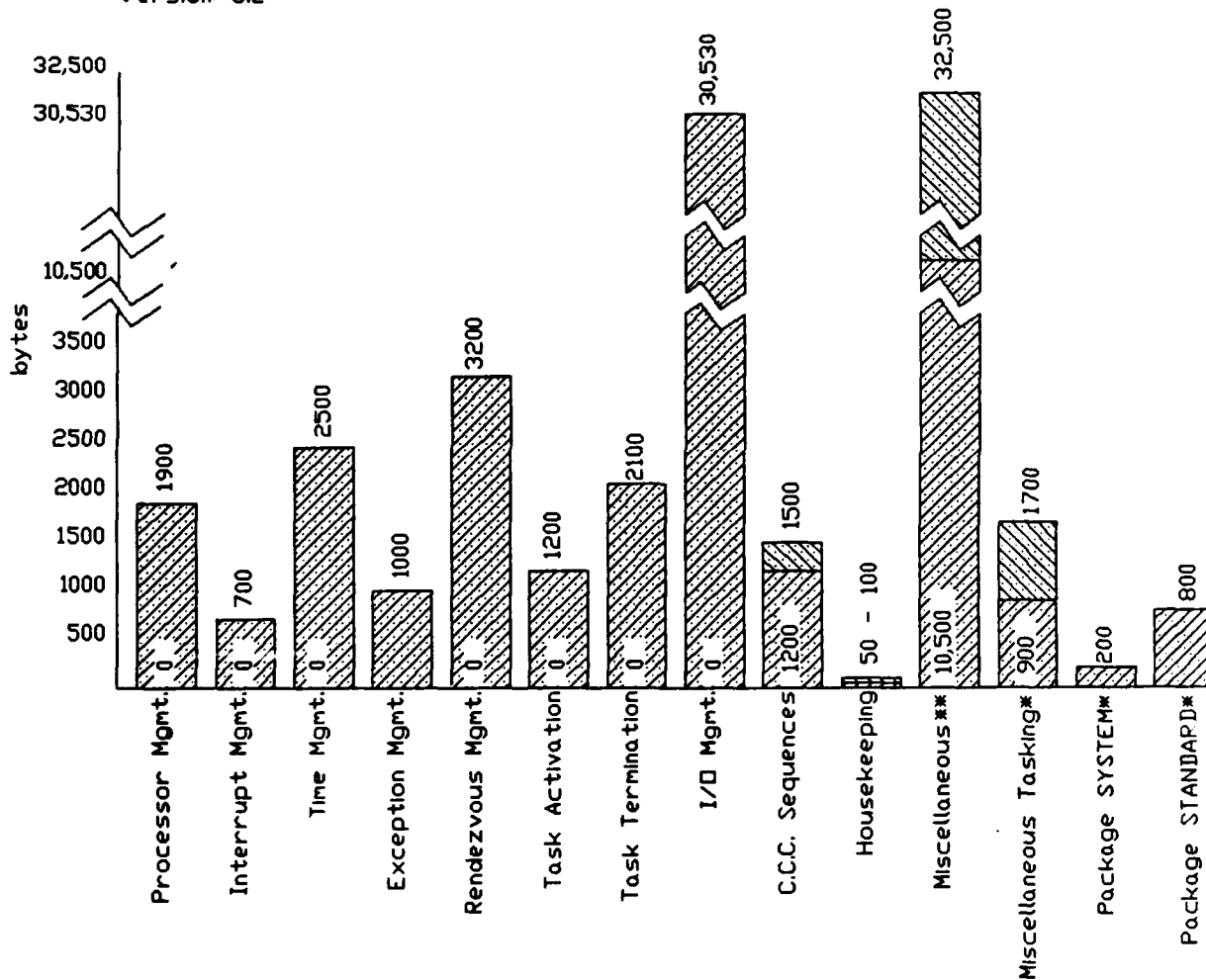
* Using standard internal math routines.

** WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Alsys, Inc.

Host: PC / AT
Target: Intel 80x86
Version: 3.2



Sum of ALL components = 14,650 - 79,930

* Component was supplied by vendor.

** Component supplied by vendor, see next page for details

Guidelines to Select, Configure and Use an Ada Runtime Environment

Vendor supplied component description

The minimum RTE including data is 14,650 bytes. The maximum RTE, including instantiations of all I/O packages and use of all possible RTE functions and I/O routines is 79,930 bytes.

A typical contribution of the RTE with usual I/O :

No tasking : approximately 22,000 - 24,000 bytes.

Tasking : approximately 25,000 - 30,000 bytes.

Processor Management

1. No tasking - 0 bytes;
2. Tasking - 1900 bytes.

Interrupt Management

1. No tasking - 0 bytes.
2. Tasking component not used - 600 bytes.
3. Tasking used (maximum) - 700 bytes.

Time Management

1. Calender not in context clause, no tasking - 0 bytes.
2. Calender in context but not called, no tasking - 200 bytes.
3. Calender in context clause (maximum) - 2,500 bytes.
4. Tasking timer component - 1,200 bytes.

Exception Management

Always present - 1000 bytes.

Rendezvous Management

1. No tasking - 0 bytes.
2. Tasking, rendezvous not used - 100 bytes.
3. Tasking, all types used (maximum) - 3,200 bytes.

Task Activation

1. No tasking - 0 bytes
2. At least one task defined - 1,200 bytes.

Task Termination

1. Abort not used - 0 bytes.
2. Abort used - 600 bytes.
3. Dependency maintenance - 2,000 - 2,100 bytes.

I/O Management:

Subcomponent	Not in context	In context, not used, instantiated	Maximum (all routines called)
Text_IO	0 bytes	2400 bytes	25,800 bytes
IO_Exceptions	0 bytes	30 bytes	30 bytes
Direct_IO	0 bytes	200 bytes	2,800 bytes
Sequential_IO	0 bytes	200 bytes	1,900 bytes

C.C.C. Sequences Arithmetic and block moves 1,200 - 1,500 bytes.

Miscellaneous category is additional to the above components, but implements some of the above functionality (including I/O, memory management, exception handling, tasking, etc.)

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

As surveys were returned, the original survey was fine-tuned based on comments received (see Survey, V. 2, in Appendix A). It was decided to comprise a list of ten important issues the user should obtain the answers to *before* selecting a compilation system for a particular application. Below are questions asked and answers received for this implementation.

Q1: What is the resolution of the clock used for **delay** statements?

A1: User configured.

Q2: How long, and for what reasons are interrupts disabled?

A2: Interrupts are only disabled in user written interrupt service routines.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: Based upon past implementations using Habermann-Nassi and Order of Arrival schemes for rendezvous execution, Alsys has abandoned these "optimizations" because they incur too much bookkeeping overhead. Accept bodies with empty statement lists are optimized to avoid context switching.

Q4: What are the restrictions for representation clauses?

A4: All representation clauses are currently supported to the byte level, except:

- bit level representation clauses (V4.1)
- pragma **PACK** (V4.1)
- change of representation for derived record types
- **T'SIZE** for types declared in a generic unit
- **T'SMALL** for fixed point types must be a power of 2, and the absolute value of the exponent must be less than 31
- enumeration clauses are not allowed if there is a range constraint on the parent subtype
- address representation clauses (the **ADDRESS** attribute is fully supported).
- the **STORAGE_SIZE** representation clause for reserving memory for task activation.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Preemptive, round-robin within priority level. Time slicing, under user control.

Q6: What are the restrictions on **pragma INLINE**?

A6: No direct or indirect recursion.

Q7: Is code "ROM"able?

A7: Yes, code, constants and initial values for global variables.

Q8: Are machine code inserts supported?

A8: No.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Q9: What object types are supported by pragma SHARED?

A9: Scalars.

Q10: What items are configurable for the runtime system?

A10: The items listed below are configurable for the runtime system.

- Max. No. of Tasks
- Task Time Slice Default
- Timer Resolution
- Exception Trace
- Default Stack Sizes
- Terminal I/O
- Optional Numeric Co-processor

Also see Technical Summary preceding this section.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Alsys Compiler version 3.5	Apollo, IBM PC/AT, Compaq 386, SUN-3, HP-300, VAX/VMS	MC680x0 (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

Any part of a library unit being required loads the entire unit. This changes with Version 4.2, October 1988.

II. Customization of the Runtime:

- By the use of compiler switches.
- By linker switches.
- Modifying/Replacing the source to selective runtime routines provided with the purchase of the compiler (i.e. Device Drivers).
- By modifying the source to the entire runtime (after purchasing it).

Technical Summary for Alsys Cross Compilation Systems for Motorola M680x0 (V. 3.5)

Alsys provides a document titled "Technical Summary for Alsys Cross Compilation Systems for Motorola M680x0 (version 3.5)", dated July 11, 1988. [17] Its purpose is to aid prospective purchasers of Alsys cross compilation systems for Motorola M680x0 microprocessors in their evaluations of the technical characteristics of the products. It addresses the more common concerns of real-time and embedded applications developers when selecting an Ada compilation system. Following is an index of the information contained in it.

- A. Components & Applicability of Product.
 - 1. Compilation System Components.
 - 2. Documentation Set.
 - 3. Available Hosts.
 - 4. Supported Targets.
 - 5. Software and Hardware Requirements.
 - 6. Validation Status.
- B. Compilation System.

Guidelines to Select, Configure and Use an Ada Runtime Environment

1. Capacities & Robustness.
 2. Speed/Throughput.
 3. Library Facilities.
 4. Code Quality.
 5. Error Messages.
 6. Ease of Use, Convenience, Flexibility.
 7. Implementation Dependent Features.
 8. Target Specific Features.
 9. Optimizations.
- C. Runtime System.
1. Capacities and Robustness.
 2. Performance/Resource Usage.
 3. Configurability Support.
 4. Tailorability.
 5. Extensions.
 6. Source Code Availability & Implementation Language.
 7. Certifiability.
 8. Features.
- D. Development Tools.
1. Debugger Support.
 2. Profiler.
 3. Support for Logic Analyzers and Emulators Independent of Debugger.
 4. Cross Referencer.
 5. Source Formatter.
 6. Language Sensitive Editor.
 7. Source Configuration Control.
 8. Communications Support.
 9. Test Case Generators.
 10. Support for Development of Independent Tools.
- E. Documentation.
1. Use of System.
 2. Implementation Dependencies.
 3. Configurability.
 4. Installation Guide.
 5. Project Development Guide.
 6. Command Reference.
 7. Index.
 8. Applications Development Guide (tips, etc.).
 9. Runtime System Guide.
 10. Bug Lists.
 11. Change Bars on Succeeding Revisions.

The following are excerpts from the section titled "Runtime System".

1. Capacities and robustness.

a. Max active tasks. For VRTX and VRTX32 (Ready Systems' real-time executives) it is 255. For ARTK (Alslys' real-time executive), as bounded by available memory.

2. Performance/resource usage.

Guidelines to Select, Configure and Use an Ada Runtime Environment

a. PIWGs and other benchmarks. See following pages.

b. Size. The runtime environment size is approximately 36 K Bytes in the current version. Further reductions will be implemented in V4.1.

3. Configurability support.

The compilation system offers comprehensive configurability support. Configuration customization of the runtime environment is effected via configuration files and user defined hook routines. The areas under user control include:

- Size and location of the Ada heap for dynamic memory management.
- Designation of default task stack size and interrupt stack size.
- Designation of maximum number of interrupts allowed to be simultaneously active or pending, and the maximum number of Ada interrupt entries defined over a program. These values are used to tailor internal data structures of the RTE.
- User supplied routines to initialize and handle the timer, and designation of its period (the effective TICK of the application).
- Facilities for integrating I/O devices.

4. Tailorability. Another aspect of tailorability, aside from the configuration facilities listed above, is that the Ada binder selects between a tasking and non-tasking runtime environment based upon the presence of tasking constructs in the application. V4.1 will support unused subprogram elimination for both user and RTE code.

5. Extensions. Package `USER_IO` is supported under ARTK. As an alternative to the address clause method of specifying interrupt entries, package `INTERRUPT_HANDLER` is provided. This package supports designation of the persistence of interrupts, and the use of a parameter.

6. Source code availability & implementation language. The compiler and a large part of the RTE are implemented in Ada. Some portions of the RTE are implemented in assembly language.

Source code for the RTE is available through a separate arrangement (AlsysARTE).

7. Certifiability. Alsys is willing to enter into special arrangements should a project require special certification of portions of the RTE.

8. Features.

a. Interrupt support.

1) Timing. Interrupt entries have roughly 150 microseconds overhead associated with them on a 20 MHz 68020.

2) Latency. Not yet measured for ARTK, but initial estimates of the maximum are in the range of 20 - 30 microseconds under certain infrequent conditions.

Ready Systems publish their interrupt latency figures for VRTX and VRTX32.

Guidelines to Select, Configure and Use an Ada Runtime Environment

3) *Fast Interrupts.* The system does not currently implement a special "fast interrupt" pragma, but note the time for interrupt entry rendezvous in 1) above.

4) *Types supported (e.g. persistence).* The system supports both persistent (unconditional) and nonpersistent (conditional) interrupts. Interrupt entries are executed as special software priority levels corresponding to the hardware priority level of the particular interrupt.

5) *Parameter support.* Parameters are not supported when using the address clause mechanism for interrupt entries. A single parameter of mode *in*, of discrete or access type, is supported when using the alternative package INTERRUPT_HANDLER mechanism.

b. Scheduler Characteristics.

1) *Preemptive.* The scheduler is fully preemptive and interrupt driven. Scheduling actions are load-insensitive under VRTX32 and ARTK: the number of simultaneously active tasks does not affect the time to perform scheduling actions.

2) *Priority levels & treatment of undefined priority..*

For ARTK, 24 user definable priority levels are available. The undefined priority value is considered to be less than any defined priority. Seven software priority levels are reserved for interrupt servicing. (See next section).

For VRTX and VTX32, users may define priorities in the range 1..248 for Ada tasks. Priorities 249 .. 255 are reserved for interrupt entries as described in the next section. Undefined priority is lower than any defined priority.

3) *Consideration of hardware interrupt priority levels.*

Under ARTK seven priority levels (25-31) are reserved for interrupt entry processing. Each level corresponds to a hardware interrupt level. Note that execution of accept bodies for interrupt entries does not take place at the hardware interrupt level, because interrupts are not disabled during execution of an accept body. A task executing an interrupt entry may be preempted by another task executing an interrupt entry for an interrupt at a higher hardware interrupt level.

An analogous scheme is used for VRTX and VRTX32, using priority values 249 -255.

4) *Time slicing.* Time slicing will be implemented for ARTK in V4.1

Time slicing is currently implemented for VRTX and VRTX32.

5) *Support for rate monotonic scheduling.* Rate monotonic scheduling is not implemented in the current version.

6) *Load sensitivity.* All scheduling and inter-task operations are load insensitive for VRTX32 and ARTK. They are not affected by the number of simultaneously active tasks.

Guidelines to Select, Configure and Use an Ada Runtime Environment

7) *Deadlock or permanent blocking detection & support for actions on same.* Not supported when interrupt entries are used in the application.

c. *Time support.*

1) *Clock resolution.* Clock resolution is determined by the user definition of the basic real-time clock period.

2) *DURATION characteristics.*

Type DURATION is delta 2.0**(-14) range -86_400.00 .. 86_400.0;

3) *TICK.* TICK has the value 1.0, but is unused. The effective TICK is designated by the application builder in the configuration file.

4) *Clock call overhead.* 68 microseconds, for a 12 MHz, VME130 with 4 wait-states.

5) *Typical time to reschedule when highest priority task times out.* On the order of 30-40 microseconds.

6) *Time operations overhead.* Not currently measured.

7) *Configurability.* As described in the configurability section, the user designates the basic period of the real-time clock used to drive all time based operations. The user also provides timer initialization and interrupt handler routines, as well as initialization of date and time.

d. *Dynamic memory management approach.*

Several classes of objects are allocated on the heap. These include objects created by the execution of an Ada allocator, task stacks, arbitrarily large objects and compiler generated temporaries. Special representations within the heap are used when objects are 32 bits or smaller, and when dynamic objects have global scope.

When an access type is defined in a task or subprogram, all objects of the type are automatically deallocated when the scope defining the type is exited. This implementation has the same effect as explicitly applying pragma CONTROLLED to each access type in the application. Compiler generated temporaries are reclaimed as soon as they are no longer needed. A task's stack is reclaimed as soon as the task terminates. UNCHECKED_DEALLOCATION reclaims an access object immediately.

e. *Exception management approach.*

The exception management implementation follows the philosophy described in the Ada Rationale document, which considers exceptions to be exceptional, and not a normal method for transferring flow of control. The language designers felt that there should be no overhead at subprogram linkage related to exception management. Alsys has followed this philosophy by using a table driven, interpretive approach to exception management, which does not penalize subprogram linkage sequences for the possibility of an exception being raised.

Guidelines to Select, Configure and Use an Ada Runtime Environment

f. Support for multiprocessor configurations.

There is no explicit support for multiprocessor configurations in the current release, but it is possible to build program per processor systems.

g. Support for multiprogramming.

There is no explicit support for multiprogramming in the current release, but it should be possible for a user to build such a system, provided that careful attention is given to correct setup of interrupt vectors.

h. Rendezvous implementation.

The rendezvous implementation uses the "naive" approach to execution of accept bodies: on the stack of the callee, executed by the callee. This approach requires less overhead for nested rendezvous implementation than alternative approaches. Parameters are always passed in a parameter area, and are never copied into the context of the accept body.

The synchronization rendezvous case, where there is an empty statement list for the accept body, is optimized and involves no context switches, unless the tasks are of unequal priority.

The selective wait statement is made reasonably "fair" in selection of among multiple open entries by varying the starting point for processing of open alternatives for selection.

Rendezvous algorithms are load-insensitive: they do not depend on the number of active tasks.

III. Documentation provided to help user configure runtime:

For off-the-shelf product, Cross Development Guide As technology transfer, full design documentation, 1 week training course and, consultant services.

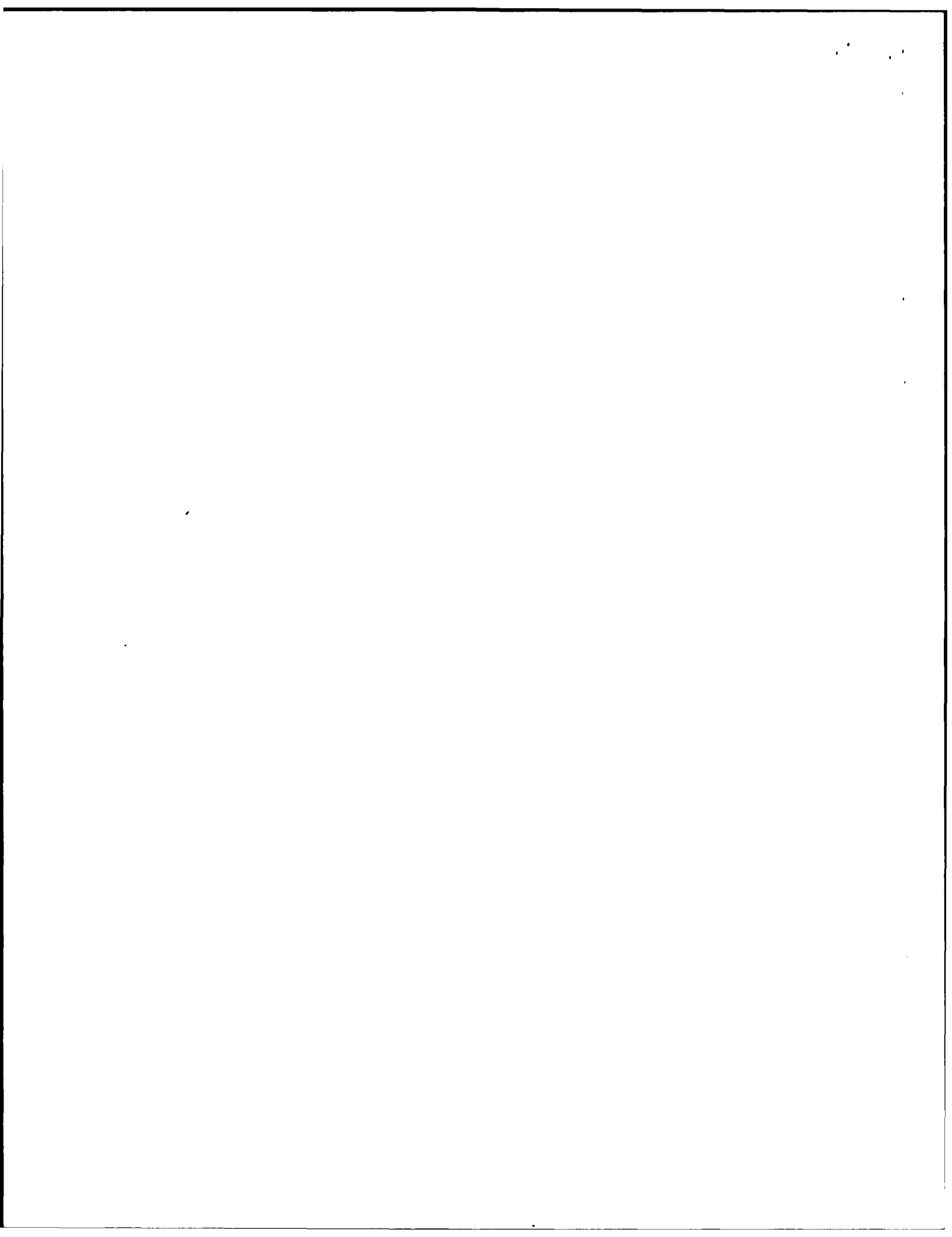
IV. Services to customize the runtime:

No.

V. Cost of runtime source code:

Approximately \$250,000.00, but it depends upon the specific situation.

VI. Source of Information: Vendor input and relevant compiler documentation supplied by the vendor.



Guidelines to Select, Configure and Use an Ada Runtime Environment

Alsys PIWG results for MVME 133A.

Clock: 20MHz, M68020, 1 Megabyte on board DRAM: 32 bit address and data access;
1 wait state. Checks ON, Optimizations ON, Timer configured to 1024 Hz TICK.
PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone	334
A000092	Whetstone benchmarks, using manufacturer's math routines	558*
A000093	Whetstone benchmarks, using standard math routines	310*
C000001	Task creation/terminate, task type declared in package.	1372.8
C000002	Task creation/terminate, task type declared in procedure.	1457.3
C000003	Task creation/terminate, task type declared in block.	1450.6
D000001	Dynamic array, use and deallocation.	8.1
D000002	Dynamic array elaboration and initialization.	6034.0
D000003	Dynamic record allocation and deallocation.	38.5
D000004	Dynamic record elaboration and initialization.	9456.1
E000001	Raise and handle an exception locally.	1044.4
E000002	Raise and handle an exception in a package.	3188.7
E000003	Raise and handle an exception nested 3 deep in procedures.	5554.6
E000004	Raise and handle an exception nested 4 deep in procedures.	6242.8
E000005	Raise and handle an exception in a rendezvous.	10098.5
F000001	Set a BOOLEAN flag using a logical equation.	4.8
F000002	Set a BOOLEAN flag using an "if" test.	5.2
G000005	TEXT_IO.Get an INTEGER from a local string.	334.2
G000006	TEXT_IO.Get a FLOAT from a local string.	1822.6
H000001	BOOLEAN operations on entire PACKed array.	41.8
H000002	BOOLEAN operations on entire array (not packed).	41.8
H000003	BOOLEAN operations on components of a PACKed array.	235.1
H000004	BOOLEAN operations on components of an array (not packed).	235.1
H000005	Move INTEGER to INTEGER (Unchecked Conversion).	4.0
H000006	Move array of 10 Floats to record (Unchecked Conversion)	3.8
H000007	Store and extract bit fields, defined by representation clauses.	(1)
L000001	Simple "for" loop.	4.1
L000002	Simple "while" loop.	5.3
L000003	Simple "exit" loop.	5.1
L000004	Loop of 5 iterations with pragma OPTIMIZE (Time).	5.0
L000005	Loop of 5 iterations with pragma OPTIMIZE (Space).	5.0
P000001	Procedure call and return (inlineable), no parameters.	0.2
P000002	Procedure call and return (not inlineable), no parameters.	6.7
P000003	Procedure call and return (compiled separately).	6.3
P000004	Procedure call and return (Pragma INLINE used).	0.0

Guidelines to Select, Configure and Use an Ada Runtime Environment

Alsys PIWG results for MVME 133A (Continued). Clock: 20MHz, M68020, 1 Megabyte on board DRAM: 32 bit address and data access; 1 wait state. Checks ON, Optimizations ON, Timer configured to 1024 Hz TICK. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
P000005	Procedure call and return {one parameter, in INTEGER}.	7.2
P000006	Procedure call and return {one parameter, out INTEGER}.	8.6
P000007	Procedure call and return {one parameter, in out INTEGER}.	9.0
P000010	Procedure call and return {ten parameters, in INTEGER}.	19.6
P000011	Procedure call and return {twenty parameters, in INTEGER}.	34.4
P000012	Procedure call and return {ten parameters, in record_type}.	20.9
P000013	Procedure call and return {twenty parameters, in record_type}.	35.9
T000001	Minimum rendezvous, entry call and return.	163.8
T000002	Task entry call and return {one task, one entry}.	169.0
T000003	Task entry call and return {two tasks, one entry each}.	228.3
T000004	Task entry call and return {one task, two entries}.	327.9
T000005	Active entry and return (ten tasks, one entry each).	213.8
T000006	Task entry call and return (one task, ten entries).	711.6
T000007	Minimum rendezvous, entry call and return.	103.6
T000008	Passing an integer from producer to consumer	616.6

* WHETSTONE : units are in KWIPS not in microseconds.

(1) Uses bit level record representation clauses. Bit level representation clauses are not supported in version 3.5, but will be implemented for version 4.1 of the compiler (October 1988).

Guidelines to Select, Configure and Use an Ada Runtime Environment

Alsys PIWG results for MVME 133A. Clock: 20MHz, M68020, 1 Megabyte on board DRAM: 32 bit address and data access; 1 wait state. Stack overflow checking only enabled. Timer configured to 1024 Hz TICK. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone	280
A000092	Whetstone benchmarks, using manufacturer's math routines	542*
A000093	Whetstone benchmarks, using standard math routines	296*
C000001	Task creation/terminate, task type declared in package.	1377.1
C000002	Task creation/terminate, task type declared in procedure.	1463.5
C000003	Task creation/terminate, task type declared in block.	1454.3
D000001	Dynamic array, use and deallocation.	11.6
D000002	Dynamic array elaboration and initialization.	2432.8
D000003	Dynamic record allocation and deallocation.	37.8
D000004	Dynamic record elaboration and initialization.	2449.7
E000001	Raise and handle an exception locally.	1038.3
E000002	Raise and handle an exception in a package.	3201.6
E000003	Raise and handle an exception nested 3 deep in procedures.	5545.3
E000004	Raise and handle an exception nested 4 deep in procedures.	6219.3
E000005	Raise and handle an exception in a rendezvous.	10095.8
F000001	Set a BOOLEAN flag using a logical equation.	0.0
F000002	Set a BOOLEAN flag using an "if" test.	0.0
G000005	TEXT_IO.Get an INTEGER from a local string.	305.5
G000006	TEXT_IO.Get a FLOAT from a local string.	1704.3
H000001	BOOLEAN operations on entire PACKed array.	44.0 (1)
H000002	BOOLEAN operations on entire array (not packed).	44.0
H000003	BOOLEAN operations on components of a PACKed array.	104.8 (1)
H000004	BOOLEAN operations on components of an array (not packed).	104.8
H000005	Move INTEGER to INTEGER (Unchecked Conversion).	0.0
H000006	Move array of 10 Floats to record (Unchecked Conversion)	0.0
H000007	Store and extract bit fields, defined by representation clauses.	(2)
L000001	Simple "for" loop.	2.3
L000002	Simple "while" loop.	3.2
L000003	Simple "exit" loop.	3.4
L000004	Loop of 5 iterations with pragma OPTIMIZE (Time).	2.9
L000005	Loop of 5 iterations with pragma OPTIMIZE (Space).	2.9
P000001	Procedure call and return (inlineable), no parameters.	0.0
P000002	Procedure call and return (not inlineable), no parameters.	6.6
P000003	Procedure call and return (compiled separately).	9.0
P000004	Procedure call and return (Pragma INLINE used).	2.9

Guidelines to Select, Configure and Use an Ada Runtime Environment

Alsys PIWG results for MVME 133A (Continued). Clock: 20MHz, M68020, 1 Megabyte on board DRAM: 32 bit address and data access; 1 wait state. Stack overflow checking only enabled. Timer configured to 1024 Hz TICK. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
P000005	Procedure call and return (one parameter, in INTEGER).	10.0
P000006	Procedure call and return (one parameter, out INTEGER).	11.6
P000007	Procedure call and return (one parameter, in out INTEGER).	13.0
P000010	Procedure call and return (ten parameters, in INTEGER).	18.4
P000011	Procedure call and return (twenty parameters, in INTEGER).	38.1
P000012	Procedure call and return (ten parameters, in record_type).	19.6
P000013	Procedure call and return (twenty parameters, in record_type).	40.9
T000001	Minimum rendezvous, entry call and return.	162.6
T000002	Task entry call and return (one task, one entry).	169.0
T000003	Task entry call and return (two tasks, one entry each).	227.9
T000004	Task entry call and return (one task, two entries).	330.3
T000005	Active entry and return (ten tasks, one entry each).	213.4
T000006	Task entry call and return (one task, ten entries).	720.8
T000007	Minimum rendezvous, entry call and return.	102.8
T000008	Passing an integer from producer to consumer	622.9

* WHETSTONE : units are in KWIPS not in microseconds.

(1) Tests the effects of boolean operations on packed arrays. This feature is not implemented in version 3.5, but will be implemented for version 4.1 (October 1988).

(2) Uses bit level record representation clauses. Bit level representation clauses are not supported in version 3.5, but will be implemented for version 4.1 of the compiler (October 1988).

Guidelines to Select, Configure and Use an Ada Runtime Environment

RUNTIME STORAGE REQUIREMENTS

It depends on which executive is used (VRTX, VRTX32, or ARTK), but in general, these approximations apply:

**Max Sequential, approximately 25 K Bytes
Max Tasking, approximately 35 K Bytes
Max I/O, + 20 K Bytes.**

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: Determined by user.

Q2: How long, and for what reasons are interrupts disabled?

A2: Max approximately 20-30 microseconds for critical code.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: Based upon past implementations using Habermann-Nassi and Order of Arrival schemes for rendezvous execution, Alsys has abandoned these "optimizations" because they incur too much bookkeeping overhead. Accept bodies with empty statement lists are optimized to avoid context switching.

Q4: What are the restrictions for representation clauses?

A4: All representation clauses are currently supported to the byte level, except:

- bit level representation clauses (V4.1)
- pragma PACK
- change of representation for derived record types
- TSIZE for types declared in a generic unit
- TSMALL for fixed point types must be a power of 2, and the absolute value of the exponent must be less than 31
- enumeration clauses are not allowed if there is a range constraint on the parent subtype
- address representation clauses for program units
- address clauses for interrupt entries are only supported when there are no parameters for the task entry. An alternative mechanism is supplied for entries with a single parameter.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Preemptive, round-robin with time slicing option.

Q6: What are the restrictions on pragma INLINE?

A6: Routines must be non-recursive.

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: No.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Q9: What object types are supported by pragma SHARED?

A9: Scalars.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

- Max. No. of Tasks
- Task Time Slice Default
- Timer Resolution
- Exception Trace
- Default Stack Sizes
- Terminal I/O
- Optional Numeric Co-processor

Also see Technical Summary preceding this section.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
CAP Industry, Ltd. Compiler version 2.1	MicroVAX II (under MicroVMS 4.6)	80286, Intel iAPX 80286 protected mode (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Any part of a library unit being required loads the entire unit. (Unoptimized)
- Individual subprograms and/or data objects may be extracted from packages only. (Optimized)

II. Customization of the Runtime:

- By pragmas
- By compiler switches
- By linker switches
- By Modifying-Replacing the source to selective runtime routines provided with the purchase of the compiler (i.e. Device Drivers).

III. Documentation provided to help user configure runtime:

- CAPTACS - E286 Users Guide (Contains chapters such as "Programming Guide", "Configurability", and "Nonstandard Programming Interfaces").

IV. Services to customize the runtime:

- CAP Industry Ltd. does not provide services to customize the runtime for a particular application.

V. Cost of runtime source code:

- \$60,000

VI. Source of Information: Vendor Input

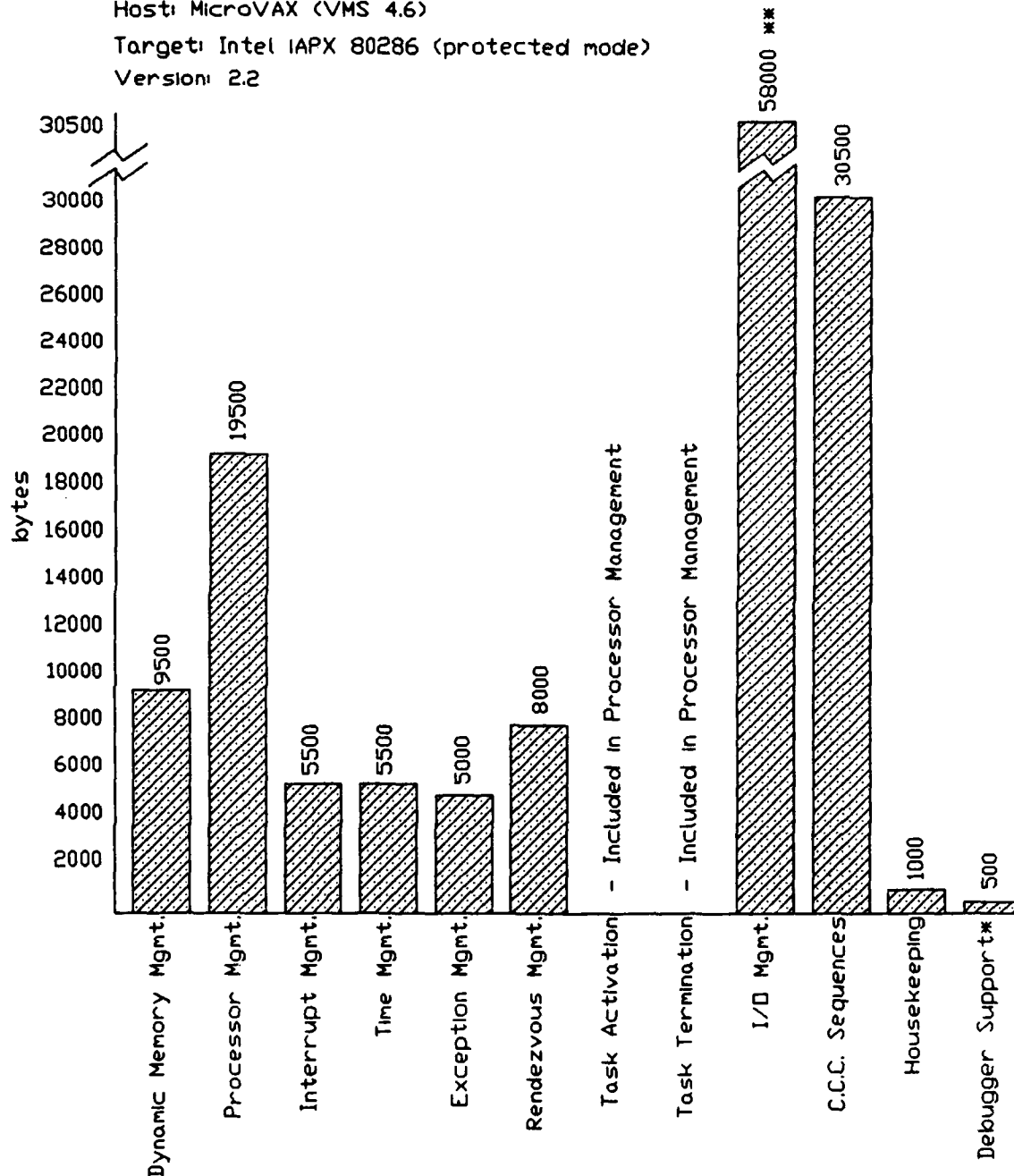
Guidelines to Select, Configure and Use an Ada Runtime Environment

CAP Industry Limited

Host: MicroVAX (VMS 4.6)

Target: Intel IAPX 80286 (protected mode)

Version: 2.2



Sum of ALL components = 143,000 bytes

* Component was supplied by vendor.

** Plus Driver (approximately 2,000 bytes)

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: 976.6 microseconds, configurable

Q2: How long, and for what reasons are interrupts disabled?

A2: Interrupts are disabled in order to protect the scheduler and memory manager while they are updating data structures.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: Synchronization rendezvous: If the accepting task has a null accept and equal or lower priority, the calling task will not stop, but the accepting task will become active.

Interrupts can be handled in the environment of the interrupted task if no interactions with other tasks occur during the rendezvous.

Q4: What are the restrictions for representation clauses?

A4: (1) Representation clauses are not supported for derived types.

(2) Enumeration representation clauses are not supported for CHARACTER and BOOLEAN types.

(3) Record representation clauses are supported with the following constraints

a. word alignment is mod 16

b. the ordering of bits within a byte is right to left.

(4) Length clause is supported:

a. for the attribute 'storage_size for task types.

b. for the attribute 'size. The value specified is checked to be sufficient but otherwise ignored.

c. for the attribute 'small.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Round-robin algorithm for tasks with the same priority.

Q6: What are the restrictions on pragma INLINE?

A6: None.

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: No.

Q9: What object types are supported by pragma SHARED?

A9: The restrictions on shared variables are only those specified in the LRM.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

Guidelines to Select, Configure and Use an Ada Runtime Environment

- **Maximum number of tasks (Heap space/Table space)**
- **Timer Resolution**
- **Default stack sizes**
- **Default task priority**
- **Optional numeric co-processor**
- **Fast interrupt entry**
- **Terminal I/O**

Additional items:

- **Device drivers**
- **Startup, normal and exception termination**
- **Number and type of interrupt devices**
- **Task lockup handling**

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the Intel 80286

package SYSTEM is

```
type SEG_OFFSET is new INTEGER;
type SEG_COLLECTOR is new INTEGER;
type ADDRESS is private;
type SUBPROGRAM_VALUE is private;
type NAME is (CAPTACS_E286);
type SYSTEM_NAME : constant NAME := CAPTACS_E286;
STORAGE_UNIT : constant := 8;
MEMORY_SIZE : constant := 2**24;
```

-- System-dependent declarations:

```
MIN_INT      : constant := -(2**31);
MAX_INT      : constant := (2**31) - 1;
MAX_DIGITS   : constant := 15;
MAX_MANTISSA : constant := 31;
FINE_DELTA   : constant := 1.0/(2**(MAX_MANTISSA - 1));
TICK         : constant := 1.0/(2**10);
```

-- Other system-dependent declarations:

```
subtype PRIORITY is INTEGR range 0 .. 15;
```

private

-- Types ADDRESS and SUBPROGRAM_VALUE are private

end SYSTEM;

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for the Intel 80286

Package STANDARD is not specified in Ada by CAPTACS. They provided information that indicates the following numeric types are supported.

long_integer is a predefined 32 bit integer type

float is a predefined 32 bit twos compliment floating point type, with 24 bits in the mantissa and an exponent range of -125 to +128

long_float is a predefined 64 bit twos compliment floating point type, with 53 bits in the mantissa and an exponent range of -1021 to +1024

short_fixed is a predefined 16 bit twos compliment fixed type

fixed is a predefined 32 bit twos compliment fixed type

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
DDC-I Compiler version 4.2	DEC MicroVAX II (under MicroVMS 4.4)	8086, Intel iSBC 86/05A, 8086, Titan SECS 86/20, 80186, Intel iSBC 186/03A 80286, Titan SECS 286/20 80386, Intel iSBC 386/21, (All bare machines)
Compiler version 4.2	DEC-VAX-11/7xx VAX-8xxx, VAX station, & MicroVAX Series (under VAX/VMS 4.6 or MicroVAX/VMS 4.6)	8086, Intel iSBC 86/35 80186, Intel iSBC 186/03A 80286, Intel iSBC 286/12 80286, Intel iSBC 286/12 (protected mode) 80386, Intel iSBC 386/21 80386, Intel iSBC 386/21 (protected mode) (All bare machines) *All Derived*
Compiler version 4.2	DEC MicroVAX II (under MicroVMS 4.4)	80386, Force CPU-386 VMEbus (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Only data objects that are referenced are allocated memory.

II. Customization of the Runtime:

- By pragmas
 - By compiler switches
 - By Modifying-Replacing the source to selective runtime routines provided with the purchase of the compiler (i.e. Device Drivers).
- The runtime system is divided into two parts:

The permanent part that is independent of the execution environment.

The user configurable part which consists of user configurable code and data. The user configurable code is typically a set of assembly language routines, called from the permanent part of the runtime system and generated Ada code.

Guidelines to Select, Configure and Use an Ada Runtime Environment

In the permanent part of the runtime system, when a module is not needed, it is not included during the linking process. In the user configurable code, it is up to the user to eliminate code that is not used. Eliminating unneeded user configurable modules can have a large effect in reducing the overall size of the RTS (it also reduces the number of modules in the permanent part as well), since the user configurable code makes calls to the permanent part.

There are two RTS versions supported: a tasking version and a non-tasking version.

Ada Linker Options:

- Maximum Number of Tasks
- Task Time Slice Default
- Timer Resolution
- Default Stack Sizes
- Default Task Priority
- Optional Numeric Co-processor

RTS Extension:

- Dynamic Task Priority
- Semaphore Operations
- Exception Trace
- Fast Interrupt Entry
- Terminal I/O
- ROMable Code
- RTS variant implements strict priority scheduling and priority inheritance.

Real-time Features Supported:

- Address Clauses
- Record Representation Clauses
- Length Clauses
- Enumeration Representation Clauses
- Interrupt Entries
- Machine Code Insertions
- Pragma Interface
- Pragma Inline
- All Chapter 13 Features

Tools: (Allows Standard Intel Tool Usage)

- In-circuit emulation
- Performance Analyzer
- Assembler, Linker, Locator
- Debuggers
- Powerful Symbolic Debugger (Q4 1988)

III. Documentation provided to help user configure runtime:

- Run-Time System detailed design for DACS-80x86 - DDC-I 5801/RPT/70 issue 1

Guidelines to Select, Configure and Use an Ada Runtime Environment

IV. Services to customize the runtime:

- Provided by DDC-I via training classes and consulting services.
- Cost: Daily consulting rates and expenses.

V. Cost of runtime source code:

- \$30,000 to \$50,000

VI. Source of Information: Vendor Input, Compiler Manuals, User Input.

Guidelines to Select, Configure and Use an Ada Runtime Environment

DDC-I PIWG results for DACS 8086. Clock : 8MHz, 1 wait-state, real mode, (all tests compiled with OPTIMIZE and NOCHECKS). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
A000091	"Dhrystone" benchmark	1684.5
A000093	"Whetstone" benchmark	119*
C000001	Task creation/terminate, task type declared in package.	2328.0
C000002	Task creation/terminate, task type declared in procedure.	1816.0
C000003	Task creation/terminate, task type declared in block.	1812.7
D000001	Dynamic array, use and deallocation.	40.0
D000002	Dynamic array elaboration and initialization.	32227.0
D000003	Dynamic record allocation and deallocation.	100.5
D000004	Dynamic record elaboration and initialization.	40830.0
E000001	Raise and handle an exception locally.	372.9
E000002	Raise and handle an exception in a package.	559.1
E000004	Raise and handle an exception nested 4 deep in procedures.	653.1
F000001	Set a BOOLEAN flag using a logical equation.	9.9
F000002	Set a BOOLEAN flag using an "if" test.	11.1
L000001	Simple "for" loop.	10.9
L000002	Simple "while" loop.	10.4
L000003	Simple "exit" loop.	10.4
P000001	Procedure call and return (inlineable), no parameters.	22.4
P000002	Procedure call and return (not inlineable), no parameters.	22.4
P000003	Procedure call and return (compiled separately).	19.9
P000004	Procedure call and return (Pragma INLINE used).	17.6
P000005	Procedure call and return (one parameter, in INTEGER).	24.9
P000006	Procedure call and return (one parameter, out INTEGER).	24.7
P000007	Procedure call and return (one parameter, in out INTEGER).	29.6
P000010	Procedure call and return (ten parameters, in INTEGER).	60.3
P000011	Procedure call and return (twenty parameters, in INTEGER).	99.3
P000012	Procedure call and return (ten parameters, in record_type).	120.2
P000013	Procedure call and return (twenty parameters, in record_type).	239.0
T000001	Minimum rendezvous, entry call and return.	430.3
T000002	Task entry call and return (one task, one entry).	426.6
T000003	Task entry call and return (two tasks, one entry each).	444.9
T000004	Task entry call and return (one task, two entries).	712.9
T000005	Active entry and return (ten tasks, one entry each).	416.3
T000006	Task entry call and return (one task, ten entries).	1102.5
T000007	Minimum rendezvous, entry call and return.	285.3

* A000093 : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

DDC-I PIWG results for DACS 80186. Clock : 8MHz, zero wait-states, real mode, (all tests compiled with OPTIMIZE and NOCHECKS). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
A000091	"Dhrystone" benchmark.	974.3
A000093	"Whetstone" benchmark.	145*
C000001	Task creation/terminate, task type declared in package.	1434.3
C000002	Task creation/terminate, task type declared in procedure.	1054.7
C000003	Task creation/terminate, task type declared in block.	1054.7
D000001	Dynamic array, use and deallocation.	24.9
D000002	Dynamic array elaboration and initialization.	21992.2
D000003	Dynamic record allocation and deallocation.	62.6
D000004	Dynamic record elaboration and initialization.	24101.6
E000001	Raise and handle an exception locally.	245.1
E000002	Raise and handle an exception in a package.	373.8
E000004	Raise and handle an exception nested 4 deep in procedures.	432.7
F000001	Set a BOOLEAN flag using a logical equation.	7.2
F000002	Set a BOOLEAN flag using an "if" test.	7.8
L000001	Simple "for" loop.	6.9
L000002	Simple "while" loop.	6.2
L000003	Simple "exit" loop.	6.6
P000001	Procedure call and return (inlineable), no parameters.	13.0
P000002	Procedure call and return (not inlineable), no parameters.	13.0
P000003	Procedure call and return (compiled separately).	10.6
P000004	Procedure call and return (Pragma INLINE used).	8.3
P000005	Procedure call and return (one parameter, in INTEGER).	13.5
P000006	Procedure call and return (one parameter, out INTEGER).	15.6
P000007	Procedure call and return (one parameter, in out INTEGER).	17.4
P000010	Procedure call and return (ten parameters, in INTEGER).	34.3
P000011	Procedure call and return (twenty parameters, in INTEGER).	58.4
P000012	Procedure call and return (ten parameters, in record_type).	80.3
P000013	Procedure call and return (twenty parameters, in record_type).	159.3
T000001	Minimum rendezvous, entry call and return.	255.9
T000002	Task entry call and return (one task, one entry).	253.4
T000003	Task entry call and return (two tasks, one entry each).	264.3
T000004	Task entry call and return (one task, two entries).	414.1
T000005	Active entry and return (ten tasks, one entry each).	247.3
T000006	Task entry call and return (one task, ten entries).	617.2
T000007	Minimum rendezvous, entry call and return.	168.0

*A000093 : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

DDC-I PIWG results for DACS 80286. Clock : 8MHz, zero wait-states, real mode, (all tests compiled with OPTIMIZE and NOCHECKS). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
A000091	"Dhrystone" benchmark.	483.9
A000093	"Whetstone" benchmark.	172*
C000001	Task creation/terminate, task type declared in package.	731.2
C000002	Task creation/terminate, task type declared in procedure.	557.3
C000003	Task creation/terminate, task type declared in block.	555.4
D000001	Dynamic array, use and deallocation.	10.6
D000002	Dynamic array elaboration and initialization.	10839.8
D000003	Dynamic record allocation and deallocation.	28.5
D000004	Dynamic record elaboration and initialization.	11445.3
E000001	Raise and handle an exception locally.	144.2
E000002	Raise and handle an exception in a package.	214.2
E000004	Raise and handle an exception nested 4 deep in procedures.	219.6
F000001	Set a BOOLEAN flag using a logical equation.	3.2
F000002	Set a BOOLEAN flag using an "if" test.	3.5
L000001	Simple "for" loop.	3.8
L000002	Simple "while" loop.	3.3
L000003	Simple "exit" loop.	3.8
P000001	Procedure call and return (inlineable), no parameters.	6.7
P000002	Procedure call and return (not inlineable), no parameters.	6.7
P000003	Procedure call and return (compiled separately).	6.4
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	7.4
P000006	Procedure call and return (one parameter, out INTEGER).	8.9
P000007	Procedure call and return (one parameter, in out INTEGER).	9.3
P000010	Procedure call and return (ten parameters, in INTEGER).	17.0
P000011	Procedure call and return (twenty parameters, in INTEGER).	28.0
P000012	Procedure call and return (ten parameters, in record_type).	31.5
P000013	Procedure call and return (twenty parameters, in record_type).	58.5
T000001	Minimum rendezvous, entry call and return.	136.6
T000002	Task entry call and return (one task, one entry).	134.6
T000003	Task entry call and return (two tasks, one entry each).	141.6
T000004	Task entry call and return (one task, two entries).	227.8
T000005	Active entry and return (ten tasks, one entry each).	132.2
T000006	Task entry call and return (one task, ten entries).	350.6
T000007	Minimum rendezvous, entry call and return.	93.9

*A000093 : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

DDC-I PIWG results for DACS 80386. Clock : 16MHz, zero wait-states, real- mode, (all tests compiled with OPTIMIZE and NOCHECKS). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
A000091	"Dhrystone" benchmark.	240.6
A000093	"Whetstone" benchmark.	776*
C000001	Task creation/terminate, task type declared in package.	343.6
C000002	Task creation/terminate, task type declared in procedure.	265.2
C000003	Task creation/terminate, task type declared in block.	264.1
D000001	Dynamic array, use and deallocation.	5.1
D000002	Dynamic array elaboration and initialization.	5605.5
D000003	Dynamic record allocation and deallocation.	15.5
D000004	Dynamic record elaboration and initialization.	6303.7
E000001	Raise and handle an exception locally.	64.4
E000002	Raise and handle an exception in a package.	97.2
E000004	Raise and handle an exception nested 4 deep in procedures.	104.9
F000001	Set a BOOLEAN flag using a logical equation.	1.5
F000002	Set a BOOLEAN flag using an "if" test.	1.4
L000001	Simple "for" loop.	1.7
L000002	Simple "while" loop.	1.6
L000003	Simple "exit" loop.	1.7
P000001	Procedure call and return (inlineable), no parameters.	4.3
P000002	Procedure call and return (not inlineable), no parameters.	4.3
P000003	Procedure call and return (compiled separately).	4.0
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	4.5
P000006	Procedure call and return (one parameter, out INTEGER).	5.0
P000007	Procedure call and return (one parameter, in out INTEGER).	5.3
P000010	Procedure call and return (ten parameters, in INTEGER).	9.4
P000011	Procedure call and return (twenty parameters, in INTEGER).	14.1
P000012	Procedure call and return (ten parameters, in record_type).	17.6
P000013	Procedure call and return (twenty parameters, in record_type).	33.2
T000001	Minimum rendezvous, entry call and return.	67.8
T000002	Task entry call and return (one task, one entry).	67.7
T000003	Task entry call and return (two tasks, one entry each).	70.6
T000004	Task entry call and return (one task, two entries).	110.0
T000005	Active entry and return (ten tasks, one entry each).	66.2
T000006	Task entry call and return (one task, ten entries).	164.6
T000007	Minimum rendezvous, entry call and return.	45.7

*A000093 : units are in KWIPS not in microseconds.

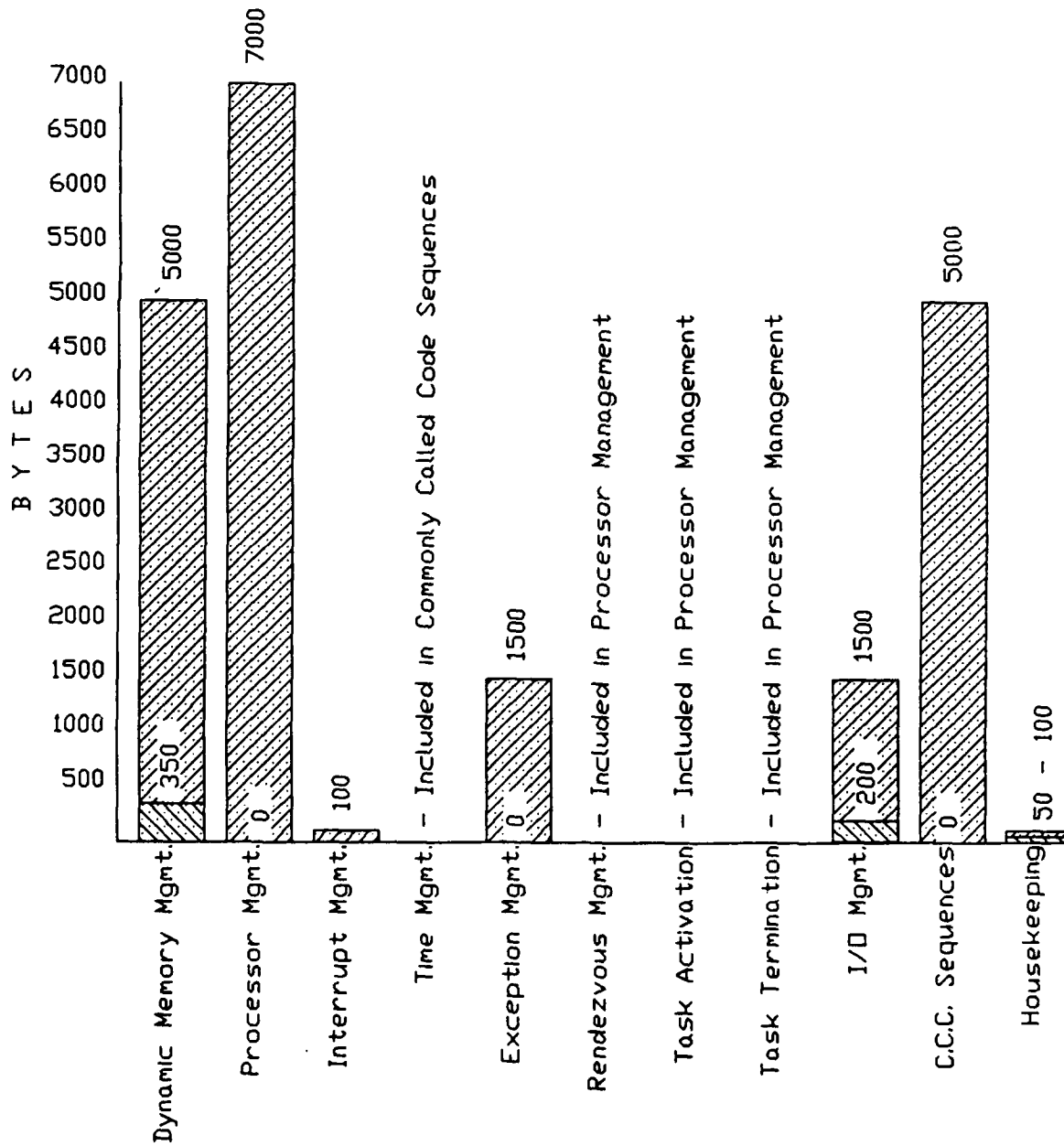
Guidelines to Select, Configure and Use an Ada Runtime Environment

DDC-I, Inc.

Host: VAX / VMS

Target: Intel 8086, 80186, 80286, 80386

Version: 4.2



- Sum Of All Components = 700 - 20200 bytes

Guidelines to Select, Configure and Use an Ada Runtime Environment

Appendix F Notes

The following excerpts are taken from the DDC-I Ada Compiler System User's Guide. [10]

Representation Clause Restrictions

The DACS-80x86 fully supports the 'SIZE representation for derived types.

Length Clause

When using the SIZE attribute for discrete types, the maximum value that can be specified is 16 bits.

SIZE is only obeyed for discrete types when the type is a part of a composite object, e.g. arrays or records, for example:

```
type byte is range 0..255;
for byte'size use 8;

sixteen_bits_allocated : byte; -- one word allocated

eight_bit_per_element : array (0..7) of byte; -- four words allocated

type rec is
  record
    c1, c2 : byte; -- eight bits per component
  end record;
```

Using the STORAGE_SIZE attribute for a collection will set an upper limit on the total size of objects allocated in this collection. If further allocation is attempted, the exception STORAGE_ERROR is raised.

When STORAGE_SIZE is specific in a length clause for a task, the process stack area will be of the specified size. The process stack area will be allocated inside the "standard" stack segment.

Enumeration Representation Clause

Enumeration representation clauses may specify representations in the range of INTEGER'FIRST + 1..INTEGER'LAST - 1.

Record Representation Clauses

When representation clauses are applied to records the following restrictions are imposed.

- the component type is a discrete type different from LONG_INTEGER
- the component type is an array with a discrete element type different from LONG_INTEGER

Guidelines to Select, Configure and Use an Ada Runtime Environment

- the storage unit is 16 bits
- a record occupies an integral number of storage units
- a record may take up a maximum of 32K storage units
- a component must be specified with its proper size (in bits), regardless of whether the component is an array or not
- if a non-array component has a size which equals or exceeds one storage unit (16 bits) the component must start on a storage unit boundary, i.e. the component must be specified as:

component at N range 0..16 * M - 1;

where N specifies the relative storage unit number (0,1,...) from the beginning of the record, and M the require number of storage units (1, 2, ...)

- the elements in an array component should always be wholly contained in one storage unit
- if a component has a size which is less than one storage unit, it must be wholly contained within a single storage unit:

component at N range X..Y;

where N is a s in previous paragraph, and $0 \leq X \leq Y \leq 15$.

When dealing with PACKED ARRAY the following should be noted:

- the elements of the array are packed into 1, 2, 4 or 8 bits

If the record type contains components which are not covered by a component clause, they are allocated consecutively after the component with the value. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler.

Alignment Clauses

Alignment clauses for records are implemented with the following characteristics:

- If the declaration of the record type is done at the outermost level in a library package, any alignment is accepted.
- If the record declaration is done at a given static level (higher than the outermost library level, i.e., the permanent area), only word alignments are accepted.
- Any record object declared at the outermost level in a library package will be aligned according to the alignment clause specified for the type. Record objects declared elsewhere can only be aligned on a word boundary. If the record type has been associated a different alignment, an error message will be issued.
- If a record type with an associated alignment clause is used in a composite type, the alignment is required to be one word; an error message is issued if this is not the case.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Implementation-Dependent Names for Implementation-Dependent Components

None defined by the compiler.

Address Clauses

This section describes the implementation of address clauses and what types of entities may have their address specified by the user.

Objects

Address clauses are supported for scalar and composite objects whose size can be determined at compile time.

Task Entries

The implementation supports two methods to equate a task entry to a hardware interrupt through an address clause:

- 1.) Direct transfer of control to a task accept statement when an interrupt occurs (requires use of the pragma `INTERRUPT_HANDLER`).
- 2.) Mapping of an interrupt onto a normal conditional entry call, i.e., the entry can be called from other tasks without special actions, as well as being called when an interrupt occurs.

Fast Interrupt Entry

Directly transferring control to an accept statement when an interrupt occurs requires the implementation dependent pragma `INTERRUPT_HANDLER` to tell the compiler that the task is an interrupt handler. By using this pragma, the user is agreeing to place certain restrictions on the task in order to speed up the software response to the hardware interrupt. Consequently, use of this method to capture interrupts is much more efficient than the general method.

The following constraints are placed on the task:

- 1.) It must be a task object, i.e., not a task type.
- 2.) The pragma must appear first in the specification of the task object.
- 3.) All entries of the task object must be single entries with no parameters.
- 4.) The entries must not be called from any task.
- 5.) The body of the task object must not contain anything other than simple accept statements (potentially enclosed in

Guidelines to Select, Configure and Use an Ada Runtime Environment

a loop) referencing only global variables, i.e., no local variables. In the statement list of a simple accept statement, it is allowed to call normal single and parameterless, entries of other tasks, but no other tasking constructs are allowed. The call to another task entry, in this case, will not lead to an immediate task context switch, but will return to the caller when complete. Once the accept is completed, the task priority rules will be obeyed, and a context switch may occur.

Normal Interrupt Entry

Mapping of an interrupt onto a normal conditional entry call puts the following constraints on the involved entries and tasks:

- 1.) The affected entries must be defined in a task object only (not a task type).
- 2.) The entries must be single and parameterless.

Any interrupt entry, which is not found in an interrupt handler (first method), will lead to an update of the interrupt vector segment at link time. The interrupt vector segment will be updated to point to the interrupt routine generated by the compiler to make the task entry call. The interrupt vector segment is part of the user configurable data and consists of a segment, preset to the "standard" interrupt routines (e.g., `constraint_error`).

Unchecked Conversions

Unchecked conversion is only allowed between objects of the same "size".

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the DACS-80X86

package SYSTEM is

```
type Word is new Integer;
type LongWord is new Long_Integer;
type UnsignedWord is range 0..65535;
for UnsignedWord'SIZE use 16;
```

```
subtype SegmentId is UnsignedWord;
```

```
type Address is record
  offset : UnsignedWord;
  segment : SegmentId;
end record;
```

```
subtype Priority is Word range 0..31;
```

```
type Name is (iAPX86, iAPX186, iAPX286, iAPX386);
```

```
System_Name : constant Name := iAPX186;
Storage_Unit : constant      := 16;
Memory_Size : constant      := 1_048_576;
Min_Int      : constant      := -2_147_483_647 - 1;
Max_Int      : constant      := 2_147_483_647;
Max_Digits   : constant      := 15;
Max_Mantissa : constant      := 31;
Fine_Delta   : constant      := 2.0 / MAX_INT;
Tick         : constant      := 0.000_000_125;
```

```
type Interface_Language is ( PLM86, ASM86);
```

```
type ExceptionId is record
  unit_number : UnsignedWord;
  unique_number : UnsignedWord;
end record;
```

```
type TaskValue is new Integer;
type AccTaskValue is access TaskValue;
```

```
type Semaphore is
  record
    counter : UnsignedWord;
    first   : TaskValue;
    last    : TaskValue;
  end record;
```

```
InitSemaphore : constant Semaphore'(1, 0, 0);
```

end SYSTEM;

Guideline to Select, Configure, and Use an Ada Runtime Environment

Description of Package STANDARD for DACS 80X86 Bare Machine Target

Integer Types

Three predefined integer types are implemented, SHORT_INTEGER, INTEGER, and LONG_INTEGER.

They have the following attributes:

Real Address Mode and 286 Protected Mode:

```
SHORT_INTEGER'FIRST = -128
SHORT_INTEGER'LAST  = 127
SHORT_INTEGER'SIZE   = 16

INTEGER'FIRST       = -32_768
INTEGER'LAST        = 32_767
INTEGER'SIZE         = 16

LONG_INTEGER'FIRST  = -2_147_483_648
LONG_INTEGER'LAST   = 2_147_483_647
LONG_INTEGER'SIZE   = 32
```

386 Protected Mode:

```
SHORT_INTEGER'FIRST = -32_768
SHORT_INTEGER'LAST  = 32_767
SHORT_INTEGER'SIZE   = 16

INTEGER'FIRST       = -2**31
INTEGER'LAST        = 2**31-1
INTEGER'SIZE         = 32

LONG_INTEGER'FIRST  = -2**63
LONG_INTEGER'LAST   = 2**63-1
LONG_INTEGER'SIZE   = 64
```

Floating Point Types

Two predefined floating point types are implemented, FLOAT and LONG_FLOAT. They have the following attributes:

```
FLOAT'DIGITS          = 6
FLOAT'EPSILON          = 9.53674316406250E-07
FLOAT'FIRST            = -3.40282366920938E+38
FLOAT'LARGE            = 1.93428038904620E+25
FLOAT'LAST             = 3.40282366920938E+38
FLOAT'MACHINE_EMAX     = 126
FLOAT'MACHINE_EMIN     = -127
```

Guideline to Select, Configure, and Use an Ada Runtime Environment

Package STANDARD for DACS 80X86 Bare Machine Target (Continued)

```

FLOAT'MACHINE_MANTISSA = 24
FLOAT'MACHINE_OVERFLOWS = TRUE
FLOAT'MACHINE_RADIX = 2
FLOAT'MACHINE_ROUNDS = TRUE
FLOAT'MANTISSA = 21
FLOAT'SAFE_EMAX = 126
FLOAT'SAFE_LARGE = 8.50705917302346E+37
FLOAT'SAFE_SMALL = 5.87747175411144E-39
FLOAT'SIZE = 32

LONG_FLOAT'DIGITS = 15
LONG_FLOAT'EPSILON = 8.88178419700125E-16
LONG_FLOAT'FIRST = -1.7976931348623157E+308
LONG_FLOAT'LARGE = 2.57110087081438E+61
LONG_FLOAT'LAST = 1.7976931348623157E+308
LONG_FLOAT'MACHINE_EMAX = 1023
LONG_FLOAT'MACHINE_EMIN = -1023
LONG_FLOAT'MACHINE_MANTISSA = 53
LONG_FLOAT'MACHINE_OVERFLOWS = TRUE
LONG_FLOAT'MACHINE_RADIX = 2
LONG_FLOAT'MACHINE_ROUNDS = TRUE
LONG_FLOAT'MANTISSA = 51
LONG_FLOAT'SAFE_EMAX = 1023
LONG_FLOAT'SAFE_LARGE = 4.49423283715579E+307
LONG_FLOAT'SAFE_SMALL = 2.22507385850720E-308
LONG_FLOAT'SIZE = 64
```

Fixed Point Types

Two kinds of anonymous predefined fixed point types are implemented, named `FIXED` and `LONG_FIXED`. Note that these names are not defined in package `STANDARD`, but only used here for reference.

16 bits are used for the representation of `FIXED` types, and 32 bits are used for the representation of `LONG_FIXED` types.

For each of `FIXED` and `LONG_FIXED` there exists a virtual predefined type for each possible value of `SMALL`. The possible values of `SMALL` are the powers of two that are representable by a `LONG_FLOAT` value.

The lower and upper bounds of these types are:

```

lower bound of FIXED types      = -32_768 * SMALL
upper bound of FIXED types      = 32_767 * SMALL
lower bound of LONG_FIXED types = -2_147_483_648 * SMALL
upper bound of LONG_FIXED types = 2_147_483_647 * SMALL
```

Guideline to Select, Configure, and Use an Ada Runtime Environment

Package STANDARD for DACS 80X86 Bare Machine Target (Continued)

A user defined fixed point type is represented as that predefined `FIXED` or `LONG_FIXED` type which has the largest value of `SMALL` not greater than the user-specified `DELTA`, and which has the smallest range that includes the user-specified range.

Any fixed point type `T` has the following attributes:

```
T'MACHINE_OVERFLOWS = TRUE
T'MACHINE_ROUNDS    = FALSE
```

The Type `DURATION`

The predefined fixed point type `DURATION` has the following attributes:

```
DURATION'AFT          = 5
DURATION'DELTA        = DURATION'SMALL
DURATION'FIRST        = -131_072.00000
DURATION'FORE         = 7
DURATION'LARGE        = 1.31071999938965E05
DURATION'LAST         = 131_071.00000
DURATION'MANTISSA     = 31
DURATION'SAFE_LARGE   = 1.31071999938965E05
DURATION'SAFE_SMALL   = DURATION'SMALL
DURATION'SIZE         = 32
DURATION'SMALL        = 6.10351562500000E-05 = 2**(-14)
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Digital Equipment Corp. Compiler version 1.5	VAX 8800 (under VAX/VMS, Version 4.7)	MicroVAX II (under VAXELN Toolkit, Version 3.0 in Combination with VAXELN Ada, Version 1.2)
Compiler version 1.5	All members of the VAX family: MicroVAX I, VAXstation I, MicroVAX II, VAXstation II, VAXstation 2000 (under MicroVMS, version 4.7); MicroVAX 3500 & 3600; VAXserver 3500, 3600, & 3602; and VAXstation 3200, 3500 (under VAX/VMS version 4.7A); VAX-11/730, 750, 780, 782, 785, VAX 8200, 8250, 8300, 8350, 8530,8550, 8600, 8650, 8700, and 8800 (under VAX/VMS, version 4.7)	Any of the following configurations: MicroVAX I & II; rtVAX 1000; KA620 (rtVAX 1000 processor board); MicroVAX 3500 & 3600; VAX-11/730 & 750; and VAX 8500, 8530, 8550, 8700, & 8800 (under VAXELN Toolkit, version 3.0 in combination with VAXELN Ada version 1.2) *Derived*

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Any part of a library unit being required loads the entire unit.

II. Customization of the Runtime:

- By pragmas
- By linker switches
- By modifying/replacing the source to selective runtime routines provided by the compiler vendor with the purchase of the compiler(i.e device drivers, etc).

III. Documentation provided to help user configure runtime:

- The "VAXELN Ada User's Manual" and "VAX Ada Run-Time Reference Manual".

Guidelines to Select, Configure and Use an Ada Runtime Environment

IV. Services to customize the runtime:

- Services to customize the runtime are not available by DEC.

V. Cost of runtime source code:

- The runtime source code is not for sale.

VI. Source of Information: Vendor Input.

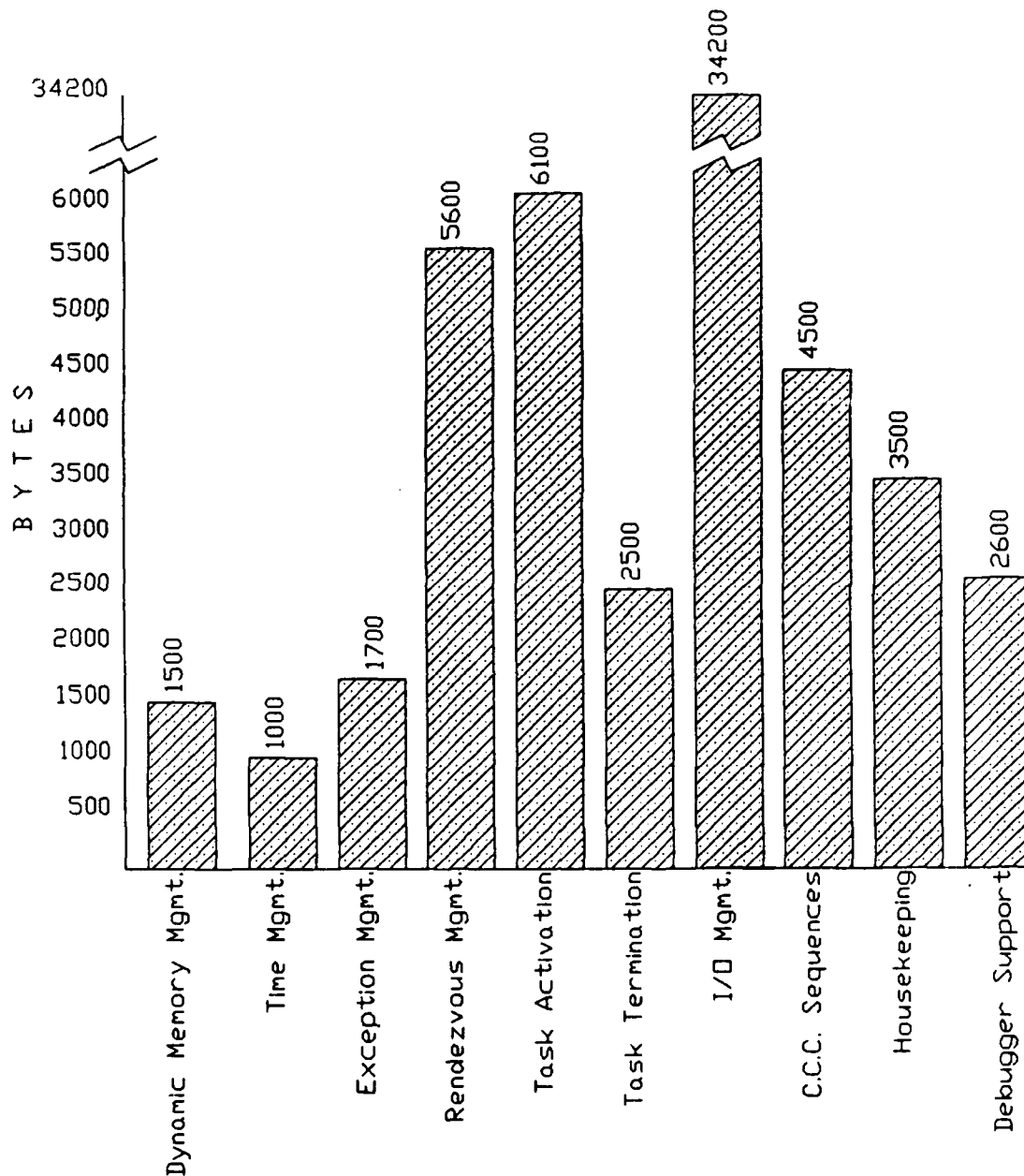
Guidelines to Select, Configure and Use an Ada Runtime Environment

Digital Equipment Corp.

Host: VAX / VMS

Target: Micro VAX (under VAXELN toolkit)

Version: 1.5



- Sum of All Components = 63,200 bytes

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: 10 milliseconds

Q2: How long, and for what reasons are interrupts disabled?

A2: VAXELN Ada runtime does not disable interrupts.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: The runtime uses inlined mutual exclusion operations to control access to resources needed during rendezvous. It also performs deferred task switching. When an interrupt arrives for a task that is the same priority as the active task, no switching occurs until the current task becomes blocked.

Q4: What are the restrictions for representation clauses?

A4: Generally, VAXELN Ada supports all implementation-dependent facilities of chapter 13 that have useful and desirable interpretation in the VAXELN environment.

Pragma PACK is supported. For a size specification for a discrete type, the given size must not exceed 32 bits; the given size becomes the default allocation for all objects and components of that type. For all other types, the given size must equal the size that would apply in the absence of a size specification.

For a collection size specification, the given size becomes the initial and maximum size of the collection. In the absence of a collection size specification, or for a size specification of zero, no storage is initially allocated for a collection, and the collection is extended as needed (until all virtual memory for the process is exhausted). If the value is less than zero, CONSTRAINT_ERROR is raised.

For a task storage specification, the given size becomes the initial and maximum size for the task activation (the task stack size). In the absence of a specification, or for a specification of zero, a default size is used. In either case the task stack size is fixed at activation and is not extendible. If the value is less than zero, CONSTRAINT_ERROR is raised.

For the specification of SMALL for a fixed point type, the given value must be a power of 2.0 (2.0^{*N} , where $-31 \leq N \leq 31$) that is less than or equal to the delta of the type, and that also satisfies the specified range of the type.

The implementation defined pragma TASK_STORAGE allows the specification of guard pages for a task stack. (Guard pages form an area of memory which has no read or write access and which thus helps in the detection of stack overflow (STORAGE_ERROR) when non-Ada code is called from a task.

The implementation defined pragma MAIN_STORAGE allows the specification of a fixed size stack and guard pages for the main program. In a VAXELN Ada program, the main program stack is always fixed and is not extended as needed. Thus Pragma MAIN_STORAGE is intended in particular to allow VAXELN Ada task stack sizes to be adjusted and to allow the simulation of the VAXELN task stack implementation on a VAX/VMS system.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Enumeration representation clauses are supported. Signed and unsigned representation in the range `MIN_INT .. MAX_INT` are allowed.

Record representation clauses are supported. The value in an alignment clause must be a power of 2 (2^N , where $0 \leq N \leq 30$). For stack objects, the alignment must not exceed 4 (longword alignment). For statically and collection- allocated (heap-allocated) objects, alignments up to 512 are supported.

VAXELN Ada distinguishes between types that are bit-alignable and those that are byte alignable. Components of bit-alignable types can be allocated beginning at arbitrary bit offsets in component clauses, while components of byte alignable types must be allocated at byte (addressable storage) boundaries. Generally, discrete types, and record types whose size is 32 bits or less are bit-alignable while other types are not.

VAXELN Ada supports address representation clauses for variables, but does not support address representation clauses for constants, subprogram, package, or task units, or single entries.

The representation attributes `ADDRESS`, `SIZE`, `POSITION`, `FIRST_BIT`, `LAST_BIT`, and `STORAGE_SIZE` are supported. The implementation-defined attribute `BIT` yields the bit offset within a storage unit of the first bit allocated to an object (a value from 0 to 7). The implementation attribute `MACHINE_SIZE` yields the actual size that is allocated for a variable of a type or subtype, taking into account the storage alignment and padding conventions of the VAX Ada compiler.

The floating point representation attributes `MACHINE_RADIX`, `MACHINE_MANTISSA`, `MACHINE_EMAX`, `MACHINE_EMIN`, `MACHINE_ROUNDS`, and `MACHINE_OVERFLOW` are also supported.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: The VAXELN Ada RTE supports a "run-until-blocked with higher priority preemption" scheduling algorithm. This algorithm lets a task run until it is either blocked or a task of a higher priority becomes runnable.

Q6: What are the restrictions on `pragma INLINE`?

A6: `Pragma INLINE` can be used to explicitly expand inline a subprogram declaration, body, or generic subprogram provided it meets the following conditions:

1. Neither its parameters or (in the case of functions) its result can be of type task type or of a composite type that has components of a task type.
2. For functions, the function result cannot be an unconstrained array type or an unconstrained type with discriminants.
3. The body of the subprogram cannot contain any of the following:
 1. A subprogram body, task or generic declaration or body stub.
 2. A package body.
 3. An exception declaration.
 4. An access type declaration.

Guidelines to Select, Configure and Use an Ada Runtime Environment

5. An array or record type declaration.
6. Any dependent task.
7. Any subprogram call that denotes the given subprogram or any containing subprogram, either directly or by means of renaming.

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: Machine code inserts are not supported. However VAXELN Ada does provide routines to read and write to processor and device registers. VAXELN Ada also provides routines to access VAX interlocked machine instructions.

Q9: What object types are supported by **pragma SHARED**?

A9: VAXELN does not support **pragma SHARED**. It does support the implementation defined **pragma VOLATILE**, which guarantees that a variable is allocated in main memory from which the value is fetched and to which the value is updated on each use. Unlike **pragma SHARED**, **pragma VOLATILE** does not force synchronization. **Pragma VOLATILE** can be used with variables of any type, including composite variables.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

- Terminal I/O
- Task stack size
- Sharable or nonsharable runtime

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for VAXELN Ada.

package SYSTEM is

type Name is (VAX_VMS, VAXELN);

System_Name : constant Name := VAX_VMS

Storage_Unit : constant := 8;

Memory_Size : constant := (2**31) - 1;

Min_Int : constant := - (2**31);

Max_Int : constant := (2**31) - 1;

Max_Digits : constant := 33;

Max_Mantissa : constant := 31;

Fine_Delta : constant := 2.0 ** (-31);

Tick : constant := 10.0 ** (-2);

subtype Priority is Integer range 0 .. 15;

-- Address type

--

type ADDRESS is private;

ADDRESS_ZERO : constant ADDRESS;

function "+" (LEFT : ADDRESS; RIGHT : INTEGER) return ADDRESS;

function "+" (LEFT : INTEGER; RIGHT : ADDRESS) return ADDRESS;

function "-" (LEFT : ADDRESS; RIGHT : ADDRESS) return INTEGER;

function "-" (LEFT : ADDRESS; RIGHT : INTEGER) return ADDRESS;

-- function "=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;

-- function "/=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;

function "<" (LEFT, RIGHT : ADDRESS) return BOOLEAN;

function "<=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;

function ">" (LEFT, RIGHT : ADDRESS) return BOOLEAN;

function ">=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;

-- Note that because ADDRESS is a private type

-- the function "=" and "/=" are already available and

-- do not have to be explicitly defined.

generic

type TARGET is private;

function FETCH_FROM_ADDRESS (A : ADDRESS) return TARGET;

generic

type TARGET is private;

function ASSIGN_TO_ADDRESS (A : ADDRESS; T : TARGET);

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for VAXELN Ada (Continued).

-- VAX Ada floating point type declaration for the VAX
-- hardware floating-point data types

```
type F_FLOAT is (digits 6);  
type D_FLOAT is (digits 9);  
type G_FLOAT is (digits 15);  
type H_FLOAT is (digits 33);
```

```
type TYPE_CLASS is (TYPE_CLASS_ENUMERATION,  
                    TYPE_CLASS_INTEGER,  
                    TYPE_CLASS_FIXED_POINT,  
                    TYPE_CLASS_FLOATING_POINT,  
                    TYPE_CLASS_ARRAY,  
                    TYPE_CLASS_RECORD,  
                    TYPE_CLASS_ACCESS,  
                    TYPE_CLASS_TASK,  
                    TYPE_CLASS_ADDRESS);
```

-- AST handler type

```
type AST_HANDLER is limited private;
```

```
NO_AST_HANDLER : constant AST_HANDLER;
```

-- Non-Ada exception

```
NON_ADA_ERROR : exception;
```

-- VAX hardware-oriented types and functions

```
type BIT_ARRAY is array (INTEGER range <>) of BOOLEAN;  
pragma PACK(BIT_ARRAY);
```

```
subtype BIT_ARRAY_8      is BIT_ARRAY (0..7);  
subtype BIT_ARRAY_16     is BIT_ARRAY (0..15);  
subtype BIT_ARRAY_32     is BIT_ARRAY (0..31);  
subtype BIT_ARRAY_64     is BIT_ARRAY (0..63);
```

```
type UNSIGNED_BYTE      is range 0..255;  
for UNSIGNED_BYTE'SIZE  use 8;
```

```
function "not" (LEFT      : UNSIGNED_BYTE) return UNSIGNED_BYTE;  
function "and" (LEFT, RIGHT : UNSIGNED_BYTE) return UNSIGNED_BYTE;  
function "or"  (LEFT, RIGHT : UNSIGNED_BYTE) return UNSIGNED_BYTE;  
function "xor" (LEFT, RIGHT : UNSIGNED_BYTE) return UNSIGNED_BYTE;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for VAXELN Ada (Continued).

```
function TO_UNSIGNED_BYTE (X : BIT_ARRAY_8) return UNSIGNED_BYTE;
function TO_BIT_ARRAY_8 (X : UNSIGNED_BYTE) return BIT_ARRAY_8;

type UNSIGNED_BYTE_ARRAY is array(INTEGER range <>) of UNSIGNED_BYTE;

type UNSIGNED_WORD is range 0 .. 65535;
for UNSIGNED_WORD'SIZE use 16;

function "not" (LEFT : UNSIGNED_WORD) return UNSIGNED_WORD;
function "and" (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;
function "or" (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;
function "xor" (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;

function TO_UNSIGNED_WORD (X : BIT_ARRAY_16) return UNSIGNED_WORD;
function TO_BIT_ARRAY_16 (X : UNSIGNED_WORD) return BIT_ARRAY_16;

type UNSIGNED_WORD_ARRAY is array(INTEGER range <>) of UNSIGNED_WORD;

type UNSIGNED_LONG_WORD is range MIN_INT .. MAX_INT;
for UNSIGNED_WORD'SIZE use 32;

function "not" (LEFT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
function "and" (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
function "or" (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
function "xor" (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;

function TO_UNSIGNED_LONGWORD(X : BIT_ARRAY_32) return UNSIGNED_LONGWORD;
function TO_BIT_ARRAY_32 (X : UNSIGNED_LONGWORD) return BIT_ARRAY_32;

type UNSIGNED_LONGWORD_ARRAY is
  array (INTEGER range <>) of UNSIGNED_LONGWORD;

type UNSIGNED_QUADWORD is record
  L0 : UNSIGNED_LONGWORD;
  L1 : UNSIGNED_LONGWORD;
end record;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for VAXELN Ada (Continued).

```
function "not" (LEFT : UNSIGNED_QUADWORD) return UNSIGNED_QUADWORD;  
function "and" (LEFT, RIGHT: UNSIGNED_QUADWORD) return UNSIGNED_QUADWORD;  
function "or" (LEFT, RIGHT: UNSIGNED_QUADWORD) return UNSIGNED_QUADWORD;  
function "xor" (LEFT, RIGHT: UNSIGNED_QUADWORD) return UNSIGNED_QUADWORD;
```

```
function TO_UNSIGNED_QUADWORD(X : BIT_ARRAY_64) return UNSIGNED_QUADWORD;  
function TO_BIT_ARRAY_64 (X : UNSIGNED_QUADWORD) return BIT_ARRAY_64;
```

```
type UNSIGNED_QUADWORD_ARRAY is  
  array (INTEGER range <>) of UNSIGNED_QUADWORD;
```

```
function TO_ADDRESS (X : INTEGER) return ADDRESS;  
function TO_ADDRESS (X : UNSIGNED_LONGWORD) return ADDRESS;  
function TO_ADDRESS (X : {universal integer}) return ADDRESS;
```

```
function TO_INTEGER (X : ADDRESS) return INTEGER;  
function TO_UNSIGNED_LONGWORD (X : ADDRESS) return UNSIGNED_LONGWORD;
```

```
function TO_UNSIGNED_LONGWORD (X : AST_HANDLER) return UNSIGNED_LONGWORD;
```

-- Conventional names for static subtypes of type UNSIGNED_LONGWORD

```
subtype UNSIGNED_1 is UNSIGNED_LONGWORD range 0 .. 2** 1-1;  
subtype UNSIGNED_2 is UNSIGNED_LONGWORD range 0 .. 2** 2-1;  
subtype UNSIGNED_3 is UNSIGNED_LONGWORD range 0 .. 2** 3-1;  
subtype UNSIGNED_4 is UNSIGNED_LONGWORD range 0 .. 2** 4-1;  
subtype UNSIGNED_5 is UNSIGNED_LONGWORD range 0 .. 2** 5-1;  
subtype UNSIGNED_6 is UNSIGNED_LONGWORD range 0 .. 2** 6-1;  
subtype UNSIGNED_7 is UNSIGNED_LONGWORD range 0 .. 2** 7-1;  
subtype UNSIGNED_8 is UNSIGNED_LONGWORD range 0 .. 2** 8-1;  
subtype UNSIGNED_9 is UNSIGNED_LONGWORD range 0 .. 2** 9-1;  
subtype UNSIGNED_10 is UNSIGNED_LONGWORD range 0 .. 2**10-1;  
subtype UNSIGNED_11 is UNSIGNED_LONGWORD range 0 .. 2**11-1;  
subtype UNSIGNED_12 is UNSIGNED_LONGWORD range 0 .. 2**12-1;  
subtype UNSIGNED_13 is UNSIGNED_LONGWORD range 0 .. 2**13-1;  
subtype UNSIGNED_14 is UNSIGNED_LONGWORD range 0 .. 2**14-1;  
subtype UNSIGNED_15 is UNSIGNED_LONGWORD range 0 .. 2**15-1;  
subtype UNSIGNED_16 is UNSIGNED_LONGWORD range 0 .. 2**16-1;  
subtype UNSIGNED_17 is UNSIGNED_LONGWORD range 0 .. 2**17-1;  
subtype UNSIGNED_18 is UNSIGNED_LONGWORD range 0 .. 2**18-1;  
subtype UNSIGNED_19 is UNSIGNED_LONGWORD range 0 .. 2**19-1;  
subtype UNSIGNED_20 is UNSIGNED_LONGWORD range 0 .. 2**20-1;  
subtype UNSIGNED_21 is UNSIGNED_LONGWORD range 0 .. 2**21-1;  
subtype UNSIGNED_22 is UNSIGNED_LONGWORD range 0 .. 2**22-1;
```


Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for VAXELN Ada (Continued).

```
subtype UNSIGNED_23 is UNSIGNED_LONGWORD range 0 .. 2**23-1;
subtype UNSIGNED_24 is UNSIGNED_LONGWORD range 0 .. 2**24-1;
subtype UNSIGNED_25 is UNSIGNED_LONGWORD range 0 .. 2**25-1;
subtype UNSIGNED_26 is UNSIGNED_LONGWORD range 0 .. 2**26-1;
subtype UNSIGNED_27 is UNSIGNED_LONGWORD range 0 .. 2**27-1;
subtype UNSIGNED_28 is UNSIGNED_LONGWORD range 0 .. 2**28-1;
subtype UNSIGNED_29 is UNSIGNED_LONGWORD range 0 .. 2**29-1;
subtype UNSIGNED_30 is UNSIGNED_LONGWORD range 0 .. 2**30-1;
subtype UNSIGNED_31 is UNSIGNED_LONGWORD range 0 .. 2**31-1;

-- Function for obtaining global symbol values

function IMPORT_VALUE (SYMBOL : STRING) return UNSIGNED_LONGWORD;

-- VAX device and process register operations

function READ_REGISTER (SOURCE : UNSIGNED_BYTE)      return UNSIGNED_BYTE;
function READ_REGISTER (SOURCE : UNSIGNED_WORD)      return UNSIGNED_WORD;
function READ_REGISTER (SOURCE : UNSIGNED_LONGWORD)
  return UNSIGNED_LONGWORD;

procedure WRITE_REGISTER (SOURCE : UNSIGNED_BYTE;
  TARGET : out UNSIGNED_BYTE);
procedure WRITE_REGISTER (SOURCE : UNSIGNED_WORD;
  TARGET : out UNSIGNED_WORD);
procedure WRITE_REGISTER (SOURCE : UNSIGNED_LONGWORD;
  TARGET : out UNSIGNED_LONGWORD);

function MFPR (REG_NUMBER : INTEGER) return UNSIGNED_LONGWORD;
Procedure MFPR (REG_NUMBER : INTEGER;
  SOURCE : UNSIGNED_LONGWORD);

-- VAX interlocked-instruction procedures

procedure CLEAR_INTERLOCKED (BIT : in out BOOLEAN;
  OLD_VALUE : out BOOLEAN);
procedure SET_INTERLOCKED (BIT : in out BOOLEAN;
  OLD_VALUE : out BOOLEAN);

type ALIGNED_WORD is
  record
    VALUE : SHORT_INTEGER := 0;
  end record;
for ALIGNED_WORD use
  record
    at mod 2;
  end record;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for VAXELN Ada (Continued).

```
procedure ADD_INTERLOCKED (ADDEND : in      SHOT_INTEGER;
                           AUGEND  : in out ALIGNED_WORD;
                           SIGN    : out    INTEGER);

type INSQ_STATUS is (OK_NOT_FIRST, FAIL_NO_LOCK, OK_FIRST);
type REMQ_STATUS is (OK_NOT_EMPTY, FAIL_NO_LOCK,
                     OK_EMPTY,      FAIL_WAS_EMPTY);

procedure INSQHI (ITEM   : in  ADDRESS;
                  HEADER : in  ADDRESS;
                  STATUS : out INSQ_STATUS);

procedure REMQHI (ITEM   : in  ADDRESS;
                  HEADER : in  ADDRESS;
                  STATUS : out REMQ_STATUS);

procedure INSQTI (ITEM   : in  ADDRESS;
                  HEADER : in  ADDRESS;
                  STATUS : out INSQ_STATUS);

procedure REMQTI (ITEM   : in  ADDRESS;
                  HEADER : in  ADDRESS;
                  STATUS : out REMQ_STATUS);

private

  -- Not shown

end SYSTEM;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for VAXELN Ada

Package STANDARD is

```
type BOOLEAN is (FALSE, TRUE);
```

```
-- The predefined relational operators for this type are as follows:
```

```
-- function "=" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;  
-- function "/=" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;  
-- function "<" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;  
-- function "<=" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;  
-- function ">" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;  
-- function ">=" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
```

```
-- the predefined logical operators and the predefined logical negation  
-- operator are as follows:
```

```
-- function "and" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;  
-- function "or" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;  
-- function "xor" (LEFT, RIGHT : BOOLEAN) return BOOLEAN;
```

```
-- function "not" (RIGHT : BOOLEAN) return BOOLEAN;
```

```
type (universal_integer) is (range unbounded .. unbounded);
```

```
type INTEGER is (range -2_147_483_648 .. 2_147_483_647);
```

```
-- The predefined operators for this type are as follows :
```

```
-- function "=" (LEFT, RIGHT : INTEGER) return BOOLEAN;  
-- function "/=" (LEFT, RIGHT : INTEGER) return BOOLEAN;  
-- function "<" (LEFT, RIGHT : INTEGER) return BOOLEAN;  
-- function "<=" (LEFT, RIGHT : INTEGER) return BOOLEAN;  
-- function ">" (LEFT, RIGHT : INTEGER) return BOOLEAN;  
-- function ">=" (LEFT, RIGHT : INTEGER) return BOOLEAN;  
-- function "+" (RIGHT : INTEGER) return INTEGER;  
-- function "-" (RIGHT : INTEGER) return INTEGER;  
-- function "abs" (RIGHT : INTEGER) return INTEGER;
```

```
-- function "+" (LEFT, RIGHT : INTEGER) return INTEGER;  
-- function "-" (LEFT, RIGHT : INTEGER) return INTEGER;  
-- function "*" (LEFT, RIGHT : INTEGER) return INTEGER;  
-- function "/" (LEFT, RIGHT : INTEGER) return INTEGER;  
-- function "rem" (LEFT, RIGHT : INTEGER) return INTEGER;  
-- function "mod" (LEFT, RIGHT : INTEGER) return INTEGER;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for VAXELN Ada (Continued).

```
-- function "***"    (LEFT : INTEGER;
--                    RIGHT : INTEGER) return INTEGER;

-- An implementation may provide additional predefined integer types.
-- It is recommended that the names of such additional types end
-- with INTEGER as in SHORT_INTEGER or LONG_INTEGER. The specification
-- of each operator for the type universal_integer, or for any
-- additional predefined integer type is obtained by replacing
-- INTEGER by the name of the type in the specification of the
-- corresponding operator of the type INTEGER, except for the right
-- operand of the exponentiating operator.

type SHORT_INTEGER is (range -32_768 .. 32_767);
type SHORT_SHORT_INTEGER is (range -128 .. 127);

type (universal_real) is (range unbounded .. unbounded);

type FLOAT is (digits 6);

-- The predefined operators for this type are as follows :

-- function "="      (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function "/="     (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function "<"      (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function "<="    (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function ">"      (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function ">="    (LEFT, RIGHT : FLOAT) return BOOLEAN;
-- function "+"      (RIGHT : FLOAT) return FLOAT;
-- function "-"      (RIGHT : FLOAT) return FLOAT;
-- function "abs"    (RIGHT : FLOAT) return FLOAT;
-- function "+"      (LEFT, RIGHT : FLOAT) return FLOAT;
-- function "-"      (LEFT, RIGHT : FLOAT) return FLOAT;
-- function "*"      (LEFT, RIGHT : FLOAT) return FLOAT;
-- function "/"      (LEFT, RIGHT : FLOAT) return FLOAT;
-- function "***"    (LEFT : FLOAT; RIGHT : INTEGER) return FLOAT;

-- An implementation may provide additional predefined FLOAT types.
-- It is recommended that the names of such additional types end
-- with FLOAT as in SHORT_FLOAT or LONG_FLOAT. The specification
-- of each operator for the type universal_real, or for any
-- additional predefined floating point type is obtained by replacing
-- FLOAT by the name of the type in the specification of the
-- corresponding operator of the type FLOAT.

type LONG_FLOAT is (digits 15);
type LONG_LONG_FLOAT is (digits 33);
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for VAXELN Ada (Continued).

-- in addition, the following operators are predefined for universal types:

```
-- function "*" (LEFT : {universal_integer};
                RIGHT : {universal_real}) return {universal_real};
-- function "*" (LEFT : {universal_real};
                RIGHT : {universal_integer}) return {universal_real};
-- function "/" (LEFT : {universal_real};
                RIGHT : {universal_real}) return {universal_real};

-- function {universal_fixed} is
    (delta unbound range unbounded .. unbounded);

-- The type universal_fixed is predefined. The only operators declared
-- for this type are:

-- function "*" (LEFT : {any_fixed_point_type};
                RIGHT : {any_fixed_point_type}) return {universal_fixed};
-- function "/" (LEFT : {any_fixed_point_type};
                RIGHT : {any_fixed_point_type}) return {universal_fixed};
```

type CHARACTER is

('nul', ... 'del');

```
-- for CHARACTER use -- 128 ASCII character set without holes
--      (0, 1, 2, 3, 4, 5, ..., 125, 126, 127);
--      for CHARACTER'SIZE use 8;
-- The predefined operators for the type CHARACTER are the same
-- as for any enumeration type.
```

package ASCII is

...

end ASCII;

-- Predefined subtypes:

```
subtype NATURAL is INTEGER range 0 .. INTEGER'LAST;
subtype POSITIVE is INTEGER range 1 .. INTEGER'LAST;
```

-- Predefined string type:

```
type string is array(POSITIVE range <>) of CHARACTER;
pragma PACK(String);
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for VAXELN Ada (Continued).

```
-- function "="      (LEFT, RIGHT : STRING) return BOOLEAN;
-- function "/="     (LEFT, RIGHT : STRING) return BOOLEAN;
-- function "<"      (LEFT, RIGHT : STRING) return BOOLEAN;
-- function "<="     (LEFT, RIGHT : STRING) return BOOLEAN;
-- function ">"      (LEFT, RIGHT : STRING) return BOOLEAN;
-- function ">="     (LEFT, RIGHT : STRING) return BOOLEAN;

-- function "&" (LEFT  : STRING;
                RIGHT : STRING) return STRING;
-- function "&" (LEFT  : CHARACTER;
                RIGHT : STRING) return STRING;
-- function "&" (LEFT  : STRING;
                RIGHT : CHARACTER) return STRING;
-- function "&" (LEFT  : CHARACTER;
                RIGHT : CHARACTER) return STRING;

type DURATION is (delta 1.0e-4 range -131_072.0 .. 131_072.9999);

-- The predefined operators for the type DURATION are the same as for
-- any fixed point type.

-- The predefined exceptions:

CONSTRAINT_ERROR : exception;
NUMERIC_ERROR    : exception;
PROGRAM_ERROR     : exception;
STORAGE_ERROR    : exception;
TASKING_ERROR    : exception;

end STANDARD;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Gould, Inc. Compiler version Apex 2.1	Gould PowerNode Model 9080 (under UTX/32 Version 2.0)	Gould PowerNode Model 6080 (or SelConnection) (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

Individual subprograms may be extracted from packages only.

II. Customization of the Runtime:

- By pragmas
- By compiler switches
- By modifying/replacing the source to selective runtime routines provided by the compiler vendor with the purchase of the compiler (i.e. device drivers, etc.)
- By modifying the source to the entire runtime (after purchasing it)

III. Documentation provided to help user configure runtime:

"Apex" (Gould Ada Compiler) Bare Machine Ada Runtime Library Reference Manual.

IV. Services to customize the runtime:

None.

V. Cost of runtime source code:

Interested users must call the home office to obtain a quote.

VI. Source of Information: Vendor.

PIWG RESULTS

This information was not supplied by the vendor.

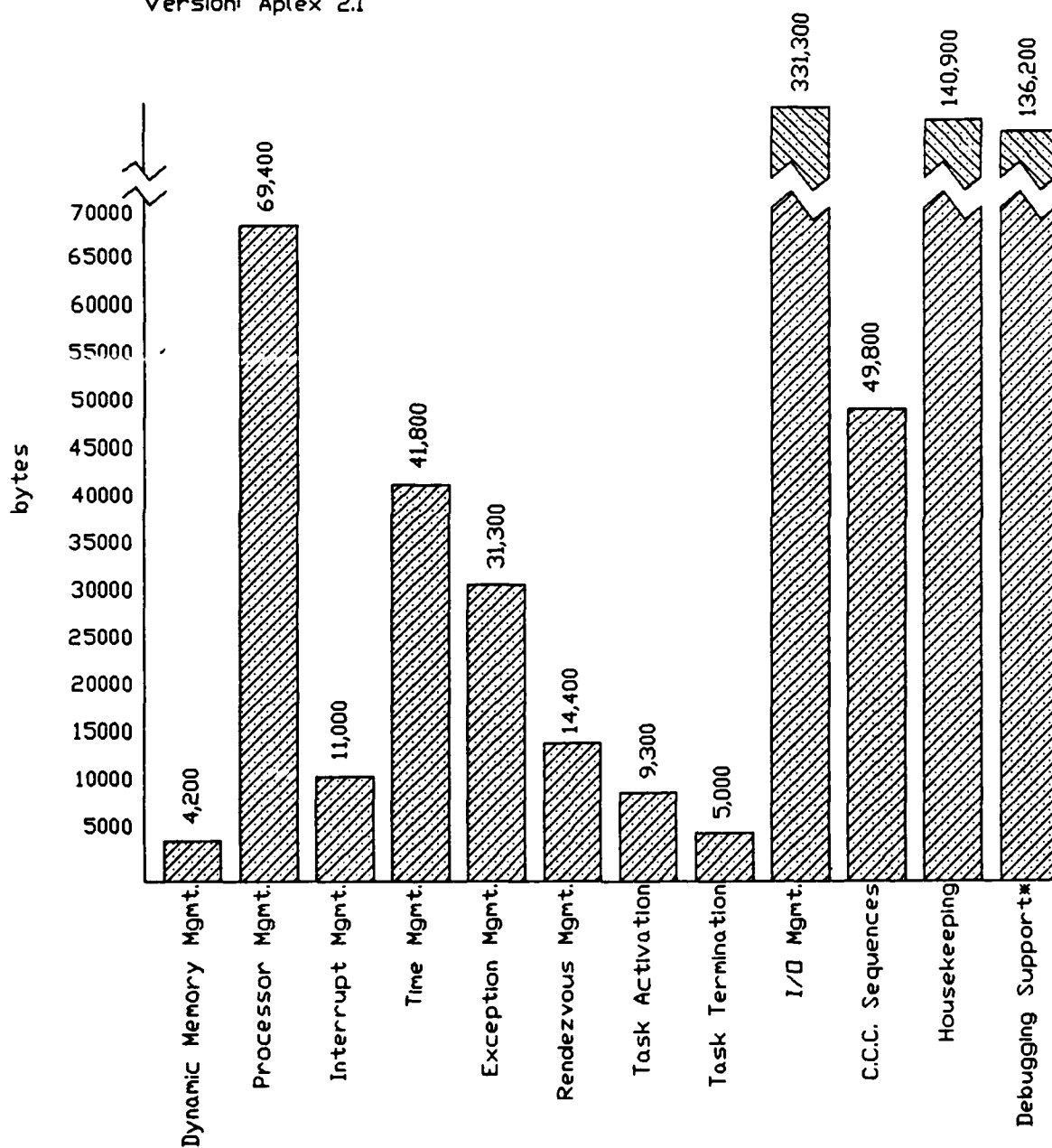
Guidelines to Select, Configure and Use an Ada Runtime Environment

Gould

Host: Gould Powernode

Target: Gould Powernode 6080 (or SelConnection)

Version: Apex 2.1



Sum of ALL components = 844600 bytes

* Component was supplied by vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Intermetrics, Inc. Compiler version 202.08A	VAX-11/785 (under VMS 4.2)	1750A, ECSPO-RAID Simulator CX-04.001 (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Individual subprograms and/or data objects may be extracted from packages only.

II. Customization of the Runtime:

- By pragmas
- By linker switches
- By modifying/replacing the source to selective runtime routines provided by the compiler vendor with the purchase of the compiler (i.e. device drivers, etc.)

III. Documentation provided to help user configure runtime:

- PQ1750A Compilation System User's Manual
- Retargeting Guide
- RTS B5/C5 Specs

IV. Services to customize the runtime:

- Intermetrics provides services to customize the runtime.
- Cost: Will do the work or help customers on a Time and Materials basis.

V. Cost of runtime source code:

- Comes as part of the product.

VI. Source of Information: Vendor input.

PIWG RESULTS

This information was not supplied by the vendor.

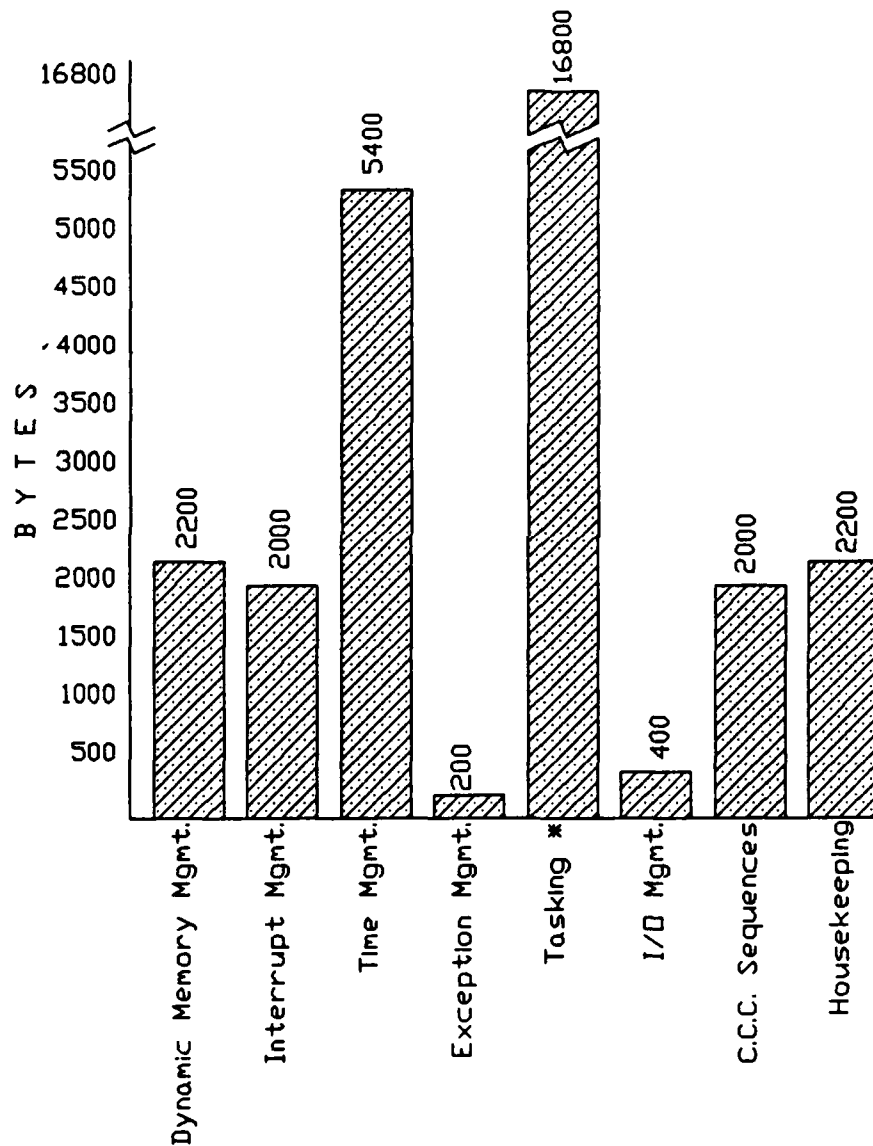
Guidelines to Select, Configure and Use an Ada Runtime Environment

Intermetrics Incorporated

Host : VAX11-785 / VMS 4.5

Target : 1750A ECSPD-RAID Simulator CX-04.001

Version : 202.08A



- Sum of ALL Components = 31,200 bytes

- * Tasking Includes :
1. Rendezvous Management
 2. Task Activation
 3. Task Termination
 4. Processor Management

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for **delay** statements?

A1: 100 microseconds on 1750A using timer-B

Q2: How long, and for what reasons are interrupts disabled?

A2: Less than 100 cycles (e.g., 10 microseconds on a 10MHz chip) for the timer-B interrupt processing.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: Acceptor always runs on caller's stack, thus avoiding many scheduling points.

Q4: What are the restrictions for representation clauses?

A4: There are two restrictions at present time, one will be removed next release:

- 1) Length and Address clauses fully supported in Fall, 1988.
- 2) Subcomponent of a record must fit in a single word.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Run-until-blocked. Priority queue is implemented as a heap. Pre-emption in some cases (semi-custom).

Q6: What are the restrictions on **pragma INLINE**?

A6: Subprogram bodies must be compiled before, and can't be recursive.

Q7: Is code "ROM"able?

A7: Yes (code and read-only data).

Q8: Are machine code inserts supported?

A8: **Pragma INTERFACE** to assembly code.

Q9: What object types are supported by **pragma SHARED**?

A9: **Pragma SHARED** is not supported.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

- Timer Resolution
- Default stack sizes
- Semaphore operations
- Exception trace
- Terminal I/O
- Memory size, number of page-registers

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the MIL_STD-1750A

package SYSTEM is

type ADDRESS is private; -- "=", "/" defined implicitly;
type NAME is (UTS, MVS, CMS, MIL_STD_1750A);

SYSTEM_NAME : constant NAME := MIL_STD_1750A;

STORAGE_UNIT : constant := 16;

MEMORY_SIZE : constant := 2**15;
-- In storage units

-- System-Dependent Named Numbers:

MIN_INT : constant := INTEGER'POS(INTEGER'FIRST);

MAX_INT : constant := INTEGER'POS(INTEGER'LAST);

MAX_DIGITS : constant := 9;

MAX_MANTISSA : constant := 31;

FINE_DELTA : constant := 2.0**(-31);

TICK : constant := 0.0001;

-- Other System-Dependent Declarations

subtype PRIORITY is INTEGER range -127..127;

-- Implementation-dependent additions to package SYSTEM --

NULL_ADDRESS : constant ADDRESS;

-- Same bit pattern as "null" access value

-- This is the value of 'ADDRESS for named numbers.

-- The 'ADDRESS of any object which occupies storage

-- is NOT equal to this value.

ADDRESS_SIZE : constant := 16;

-- Number of bits in ADDRESS objects, = ADDRESS'SIZE, but static.

-- ADDRESS_SEGMENT_SIZE : constant := 2**16;

-- Number of storage units in address segment

type ADDRESS_OFFSET is new INTEGER; -- Used for address arithmetic

type ADDRESS_SEGMENT is new INTEGER; -- Always zero on targets with
-- unsegmented address space.

subtype NORMALIZED_ADDRESS_OFFSET is ADDRESS_OFFSET;

-- Range of address offsets returned by OFFSET_OF

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the MIL_STD-1750A (Continued)

```
function "+"(addr : ADDRESS; offset : ADDRESS_OFFSET) return ADDRESS;
function "+"(offset : ADDRESS_OFFSET; addr : ADDRESS) return ADDRESS;
    -- Provide addition between addresses and
    -- offsets. May cross segment boundaries on targets where
    -- objects may span segments.
    -- On other targets, CONSTRAINT_ERROR will be raised when
    -- OFFSET_OF(addr) + offset not in NORMALIZED_ADDRESS_OFFSET.

function "-"(left, right : ADDRESS) return ADDRESS_OFFSET;
    -- May exceed SEGMENT_SIZE on targets where objects may
    -- span segments.
    -- On other targets, CONSTRAINT_ERROR
    -- will be raised if SEGMENT_OF(left) /= SEGMENT_OF(right).

function "-"(addr : ADDRESS; offset : ADDRESS_OFFSET) return
    ADDRESS;
    -- Provide subtraction of addresses and offsets.
    -- May cross segment boundaries on targets where
    -- objects may span segments.
    -- On other targets, CONSTRAINT_ERROR will be raised when
    -- OFFSET_OF(addr) - offset not in NORMALIZED_ADDRESS_OFFSET.

function OFFSET_OF (addr : ADDRESS) return NORMALIZED_ADDRESS_OFFSET;
    -- Extract offset part of ADDRESS
    -- Always in range 0..seg_size - 1

function SEGMENT_OF (addr : ADDRESS) return ADDRESS_SEGMENT;
    -- Extract segment
    -- part of ADDRESS
    -- (zero on targets with
    -- unsegmented address space)

function MAKE_ADDRESS (offset : ADDRESS_OFFSET;
    segment : ADDRESS_SEGMENT := 0) return ADDRESS;
    -- build address given offset and segment.
    -- Offset may be > seg_size on targets where
    -- objects may span segments, in which case it is equivalent
    -- to "MAKE_ADDRESS(0, segment) + offset".
    -- On other targets, CONSTRAINT_ERROR will be raised when
    -- offset not in NORMALIZED_ADDRESS_OFFSET.

type Supported_Language_Name is ( -- Target dependent
    -- The following are "foreign" languages:

    AIE_ASSEMBLER, -- NOT a "foreign" language - uses AIE RTS
    UNSPECIFIED_LANGUAGE );
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the MIL_STD-1750A (Continued)

-- Most/least accurate built-in integer and float types

subtype LONGEST_INTEGER is STANDARD.INTEGER;

subtype SHORTEST_FLOAT is STANDARD.FLOAT;

private

type ADDRESS is access INTEGER;

-- Note: The designated type here (INTEGER) is irrelevant.

-- ADDRESS is made an access type simply to guarantee it has

-- the same size as access values, which are single addresses.

-- Allocators of type ADDRESS are NOT meaningful.

NULL_ADDRESS : constant ADDRESS := null;

end SYSTEM;

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Rational Compiler version 2.0.122	Rational 1000	1750A, MIL-STD-1750A (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Any part of a library unit being required loads the entire unit.
- Link Time Dead Code Elimination (LTDCE) is currently under development. It is not available in the current release product, but is scheduled to be available in the fourth quarter.

II. Customization of the Runtime:

- By pragmas
- By compiler switches
- By linker switches
- By Modifying-Replacing the source to selective runtime routines provided with the purchase of the compiler (i.e. Device Drivers).
- By modifying the source to the entire runtime (after purchasing it).

The following are excerpts from Rational's "Technical Specification" for the Rational R1000 to MIL-STD-1750A Cross-Development Facility. [20]

Cross-Compiler Performance

The runtime performance is comparable to that of code generated by a mature optimizing JOVIAL compiler. Runtime performance is measured in terms of both size of object code and speed of execution.

Optimizations

- Elimination of common subexpressions: Detects redundant expressions and uses knowledge of the target machine, expression context, loop depth, and expression frequency to determine which of the feasible common subexpression replacements are desirable.
- Code redistribution.
- Strength reduction: Replaces multiplication involving loop counters with appropriate additions.
- Compile-time constant arithmetic and conversions, value folding: Performs arithmetic and logical computations at compile time, and removes dead code.
- Code straightening: Eliminates jumps to jumps.
- Elimination of dead code: Eliminates code that can never be reached by the program.

Guidelines to Select, Configure and Use an Ada Runtime Environment

- Branch optimization.
- Global register assignment: Allocates frequently referenced locals to machine registers.
- Peephole optimization: Includes removing redundant loads, stores, and comparisons and replacing general-case code sequences with shorter or faster special-case idioms.
- Elimination of constraint checks: Eliminates unnecessary constraint checks for efficiency. Consider the following constraint checks:
 - When assigning a scalar value to a variable, check that the value is within the declared (subtype) range.
 - When accessing a component of an array, check that the index is within the declared array index range.
 - When selecting a component of a record controlled by a discriminant, check that the discriminant has the correct value.
 - When dereferencing an access object, check that the access value is non-null.

Although these checks can be very expensive, they are essential if the object code is to be safe; the Suppress pragma is supported, but it is better to retain the checking code where it is necessary. The compiler eliminates checks that it can prove unnecessary. Ada programs provide explicit subtype information that can be used to eliminate many runtime checks. In general, a substantial percentage of the checks can be eliminated by one or more of these techniques:

- Value tracking: The value to be checked is known, and it is known to be valid.
 - Range tracking: The range of the value can be computed, and it is contained within the required range. Note that the ranges of expressions can be synthesized from the ranges of their operands.
 - Equivalence propagation: The range of the value is not known numerically, but it is known algebraically, and the check can be proven satisfied by algebraic identity.
 - Truth propagation: The value is known to be within range because it is a precondition of the code.
- Store suppression: Deletes an assignment to a variable when it is not followed by any subsequent reference to the variable before another assignment or before the variable passes out of scope.

Controlling Optimization

The debugger provides varying degrees of capability depending on optimizations performed by the code generator. The code generator permits the user to specify the degree of optimization on a unit-by-unit basis. The following optimization levels can be specified:

Guidelines to Select, Configure and Use an Ada Runtime Environment

- Level 0: Performs only constant folding and algebraic transformations.
- Level 1: Adds peephole optimization and common subexpression elimination on basic blocks; also optimizes the evaluation ordering of expressions.
- Level 2: Adds everything else except those optimizations expressly called out in higher levels.
- Level 3: Adds strength reduction.
- Level 4: Adds in-line expansion.

Code Generation Strategy

Many decisions about the runtime representation of program entities are critical to the performance of an Ada system. The cross-compiler performs extensive analysis so that common special cases can be implemented efficiently. For example:

- Runtime type descriptors (such as array dope vectors) are created only when they are really needed.
- Record fields are reordered so as to minimize wasted space, satisfy alignment requirements, and remain efficiently addressable.
- The size of a constrained discriminated record object is determined by the sizes of active fields in the object, not by the sizes of the fields in the largest possible object of the unconstrained type.
- Record objects are always allocated contiguously; individual fields are never allocated on the heap. Record assignment and comparison are performed using block operations.
- Record and array parameters are passed by reference; array slices are treated as references rather than copies.
- Local objects of dynamic size are allocated on the stack rather than on the heap.

Runtime Library

The runtime library provides an efficient implementation of Ada language features, including exception handling, tasking support, and storage management. Source code for the runtime library is provided, with rights to an object code sub-license for delivery to third parties. This ensures that development teams can modify the runtime library if performance-critical sections of their applications require it or to interface to a specific kernel.

Exceptions

The exception-handling facilities are designed so that little or no cost is incurred in subprograms that have no exception handler. When an exception is raised, the processing cost depends on whether the exception is propagated out of a rendezvous, the number of reraises, and the complexity of the handlers.

Tasking Model

Tasking is implemented by the Ada runtimes. Entry parameters are passed as if they were subprogram parameters. No copying of parameters or argument lists is performed by the runtime library.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Stack storage for a task is allocated when a task is activated and is never subsequently extended. The amount of space allocated can be controlled by the 'Storage_Size' attribute. The stack is deallocated when the task terminates.

The accuracy of the realtime clock and the delay statement is 5 milliseconds.

Storage Management

Access objects are allocated from an access object storage area for which storage is allocated at the time that the access type declaration is elaborated and deallocated when the access type declaration passes out of scope. This area is extended as needed unless a Storage_Size representation clause has been specified for the type, in which case the size of the collection is fixed.

Unchecked_Deallocation is supported; an efficient algorithm adds the deallocated storage to a list of free cells that are available for subsequent allocations.

Runtime Library Size

The runtime library is packaged to prevent loading unused modules as well as unnecessary control sections. Storage management occupies less than 1.0 K words, tasking services about 6.0 K words, and exception handling less than 0.5 K words. Although the complete library would occupy 7.5 K words, many applications require only a fraction of the complete capability and thus use much less space.

Real-time Kernel

The real-time kernel provides the support necessary to run programs on a bare 1750A with or without extended memory. In addition to process initiation and scheduling, it provides a number of services for use by application programs. These services are described in the "Kernel Services" subsection below.

Two kernels are provided: a single-process kernel and a multiprogramming kernel. The single-process kernel provides a small, efficient set of services on top of which a single program can be run; the linker loads only those parts that are required by the application. The multiprogramming kernel provides an environment on top of which several programs can be run; a single copy of this configurable kernel is shared by all programs.

Kernel Interface

The majority of kernel code is written in Ada; performance-critical regions are written in assembly language. An Ada package is provided to give a user program access to all services provided by the kernel. Subprograms in this Ada package are responsible for extensive validity checks on parameters. After validity checks are performed, a lower layer, containing the actual kernel service is called. This structure permits bypassing the validity-checking layer and directly calling the lower-level service when increased performance is essential.

Source code is provided for both the Ada and assembly-language portions of each kernel, with rights to an object code sub-license for delivery to third parties.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Kernel Services

Clock Services

The kernel maintains a real-time clock. The real-time clock is initialized at system startup and can be read or modified by application programs. The kernel utilizes the two 1750A hardware timers; one of them can be optionally allocated to an application program.

Dynamic Memory-Management Services

In addition to Ada's memory-management facilities, other programmatic services are provided. Applications can:

- Request the assignment of additional memory in 4 K word increments.
- Release memory in 4 K word increments.
- Define an already allocated page as a shared one and associate a name with it.
- Gain access to a shared page by name.
- Relinquish a previous association with a shared page.
- Utilize the block protect facilities if they are available.

Interlock Services

Interlocks are low-level primitives that can be used to implement a variety of synchronization, reservation, or communication algorithms among programs. Examples are monitors, critical regions, and semaphores.

Interrupt Handling

A number of methods are provided for servicing interrupts:

- Using address clauses associates a task entry with an interrupt.
- Invoking a system service identifies an Ada subprogram (or assembly-language subroutine) as a call target when a specified interrupt occurs. Restrictions apply when the routine is an Ada subprogram.
- Invoking a system service puts a program into a wait state until the specified interrupt occurs.

Input/Output Services and Device Drivers

This group of functions addresses I/O beyond the standard Ada facilities:

- Frame input and output either read or write one frame from a TTY-like device. This device can be connected through either the console interface or an RS232 interface. A frame is defined to be any number of bits (up to 16) that can be transferred in a single I/O operation. The exact number is user definable.
- A user program can be suspended until a frame arrives from a specified channel.
- In addition to the language-defined I/O packages, a file I/O package is

Guidelines to Select, Configure and Use an Ada Runtime Environment

provided that can be adapted to most mass storage or data transfer devices. The primitive operations for reading a block, writing a block, and checking on the I/O status must be filled in by the user for the specific device.

- Kernel services are available to enable an application program to control devices. This includes services to handle interrupts, access device registers, wait on interrupts, and execute privileged instructions. The application can also:

- Clear or set a named bit in a discrete register.
- Read a named bit from a discrete register.
- Read or write a full word of a discrete register.
- Check a named discrete bit and, if its status is not as expected, suspend the program until the bit changes its status.

- Facilities are also provided for a user to install a device driver in the kernel.

Real-time Kernel Size

The single-process kernel varies in size from 0.5 to 1.7 K words, depending on which parts are required by the application and loaded by the linker. The multiprogramming kernel is configurable to suit specific application requirements; it occupies 24 K words if all functions are used. A single copy of this kernel is shared by all programs.

Debugger

The host/target debugger allows users to debug applications executing on 1750A hardware or a simulator. Several modes are supported:

- Debugging on the R1000 using an instruction-level simulator.
- Debugging on the target hardware with a small resident debug monitor.
- Debugging on the target hardware using an in-circuit emulator.

These modes will be discussed in turn, because they represent in somewhat chronological order the way in which an application might be debugged.

Simulator

An instruction-level simulator executes the MIL-STD-1750A instruction set. From the information provided to it by the linker, the behavior or a particular implementation of MIL-STD-1750A can be simulated.

In addition to the features described below in the "Debugging Capabilities" section, these features are supported by the simulator:

Guidelines to Select, Configure and Use an Ada Runtime Environment

- The simulator is configurable to simulate the particular implementation of the 1750A required. The simulator can also be configured with instruction timing data.
- The memory in the simulator is loaded with the load module output by the linker. The simulator can also write the contents of its memories (that is, the physical memory, the page registers, the memory-protect RAM, and the startup ROM) at another address. This state can be compared to the contents of a range of memory addresses and the variances listed.
- The simulator can write its entire current state into a file for later restarting. From this file, the simulator can regain all attributes of its condition at the point where the state was saved and carry on.
- Individual registers, memory locations, and the real-time clock can be monitored and their values output when certain values are obtained or the value changes. The real-time clock can be used as a breakpoint.

Host/Target Communications

To be able to make use of host/target debugging facilities offered by the Rational Environment, the target computer must be made accessible to the host R1000. Download capability for particular 1750A implementations or configurations is provided as necessary. Downloading can be done to 1750A hardware, to an in-circuit emulator, or to an industry-standard PROM programmer. The standard supported protocols are Ethernet with TCP/IP or RS232 operating up to 19,200 baud. Customers with other communications requirements should contact a Rational representative.

It may be necessary to install a small debug monitor on the target computer, depending on the degree of hardware support.

User Debugging Model

The host/target debugger allows users to debug programs running on the 1750A from within the Rational Environment. All the facilities that are available when debugging a program executing on an R1000 carry over to host/target debugging on the 1750A. Because host/target debugging is integrated into the Rational Environment, the interface is the same as when debugging a program executing on the R1000. The user benefits from a multiwindow display containing debugger information, source code corresponding to program location, and the output of the program itself. In this situation, input and output from the program running on the 1750A are redirected to the Environment. When this redirection is inconvenient - when a different type of display is required, for example - a terminal can be connected as the 1750A console and used as the program's I/O device. In some applications, there will be no console.

The user must compile the Ada unit with the debug switch set in order to do source-level host/target debugging. Code generated with debugging enabled can be run without the debugger; the generated code has the same performance characteristics.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Executing and Debugging a Program on the Target 1750A

The debugger has two components, an R1000 part and a part that resides on the target 1750A. Application program execution can be initiated from the Rational Environment or directly on the target 1750A. In either case, the application running on the 1750A can be started with or without debugger control. When initiated from the R1000 under debugger control, the application waits for a command before elaboration. When initiated on the 1750A under debugger control, it also stops before elaboration and awaits a command. The R1000 part of the debugger can then be started and attached to the 1750A-resident portion. Debugging then proceeds as in the first case.

Debugging After Program Initiation

Even if the application was initiated without debugging, it is possible to invoke the R1000 host debugger and have it subsequently control the application executing on the 1750A.

Debugging a Memory Image

Additional debugger facilities permit high-level interrogation of the memory image of a program that has terminated abnormally. A typical use would be to examine the program state after an unhandled exception has caused termination of the program.

Multiprogramming Debugging

Multiple-program debugging is supported. Two or more separate debugger jobs can be run at the same time, each controlling a different 1750A process and each having its own window on the R1000 terminal. The programs themselves can be run on the same or different target 1750As.

Debugging Capabilities

The 1750A host/target debugger provides the following capabilities:

- Display of task call stacks.
- Display of task state.
- Task control. The debugger provides two task control models:
 - Separate control: A single task can stop at a breakpoint or exception event while others continue to run.
 - Synchronous control: A breakpoint or exception event causes all the tasks in the program to stop.

Guidelines to Select, Configure and Use an Ada Runtime Environment

- Display and modification of program data values.
- Display of processor registers and memory.
- Display of location in source program code.
- Ada and machine-level breakpointing.
- Ada and machine-level stepping.
- Controlling the catching and propagation of exceptions.
- Display of program history.
- Performance monitoring.
- Tracing.
- Disassembly.

Levels of Debugging

The debugger provides varying degrees of capability depending on the optimizations performed by the code generator; the precision with which the debugger can locate objects decreases as code is more highly optimized. To ensure adequate flexibility in the development and debugging of large applications, the optimization level can be specified on a unit-by-unit basis, as described in the "Optimizations" subsection.

The degree of optimization can also be controlled by specifying one of three levels of debugging:

- None: No debug tables are created and no attempt is made to limit optimizations.
- Partial: The ability to display the value of formal parameters is preserved. There are no restriction on optimizations; the ability to display objects and determine program locations is reduced in an amount determined by the optimization setting.
- Full: Optimizations that prevent accurate evaluation of objects or determination of source location are inhibited. Automatic in-lining is disabled, and there is no code motion across statement boundaries and no reduction of object lifetimes.

Support for In-Circuit Emulators

The host/target debugger also supports debugging through a Tektronix 8540 in-circuit emulator. The 1750A is controlled by the emulator, and R10 host debugger commands are translated for the 8540, which in turn passes results back to the R1000. Thus the interaction can proceed using the R1000 high-level debugging model, or it can be treated at a lower level by using the emulator locally. In both cases, the emulator enables the developer to run the 1750A application at speed on the processor.

The host/target debugging design allows some flexibility in the choice of in-circuit emulators. Minimal changes to the software allow support for other industry-standard emulators. Customers with other models should contact a Rational representative for more details.

Other Components

Guidelines to Select, Configure and Use an Ada Runtime Environment

To facilitate the production of real-time applications, an assembler and a linker are provided. These facilities enable developers to construct mixed-language applications.

Assembler

The assembler provides programmers with a powerful set of pseudo-operations, including a macro capability. MIL-STD-1750A mnemonics are supported.

The macro facility is integrated with the assembler itself as opposed to a preprocessing facility. It allows the passing of arguments and the definition of default argument values. A submacro directive is also available for defining a local macro for use within a macro body. The assembler also offers a pseudo-operation set providing programmers with a powerful capability to control the assembly process. Assembler directives include:

- Block conditional assembly
- Forward or backward branching to a label at assembly time
- Looping
- Unconditional exit from a loop
- Support for base registers
- Control section definition
- User-invoked origin
- Listing of control directives

Linker

The linker collects object module outputs into a load module potentially covering the entire physical address space, including the page registers, the memory-protect RAM, and the initial address state. The problem of switching address states is handled automatically by the linker through the insertion of transit routines.

Directives permit the inclusion of entire object files, specified modules, or all modules except those specified. Any object file can be searched as a library to satisfy external references. The target memory configuration can be described to the linker so that allocation is compatible with the memory size, number of page registers, and reserved areas of the target.

The extended-memory linker generates an alphabetical symbol list, a symbol cross-reference, and an allocation map. The map shows the instruction and operand external symbols for each module, the date/time of translation of each module, and the name/version of its translator (Ada, JOVIAL, or assembler). A symbol table file is produced.

The linker has the capability of collecting several modules and producing either an absolute load image or a new relocatable module with user-specified entry points and external references exposed from the contained modules. The linker optionally produces a relocatable module that contains only the entry points resulting from a link (either a new relocatable or load image link). The linker also optionally produces a map of the storage allocated by cluster and control section; each is described with a starting address, an ending address, and a length.

Guidelines to Select, Configure and Use an Ada Runtime Environment

The cross-compiler translates relocatable modules for packages such that each top-level subprogram within the package is contained in a separate control section. This permits selective loading only if the subprogram is required. The compiler distinguishes control sections as containing instructions, data, or literals. The compiler supports control sections up to 64 K words each and supports calls, using the above-mentioned transit routines provided by the linker, to control sections from other compilation units that reside in an address state other than that containing the current module.

The linker supports both memory-resident and auxiliary device-resident overlays. It supports multiple control section modules, each of which can have independent attributes, entry points, or external references.

By default, control sections are clustered by attribute, but control sections can be combined by name, module, or control section attribute. clusters, control sections, and modules can be placed at a user-specified location. The linker permits specification of clusters that are to be shared across address states.

Other features include:

- Extended-memory links (greater than 64 K words) and out-of-state references for code are supported.
- Multiple address state programs, multiple programs per address state, and multiple program in multiple address states are supported.
- A limiting address can be defined; exceeding this address causes a warning diagnostic to be issued at link time.
- All locations, values, and space sizes can be specified by relocatable or absolute expressions.
- Memory protection is supported. Protection attributes can be defined by control section and cluster.
- The linker optionally generates checksums/CRC values to permit load validation, periodic memory-destruction tests, and swap-out elimination.

Reusable Software Components

Rational provides a library of packages that reduce implementation, test, and debugging time by providing reusable parts when working on the R1000. As part of the 1750A Cross-Development Facility, a source license is provided so that these components can be incorporated in the applications running on a 1750A that are developed on the R1000. These packages and procedures include:

- Bounded_String
- Concurrent_Map_Generic
- List_Generic
- Map_Generic
- Queue_Generic
- Set_Generic
- Stack_Generic
- String_Map_Generic
- String_Table
- String_Uilities

Guidelines to Select, Configure and Use an Ada Runtime Environment

- Table_Formatter
- Table_Sort_Generic
- Time_Utilities
- Unbounded_String

Relation to Other Products

The 1750A can be connected to the R1000 using an RS232 port without obtaining other products. If an Ethernet connection is desired, Rational Networking is available.

For projects requiring code delivery with some other compiler, the 1750A Cross-Development Facility can be used during the development phase for efficient host/target debugging and rapid turnaround. Actual code delivery can be done with the Rational Target Build Utility, which enables the downloading of Ada source to another system for compilation. The Target Build Utility provides a complete change-tracking history on the R1000 so that the minimum number of units is downloaded and recompiled. In addition, a compilation and link script is downloaded for batch submission on the other system.

III. Documentation provided to help user configure runtime:

- Rational MIL-STD-1750A Cross_development Facility Manual

IV. Services to customize the runtime:

Rational offers an implementation program with each CDF. The plan typically includes installation, training, and site specific customization of the CDF. The plan can be expanded to include runtime customization. The cost of the implementation plan for the MIL-STD-1750A CDF for a R1000 model 20 is \$15,000.

V. Cost of runtime source code:

- The runtime source code is provided with the CDF product. The cost of the MIL-STD-1750A CDF for a R1000 model 20 is \$49,000.

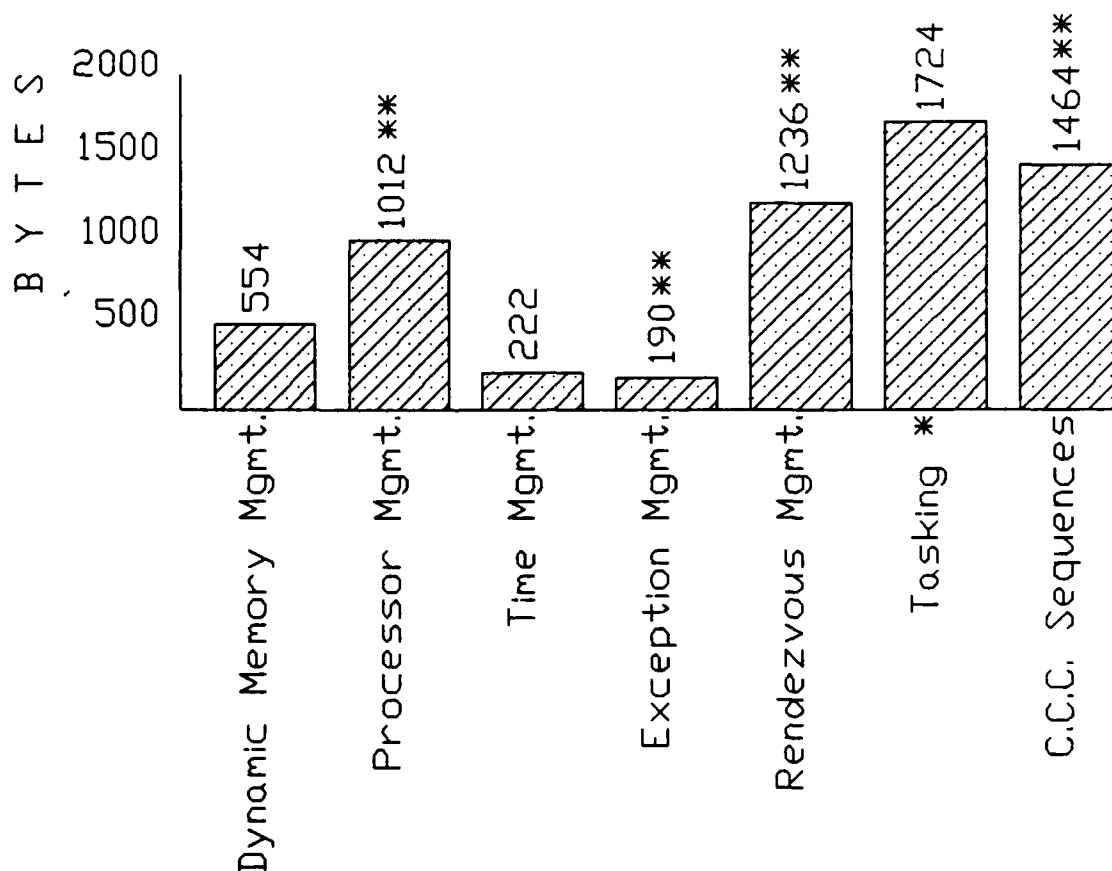
VI. Source of Information: Vendor Input

Rational

Host : Rational R1000

Target : MIL-STD-1750A

Version : 2.0.122



- Sum of ALL Components = 6,402 bytes**

* Includes task creation, activation,
and termination

** See next page.

Note: the actual granularity is much finer
than represented here.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Vendor Supplied Component Description

These components of the runtime for the MIL-STD-1750A have been broken down into further storage requirements for code, data, and constant.

Processor Management	614 bytes code 162 bytes constant 236 bytes data
Exception Management	98 bytes code 92 bytes constant
Rendezvous Management	1234 bytes code 2 bytes data
Commonly Called Code Sequences	1376 bytes code 4 bytes constant 84 bytes data
The maximum RTE including data :	5822 bytes code 258 bytes constant 322 bytes data

All sizes are in bytes. The actual granularity is much finer than represented here.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: 100 microseconds

Q2: How long, and for what reasons are interrupts disabled?

A2: Interrupts are disabled in time critical sections of the runtime. This is an important and complicated area and Rational would welcome the opportunity to discuss your needs in this area. For additional information please contact the Rational sales representative for your area.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: Rational make optimizations which reduce the number of context switches required to perform a rendezvous in which the acceptor's accept statement or accept alternative (in the case of a select wait) contains no associated statements. For example:

```
begin
  accept e;
end;
or
begin
  accept e(P1 : T1; P2 : T2; ...);
end;
or
select
  accept e;
or
  accept e1(P1 : T1; P2 : T2; ...);
.
.
.
end select.
```

Note that depending on the path the selective wait takes, it is the form of that arm which is being taken which determines if this optimization can be performed.

Q4: What are the restrictions for representation clauses?

A4: The MIL-STD-1750A cross-compiler supports the following representation clauses (the following are excerpts from Appendix F, Rational MIL-STD-1750A [23]):

(F.1.5.) Representation Clauses

- Length clauses:

for Access_Type'Storage_Size use X;

If X is static and equal to zero, no collection is allocated. Any attempt to evaluate an allocator will raise the predefined Storage_Error exception. (Other values of X, which need not be static, are honored.)

Guidelines to Select, Configure and Use an Ada Runtime Environment

for Discrete_Type'Size use X;

for Task_Object'Storage_Size use X;

for Task_Type'Storage_Size use X;

for Fixed_Type'Small use X;

- Record representation clauses: The compiler supports both full and partial representation clauses for both discriminated and undiscriminated records.

- Enumeration representation clauses.

(F.1.6.) Restrictions on Array and Record packing and Record Representation Clauses

- Arrays: Packed arrays of discretely (Integer and Enumeration types, including Booleans) are supported. Components of packed arrays occupy the minimum possible number of bits, which may range from 1 through 16.

- Records: A record field can consist of any number of bits between 1 and 16, inclusive; otherwise, it must be an integral number of words.

- Change of representation: Change of representation is supported wherever it is implied by support for representation specifications. In particular, implicit or explicit type conversions between array types or record types may cause packing or unpacking to occur; conversions between related enumeration types with different representations may result in table lookup operations.

The following example shows support for a change of representation of an array:

```
type Arr is array (1..10) of Boolean;  
type Brr is new Arr;  
pragma Pack (Brr)
```

```
X : Arr := (1..10 => false); Y : Brr := Brr (X);
```

Change of representation occurs in the type conversion to Brr.

(F.1.7.) Names Denoting Implementation_Dependent Components

- There are no user-visible implementation names.

(F.1.8.) Interpretation of Expressions That Appear in Address Clauses

- Address clauses are not supported at this time.

(F.1.9.) Unchecked Conversion

- The target type of an unchecked conversion cannot be unconstrained array type or an unconstrained discriminated type.

Guidelines to Select, Configure and Use an Ada Runtime Environment

(F.1.10.) Machine Code

- Machine-code insertions are not supported at this time.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Preemptive scheduling.

Q6: What are the restrictions on `pragma INLINE`?

A6: Subprograms that require elaboration checks will not be inlined.

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: Machine code insertion is not supported at this time.

Q9: What object types are supported by `pragma SHARED`?

A9: `Pragma SHARED` is supported for 16 bit discrete and fixed point types and access types.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

Default stack sizes:	Yes
Semaphore operations:	Yes
Exception trace:	Under control of the debugger
Fast interrupt entry:	Yes
Terminal I/O:	Output only

Additional items:

- Heap size.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the MIL-STD-1750A

package SYSTEM is

type Name is (MIL-STD-1750A);
System_Name : constant Name := MIL-STD-1750A;

Storage_Unit : constant := 16;
Memory_Size : constant := 2 ** 16;

Min_Int : constant := -(2 ** 15);
Max_Int : constant := +(2 ** 15) - 1;

Max_Digits : constant := 9;
Max_Mantissa : constant := 31;
Fine_Delta : constant := 2.0 ** (-31);
Tick : constant := 1.0E-04;

subtype Priority is Integer range 1 .. 254;

type Address is private;

Address_Zero : constant Address;

function "+" (Left : Address; Right : Integer) return Address;
function "+" (Left : Integer; Right : Address) return Address;
function "-" (Left : Address; Right : Address) return Integer;
function "-" (Left : Address; Right : Integer) return Address;

function "<" (Left, Right : Address) return Boolean;
function "<=" (Left, Right : Address) return Boolean;
function ">" (Left, Right : Address) return Boolean;
function "<=" (Left, Right : Address) return Boolean;

function To_Address (X : Integer) return Address;
function To_Integer (X : Address) return Integer;

private

...

end System;

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for the MIL-STD-1750A

package STANDARD is

```
type *Universal_Integer* is [universal_integer] ...
type *Universal_Real* is [universal_real] ...
type *Universal_Fixed* is [universal_fixed] ...
type Boolean is (False, True);
type Integer is range -32_768 .. 32_767;
type Float is digits 6 range -1.70141183460469E+38 .. 1.70141163178060E
type Long_Float is digits 9 -1.70141183460469E+38 .. 1.70141183460160E+
    range
type Duration is delta 6.103515625000000E-05
    range -1.310720000000000E+05 .. 1.3107999938965E+05;
subtype Natural is Integer range 0 .. 32_767;
subtype Positive is Integer range 1 .. 32_767;

type String is array (Positive range <>) of Character;
Pragma Pack (String);
Package Ascii is
...
end Ascii;

Constraint_Error : exception;
Numeric_Error : exception;
Storage_Error : exception;
Tasking_Error : exception;
Program_Error : exception;

type Character is ... ;

end Standard;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Rational Compiler version 2.0.30	Rational 1000	68020, Motorola 68020 (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Any part of a library unit being required loads the entire unit.
- Link Time Dead Code Elimination (LTDCE) is currently under development. It is not available in the current release product, but is scheduled to be available in the fourth quarter (88).

II. Customization of the Runtime:

- By pragmas
- By compiler switches
- By linker switches
- By Modifying-Replacing the source to selective runtime routines provided with the purchase of the compiler (i.e. Device Drivers).
- By modifying the source to the entire runtime (after purchasing it).

The following excerpts are from Rational's "Technical Specification" for the Rational R1000 to M68000 Family Cross-Development Facility. [19]

Cross-Compiler Performance

The runtime performance is comparable to that of code generated by a mature optimizing FORTRAN compiler. Runtime performance is measured in terms of both size of object code and speed of execution.

Optimizations

1. Machine-independent optimizations: The cross-compiler performs these traditional machine-independent code transformations:

- Values and variables: These transformations affect the handling of values and variables:

Guidelines to Select, Configure and Use an Ada Runtime Environment

- Context determination: Determines context-dependent information that is used to guide optimization policies in later optimization phases.
 - Lifetime analysis: Creates a conflict graph to identify which variables must exist simultaneously.
 - Constant folding: Performs arithmetic and logical computations at compile time and removes dead code.
 - Algebraic transformations: Simplifies certain algebraic or logical expressions and transforms expressions into standard formats for more efficient processing in later optimization phases.
 - Value tracking: Substitutes a value for a reference to a variable where the variable is known to have a specific value. This value can then participate in constant folding.
 - Elimination of assignment of unreferenced values: Deletes an assignment to a variable when it is not followed by any subsequent reference to the variable before another assignment or before the variable passes out of scope.
 - Elimination of dead variables: Does not allocate storage for unused variables. The cross-compiler recognizes when two variables can share the same storage (because they are never simultaneously active), and it recognizes when a variable is temporarily dead.
- Code motion: These transformations move code, a process that is more difficult in Ada than in other languages because of the rules governing the interaction between code motion and exceptions; in general, code motion can be done only in conjunction with range tracking.
- Flow and call graph construction: Constructs a flow graph for each subprogram and a call graph for the entire compilation unit.
 - Elimination of common subexpressions: Detects redundant expressions and uses knowledge of the target machine, expression context, loop depth, and expression frequency to determine which of the feasible common subexpression replacements are desirable.
 - Cross-jumping: Identifies identical code sequences and trades off the use of a jump for a smaller sequence of instructions.
 - Elimination of dead code: Eliminates code that can never be reached by the program.
 - Loop strength reduction: Replaces multiplication involving loop counters with appropriate additions.
 - Tail recursion elimination: Replaces tail recursion with the appropriate loop.
2. Elimination of constraint checks: Efficiency dictates the elimination of unnecessary constraint checks. consider the following constraint checks:
- When assigning a scalar value to a variable, check that the value is within the declared (subtype) range.
 - When accessing a component of an array, check that the index is within the declared array index range.
 - When selecting a component of a record controlled by a discriminant, check that the discriminant has the correct value.
 - When dereferencing an access object, check that the access value is nonnull.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Although these checks can be very expensive, they are essential if the object code is to be safe; the Suppress pragma is supported, but it is better to retain the checking code where it is necessary. The compiler eliminates checks that it can prove unnecessary. Ada programs provide explicit subtype information that can be used to eliminate many runtime checks. In general, a substantial percentage of the checks can be eliminated by one or more of these techniques:

- Value tracking: The value to be checked is known, and it is known to be valid.
- Range tracking: The range of the value can be computed, and it is contained within the required range. Note that the ranges of expressions can be synthesized from the ranges of their operands.
- Equivalence propagation: The range of the value is not known numerically, but it is known algebraically, and the check can be proven satisfied by algebraic identity.
- Truth propagation: The value is known to be within range because it is a precondition of the code.

3. 680x0-Specific Optimizations: The cross-compiler performs these additional machine-specific optimizations where appropriate:

- Mapping of local variables onto registers: Allocates frequently referenced locals to machine registers.
- Register targeting: Organizes the intermediate results in a way that will cause the final result to be computed directly in the appropriate register when the result of a computation must be in a certain register (for example, a parameter or result register).
- Determination of evaluation order: Minimizes the temporary register requirements of expressions.
- Short-circuit evaluation: Minimizes the number of tests in compound Boolean expressions without side effects.
- Peephole optimizations: Include removing redundant loads, stores, and comparisons; replacing general-case code sequences with shorter or faster special-case idioms; and eliminating jumps to jumps.
- Elimination of entry/exit protocol for simple procedures: Simplifies in the following ways: if a procedure does not declare dynamic locals, no stack frame pointer is needed; if it declares no locals, the stack need not be moved; if it contains no inner calls, the static link can be left in a register.
- Automatic in-lining of static subprogram calls: Expands in-line (independent of the Inline pragma) subprogram calls within a compilation unit where time/space tradeoffs warrant.

Controlling Optimization

The debugger provides varying degrees of capability depending on optimizations performed by the code generator. The code generator permits the user to specify the degree of optimization on a unit-by-unit basis. The following optimization levels can be specified:

Guidelines to Select, Configure and Use an Ada Runtime Environment

- Level 0: Performs only constant folding and algebraic transformations.
- Level 1: Adds peephole optimization and common subexpression elimination within basic blocks; also optimizes the evaluation ordering of expressions.
- Level 2: Adds everything else except those optimizations expressly called out in higher levels.
- Level 3: Adds strength reduction and tail recursion removal.
- Level 4: Adds in-line expansion.

Code Generation Strategy

Many decisions about the runtime representation of program entities are critical to the performance of an Ada system. The cross-compiler performs extensive analysis so that common special cases can be implemented efficiently. For example:

- Runtime type descriptors (such as array dope vectors) are created only when they are really needed.
- Record fields are reordered so as to minimize wasted space, satisfy alignment requirements, and remain efficiently addressable.
- The size of a constrained discriminated record object is determined by the sizes of active fields in the object, not by the sizes of the fields in the largest possible object of the unconstrained type.
- Record objects are always allocated contiguously; individual fields are never allocated on the heap. Record assignment and comparison are performed using block operations.
- Record and array parameters are passed by reference; array slices are treated as references rather than copies.
- Local objects of dynamic size are allocated on the stack rather than on the heap.

Runtime Library

The runtime library provides an efficient implementation of Ada language features, including exception handling, tasking support, and storage management. Source code for the runtime library is provided, with rights to an object code sub-license for delivery to third parties. This ensures that development teams can modify the runtime library if performance-critical sections of their applications require it or to interface to a specific kernel.

Exceptions

The exception-handling facilities are designed so that little or no cost is incurred in subprograms that have no exception handler. When an exception is raised, the processing cost depends on whether the exception is propagated out of a rendezvous, the number of reraises, and the complexity of the handlers.

Tasking Model

Tasking is implemented by the Ada runtimes. Entry parameters are passed as if they were subprogram parameters. No copying of parameters or argument lists is performed by the runtime library.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Stack storage for a task is allocated when a task is activated and is never subsequently extended. The amount of space allocated can be controlled by the 'Storage_Size' attribute. The stack is deallocated when the task terminates.

Storage Management

Access objects are allocated from an access object storage area for which storage is allocated at the time that the access type declaration is elaborated and deallocated when the access type declaration passes out of scope. This area is extended as needed unless a Storage_Size representation clause has been specified for the type, in which case the size of the collection is fixed.

Unchecked_Deallocation is supported; an efficient algorithm adds the deallocated storage to a list of free cells that are available for subsequent allocations.

Real-time Kernel

Rational supplies a kernel interface - an Ada specification - providing all facilities required by the runtime library. Many customers already have hardware configurations and kernel operating systems in place, either developed as proprietary products or obtained from third-party vendors. To develop applications for a specific kernel with the M68000 Family Cross-Development Facility, the user provides a body for the kernel services specification. This body implements the semantics and issues calls appropriate to the specific user's kernel.

Because source code for the runtime library is provided, customization of the runtime library for a specific kernel is also possible. Rational provides implementation and consulting assistance in tailoring the Cross-Development Facility to specific kernels.

Debugger

The host/target debugger allows users to debug applications executing on 680x0 hardware. Debugging can be done on "bare" target hardware with a small resident debug monitor, or an in-circuit emulator can be used.

Host/Target Communications

To be able to make use of host/target debugging facilities offered by the Rational Environment, the target computer must be made accessible to the host R1000. Download capability for particular 680X0 implementations or configurations is provided as necessary. Downloading can be done to 680X0 hardware, to an in-circuit emulator, or to an industry-standard PROM programmer. The standard supported protocols are Ethernet with TCP/IP or RS232 operating up to 19,200 baud. Customers with other communications requirements should contact a Rational representative.

It may be necessary to install a small debug monitor on the target computer, depending on the degree of hardware support.

User Debugging Model

Guidelines to Select, Configure and Use an Ada Runtime Environment

The host/target debugger allows users to debug programs running on the 680X0 from within the Rational Environment. All the facilities that are available when debugging a program executing on an R1000 carry over to host/target debugging on the 680X0. Because host/target debugging is integrated into the Rational Environment, the interface is the same as when debugging a program executing on the R1000. The user benefits from a multi-window display containing debugger information, source code corresponding to program location, and the output of the program itself. In this situation, input and output from the program running on the 680X0 are redirected to the Environment. When this redirection is inconvenient - when a different type of display is required, for example - a terminal can be connected as the 680X0 and used as the program's I/O device.

The user must compile the Ada unit with the debug switch set in order to do source-level host/target debugging. Code generated with debugging enabled can be run without the debugger; the generated code has the same performance characteristics.

Executing and Debugging a Program on the Target 680X0

The debugger has two components, an R1000 part and a part that resides on the target 680X0. Application program execution can be initiated from the Rational Environment or directly on the target 680X0. In either case, the application running on the 680X0 can be started with or without debugger control. When initiated from the R1000 under debugger control, the application waits for a command before elaboration. When initiated on the 680X0 under debugger control, it also stops before elaboration and awaits a command. The R1000 part of the debugger can then be started and attached to the 680X0-resident portion. Debugging then proceeds as in the first case.

Debugging After Program Initiation

Even if the application was initiated without debugging, it is possible to invoke the R1000 host debugger and have it subsequently control the application executing on the 680X0.

Debugging a Memory Image

Additional debugger facilities permit high-level interrogation of the memory image of a program that has terminated abnormally. A typical use would be to examine the program state after an unhandled exception has caused termination of the program.

Multiprogramming Debugging

Multiple-program debugging is supported. Two or more separate debugger jobs can be run at the same time, each controlling a different 680X0 process and each having its own window on the R1000 terminal. The programs themselves can be run on the same or different target 680X0s.

Debugging Capabilities

Guidelines to Select, Configure and Use an Ada Runtime Environment

The 680X0 host/target debugger provides the following capabilities:

- Display of task call stacks.
- Display of task state.
- Task control. The debugger provides two task control models:
 - Separate control: A single task can stop at a breakpoint or exception event while others continue to run.
 - Synchronous control: A breakpoint or exception event causes all the tasks in the program to stop.
- Display and modification of program data values.
- Display of processor registers and memory.
- Display of location in source program code.
- Ada and machine-level breakpointing.
- Ada and machine-level stepping.
- Controlling the catching and propagation of exceptions.
- Display of program history.
- Performance monitoring.
- Tracing.
- Disassembly.

Levels of Debugging

The debugger provides varying degrees of capability depending on the optimizations performed by the code generator; the precision with which the debugger can locate objects decreases as code is more highly optimized. To ensure adequate flexibility in the development and debugging of large applications, the optimization level can be specified on a unit-by-unit basis, as described in the "Optimizations" subsection.

The degree of optimization can also be controlled by specifying one of three levels of debugging:

- None: No debug tables are created and no attempt is made to limit optimizations.
- Partial: The ability to display the value of formal parameters is preserved. There are no restriction on optimizations; the ability to display objects and determine program locations is reduced in an amount determined by the optimization setting.
- Full: Optimizations that prevent accurate evaluation of objects or determination of source location are inhibited. Automatic in-lining is disabled, and there is no code motion across statement boundaries and no reduction of object lifetimes.

Support for In-Circuit Emulators

The host/target debugger also supports debugging through an in-circuit emulator. The 680X0 is controlled by the emulator, and R1000 host debugger commands are translated for the emulator, which in turn passes results back to the R1000. Thus the interaction can proceed using the R1000 high-level debugging model, or it can be

Guidelines to Select, Configure and Use an Ada Runtime Environment

treated at a lower level by using the emulator locally. In both cases, the emulator enables the developer to run the 680X0 application at speed on the processor.

The host/target debugging design allows some flexibility in the choice of in-circuit emulators. Customers with other models should contact a Rational representative for more details.

Other Components

To facilitate the production of real-time applications, an assembler and a linker are provided. These facilities enable developers to construct mixed-language applications.

Assembler

The assembler provides programmers with a powerful set of pseudo-operations, including a macro capability. Features include:

- Macro expansion capability
 - Number of arguments function
 - Automatic creation of local symbols
 - Argument concatenation
- Multiple relocatable program sections
- List files available on demand
- Conditional assembly
- Macro construct for statement iteration (looping)
- Local symbols
- Absolute and relocatable assembly
- Code and data alignment

Linker

The linker collects object modules produced by the code generator or assembler into a load module. This load module is a memory image of some portion of the logical address space for the program. A set of linker commands enables the user to specify the alignment and ordering of program sections and modules. Both absolute and relocatable load modules can be produced. The linker is structured to provide flexibility in the format of the final load module. This allows adaptation of the load module to the requirements of a particular customer's loader and operating system kernel. List files produced include a memory map, a symbol table file, and a cross-reference.

Reusable Software Components

Rational provides a library of packages that reduce implementation, test, and debugging time by providing reusable parts when working on the R1000. As part of the 680X0 Cross-Development Facility, a source license is provided so that these components can be incorporated in the applications running on a 680X0 that are developed on the R1000. These packages and procedures include:

Guidelines to Select, Configure and Use an Ada Runtime Environment

- Bounded_String
- Concurrent_Map_Generic
- List_Generic
- Map_Generic
- Queue_Generic
- Set_Generic
- Stack_Generic
- String_Map_Generic
- String_Table
- String_Uutilities
- Table_Formatter
- Table_Sort_Generic
- Time_Uutilities
- Unbounded_String

Relation to Other Products

The 680X0 can be connected to the R1000 using an RS232 port without obtaining other products. If an Ethernet connection is desired, Rational Networking is available.

For projects requiring code delivery with some other compiler, the M68000 Family Cross-Development Facility can be used during the development phase for efficient host/target debugging and rapid turnaround. Actual code delivery can be done with the Rational Target Build Utility, which enables the downloading of Ada source to another system for compilation. The Target Build Utility provides a complete change-tracking history on the R1000 so that the minimum number of units is downloaded and recompiled. In addition, a compilation and link script is downloaded for batch submission on the other system.

III. Documentation provided to help user configure runtime:

- Rational MC68020 Cross_Development Facility Manual

IV. Services to customize the runtime:

Rational offers an implementation program with each CDF (Cross Development Facility). The plan typically includes installation, training, and site specific customization of the CDF. The plan can be expanded to include runtime customization. The cost of the implementation plan for the MC68020 for a R1000 model 20 is \$15,000.

V. Cost of runtime source code:

- The runtime source code is provided with the CDF product. The cost of the MC68020 CDF for a R1000 model 20 is \$49,000.

VI. Source of Information: Vendor Input

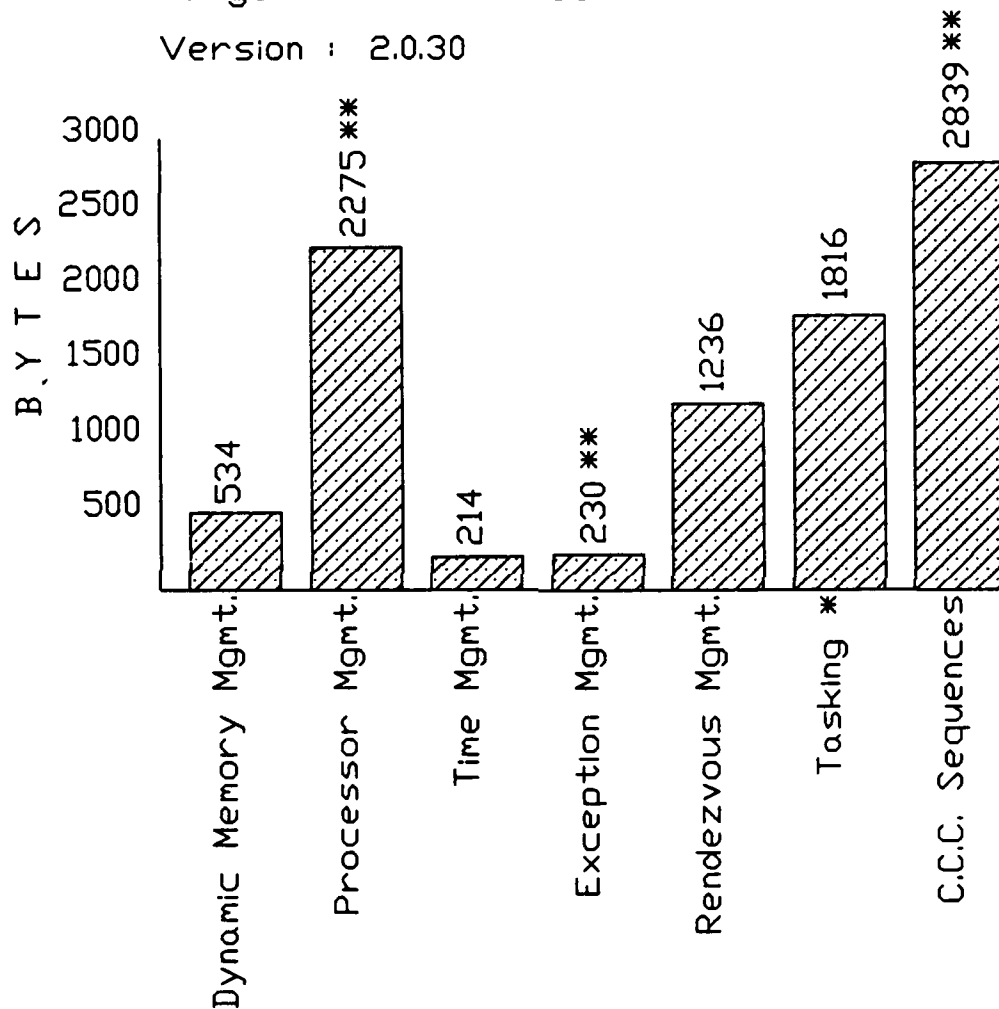
Guidelines to Select, Configure and Use an Ada Runtime Environment

Rational

Host : Rational R1000

Target : Motorola 68020

Version : 2.0.30



- Sum of ALL Components = 9144 bytes**

* Includes task creation, activation,
and termination

** See next page.

Note: the actual granularity is much finer
than represented here.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Vendor Supplied Component Description

These components of the runtime for the Motorola 68020 have been broken down into further storage requirements for code, data, and constant.

Processor Management

1770 bytes code
129 bytes constant
376 bytes data

Exception Management

142 bytes code
88 bytes constant

Commonly Called Code Sequences

2044 bytes code
711 bytes constant
84 bytes data

The maximum RTE including data :

7756 bytes code
928 bytes constant
460 bytes data

All sizes are in bytes. The actual granularity is much finer than represented here.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: 8 microseconds

Q2: How long, and for what reasons are interrupts disabled?

A2: Interrupts are disabled in time critical sections of the runtime. This is an important and complicated area and Rational would welcome the opportunity to discuss your needs in this area. For additional information please contact the Rational sales representative for your area.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: Rational makes optimizations which reduce the number of context switches required to perform a rendezvous in which the operator's accept statement or accept alternative (in the case of a select wait) contains no associated statements. For example:

```
begin
  accept e;
end;
or
begin
  accept e(P1 : T1; P2 : T2; ...);
end;
or
select
  accept e;
or
  accept e1(P1 : T1; P2 : T2; ...);
.
.
.
end select.
```

Note that depending on the path the selective wait takes, it is the form of that arm which is being taken which determines if this optimization can be performed.

Q4: What are the restrictions for representation clauses?

A4: The MC68000 cross-compiler supports the following representation clauses:

(F.1.5.) Representation Clauses

- Length clauses:

for Access_Type'Storage_Size use X;

If X is static and equal to zero, no collection is allocated. Any attempt to evaluate an allocator will raise the predefined Storage_Error exception. (Other values of X, which need not be static, are honored.)

Guidelines to Select, Configure and Use an Ada Runtime Environment

for Discrete_Type'Size use X;

for Fixed_Type'Small use X;

for Task_Object'Storage_Size use X;

for Task_Type'Storage_Size use X;

- Record representation clauses: The compiler supports both full and partial representation clauses for both discriminated and undiscriminated records.

- Enumeration representation clauses.

- Address clauses for objects.

(F.1.6.) Restrictions on Array and Record packing and Record Representation Clauses

- Arrays: Packed arrays of discretely (Integer and Enumeration types, including Booleans) are supported. Components of packed arrays occupy the minimum possible number of bits, which may range from 1 through 24.

- Records: A record field can consist of any number of bits between 1 and 32, inclusive; otherwise, it must be an integral number of words.

- Change of representation: Change of representation is supported wherever it is implied by support for representation specifications. In particular, implicit or explicit type conversions between array types or record types may cause packing or unpacking to occur; conversions between related enumeration types with different representations may result in table lookup operations.

The following example shows support for a change of representation of an array:

```
type Arr is array (1..10) of Boolean;  
type Brr is new Arr;  
pragma Pack (Brr)
```

```
X : Arr := (1..10 => false);  
Y : Brr := Brr (X);
```

Change of representation occurs in the type conversion to Brr.

(F.1.7.) Names Denoting Implementation_Dependent Components

- There are no user-visible implementation names.

(F.1.8.) Interpretation of Expressions That Appear in Address Clauses

- Address clauses can be used with statically allocated objects.

(F.1.9.) Unchecked Conversion

Guidelines to Select, Configure and Use an Ada Runtime Environment

- The target type of an unchecked conversion cannot be unconstrained array type or an unconstrained discriminated type.

(F.1.10.) Machine Code

- Machine-code insertions are not supported at this time.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Preemptive scheduling.

Q6: What are the restrictions on `pragma INLINE`?

A6: Subprograms that require elaboration checks will not be inlined.

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: Machine code insertion is not supported at this time.

Q9: What object types are supported by `pragma SHARED`?

A9: `Pragma SHARED` is supported for 16 bit discrete and fixed point types and access types.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

Maximum number of tasks:	Memory dependent
Default stack sizes:	Yes
Default task priority:	Yes
Optional numeric coprocessor:	Yes
Semaphore operations:	Yes
Exception trace:	Under control of the debugger
Fast interrupt entry:	Yes
Terminal I/O:	Output only

Additional items:

- Heap size.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the MC68000

package SYSTEM is

type Name is (Motorola_68K);

System_Name : constant Name := Motorola_68K;

Storage_Unit : constant := 8;

Memory_Size : constant := 2 ** 31 - 1;

Min_Int : constant := -(2 ** 31);

Max_Int : constant := +(2 ** 31) - 1;

Max_Digits : constant := 15;

Max_Mantissa : constant := 31;

Fine_Delta : constant := 2.0 ** (-31);

Tick : constant := 1.0E-03;

subtype Priority is Integer range 1 .. 254;

type Address is private;

Address_Zero : constant Address;

function "+" (Left : Address; Right : Integer) return Address;

function "+" (Left : Address; Right : Address) return Integer;

function "-" (Left : Address; Right : Address) return Integer;

function "-" (Left : Address; Right : Integer) return Address;

function "<" (Left, Right : Address) return Boolean;

function "<=" (Left, Right : Address) return Boolean;

function ">" (Left, Right : Address) return Boolean;

function "<=" (Left, Right : Address) return Boolean;

function To_Address (X : Integer) return Address;

function To_Integer (X : Address) return Integer;

private

...

end System;

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for the MC68020

package STANDARD is

```
type *Universal_Integer* is ...
type *Universal_Real* is ...
type *Universal_Fixed* is ...
type Boolean is (False, True);
type Integer is range -2_147_483_648 .. 2_147_483_647;
type Short_Short_Integer is range -128 .. 127;
type Short_Integer is range -32_768 .. 32_767;
type Float is digits 6
    range -3.40282346638529E+38 .. 3.40282346638529E+38;
type Long_Float is digits 15
    range -1.79769313486231E+308 .. 1.79769313486231E+308;
type Duration is delta 6.103515625000000E-05
    range -1.70141183460469E+38 .. 1.70141183460469E+38;
subtype Natural is Integer range 0 .. 2_147_483_647;
subtype Positive is Integer range 1 .. 2_147_483_647;

type String is array (Positive range <>) of Character;
Pragma Pack (String);
Package Ascii is ...

Constraint_Error : exception;
Numeric_Error : exception;
Storage_Error : exception;
Tasking_Error : exception;
Program_Error : exception;

type Character is ...
```

end Standard;

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
SofTech, Inc. Compiler version 2.0	VAX-11/780 and VAX 11/785 (under VAX/VMS 4.5)	8086, Intel iAPX 8086 80186, Intel iAPX 80186 80286, Intel iAPX 80286 (real mode) 80286, Intel iAPX 80286 (protected mode) 80386, Intel iAPX 80386 (compatibility mode) (All bare machines)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Individual subprogram extraction from packages only. The linker process is done in two steps. To prepare an object module, link and export are needed. There is a ELIMINATE option of the exporter. If used, not all the overhead would appear in the downloaded module. Actual amounts would depend upon the features of the application code.

II. Customization of the Runtime:

- Modifying-Replacing the source to selective runtime routines provided with the purchase of the compiler (i.e. Device Drivers)

III. Documentation provided to help user configure runtime:

- Runtime Support Library Guide

IV. Services to customize the runtime:

- Provided by SofTech
- Cost: Charges are negotiated for each case, depending on the complexity of the customization.

V. Cost of runtime source code:

- \$50,000

VI. Source of Information: Vendor input.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SofTech PIWG results for 8086/8087. Clock : 8MHz, 2 wait-states. PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	2092.0
Whetstone	"Whetstone" benchmark.	103*
C000001	Task creation/terminate, task type declared in package.	7259.5
C000002	Task creation/terminate, task type declared in procedure.	7302.2
C000003	Task creation/terminate, task type declared in block.	7257.1
D000001	Dynamic array, use and deallocation.	777.6
D000002	Dynamic array elaboration and initialization.	65554.2
D000003	Dynamic record allocation and deallocation.	833.3
E000001	Raise and handle an exception locally.	15.8
E000002	Raise and handle an exception in a package.	4917.6
E000004	Raise and handle an exception nested 4 deep in procedures.	9215.1
F000001	Set a BOOLEAN flag using a logical equation.	11.5
F000002	Set a BOOLEAN flag using an "if" test.	11.2
L000001	Simple "for" loop.	11.6
L000002	Simple "while" loop.	11.5
L000003	Simple "exit" loop.	11.5
P000001	Procedure call and return (inlineable), no parameters.	20.7
P000002	Procedure call and return (not inlineable), no parameters.	33.5
P000003	Procedure call and return (compiled separately).	21.1
P000004	Procedure call and return (Pragma INLINE used).	0.1
P000005	Procedure call and return (one parameter, in INTEGER).	26.3
P000006	Procedure call and return (one parameter, out INTEGER).	28.5
P000007	Procedure call and return (one parameter, in out INTEGER).	29.0
P000010	Procedure call and return (ten parameters, in INTEGER).	67.1
P000011	Procedure call and return (twenty parameters, in INTEGER).	111.5
P000012	Procedure call and return (ten parameters, in record_type).	124.5
P000013	Procedure call and return (twenty parameters, in record_type).	227.4
T000001	Minimum rendezvous, entry call and return.	3807.4
T000002	Task entry call and return (one task, one entry).	3765.4
T000003	Task entry call and return (two tasks, one entry each).	3802.0
T000004	Task entry call and return (one task, two entries).	5589.6
T000005	Active entry and return (ten tasks, one entry each).	3758.4
T000006	Task entry call and return (one task, ten entries).	13984.6
T000007	Minimum rendezvous, entry call and return.	3210.3

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SofTech PIWG results for 8086/8087. Clock : 8MHz, 2 wait-states (Tests were compiled with FAST TASKING pragmas). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	2092.0
Whetstone	"Whetstone" benchmark.	103*
C000001	Task creation/terminate, task type declared in package.	7259.5
C000002	Task creation/terminate, task type declared in procedure.	7302.2
C000003	Task creation/terminate, task type declared in block.	7257.1
D000001	Dynamic array, use and deallocation.	777.6
D000002	Dynamic array elaboration and initialization.	65554.2
D000003	Dynamic record allocation and deallocation.	833.3
E000001	Raise and handle an exception locally.	15.8
E000002	Raise and handle an exception in a package.	4917.6
E000004	Raise and handle an exception nested four deep.	9215.1
F000001	Set a BOOLEAN flag using a logical equation.	11.5
F000002	Set a BOOLEAN flag using an "if" test.	11.2
L000001	Simple "for" loop.	11.6
L000002	Simple "while" loop.	11.5
L000003	Simple "exit" loop.	11.5
P000001	Procedure call and return (inlineable), no parameters.	20.7
P000002	Procedure call and return (not inlineable), no parameters.	33.5
P000003	Procedure call and return (compiled separately).	21.1
P000004	Procedure call and return (Pragma INLINE used).	0.1
P000005	Procedure call and return (one parameter, in INTEGER).	26.3
P000006	Procedure call and return (one parameter, out INTEGER).	28.5
P000007	Procedure call and return (one parameter, in out INTEGER).	29.0
P000010	Procedure call and return (ten parameters, in INTEGER).	67.1
P000011	Procedure call and return (twenty parameters, in INTEGER).	111.5
P000012	Procedure call and return (ten parameters, in record_type).	124.5
P000013	Procedure call and return (twenty parameters, in record_type).	227.4
T000001	Minimum rendezvous, entry call and return.	96.1
T000002	Task entry call and return (one task, one entry).	94.6
T000003	Task entry call and return (two tasks, one entry each).	130.3
T000004	Task entry call and return (one task, two entries).	130.3
T000005	Active entry and return (ten tasks, one entry each).	85.8
T000006	Task entry call and return (one task, ten entries).	85.8

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SofTech PIWG results for 80186/8087. Clock : 8MHz, 2 wait-states. PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	1593.4
Whetstone	"Whetstone" benchmark.	118*
C000001	Task creation/terminate, task type declared in package.	6408.0
C000002	Task creation/terminate, task type declared in procedure.	6443.8
C000003	Task creation/terminate, task type declared in block.	6409.9
D000001	Dynamic array, use and deallocation.	629.3
D000002	Dynamic array elaboration and initialization.	52302.9
D000003	Dynamic record allocation and deallocation.	669.7
E000001	Raise and handle an exception locally.	14.2
E000002	Raise and handle an exception in a package.	3908.1
E000004	Raise and handle an exception nested four deep.	7367.6
F000001	Set a BOOLEAN flag using a logical equation.	10.4
F000002	Set a BOOLEAN flag using an "if" test.	9.6
L000001	Simple "for" loop.	8.4
L000002	Simple "while" loop.	7.4
L000003	Simple "exit" loop.	7.4
P000001	Procedure call and return (inlineable), no parameters.	15.4
P000002	Procedure call and return (not inlineable), no parameters.	27.1
P000003	Procedure call and return (compiled separately).	15.4
P000004	Procedure call and return (Pragma INLINE used).	0.1
P000005	Procedure call and return (one parameter, in INTEGER).	18.0
P000006	Procedure call and return (one parameter, out INTEGER).	21.6
P000007	Procedure call and return (one parameter, in out INTEGER).	22.2
P000010	Procedure call and return (ten parameters, in INTEGER).	47.5
P000011	Procedure call and return (twenty parameters, in INTEGER).	77.9
P000012	Procedure call and return (ten parameters, in record_type).	97.3
P000013	Procedure call and return (twenty parameters, in record_type).	178.9
T000001	Minimum rendezvous, entry call and return.	3241.6
T000002	Task entry call and return (one task, one entry).	3241.4
T000003	Task entry call and return (two tasks, one entry each).	3274.5
T000004	Task entry call and return (one task, two entries).	4738.0
T000005	Active entry and return (ten tasks, one entry each).	3244.6
T000006	Task entry call and return (one task, ten entries).	11543.5
T000007	Minimum rendezvous, entry call and return (one task, one entry).	2716.1

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SofTech PIWG results for 80186/8087. Clock : 8MHz, 2 wait-states.(Tests were compiled with FAST TASKING pragmas). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	1593.4
Whetstone	"Whetstone" benchmark.	118*
C000001	Task creation/terminate, task type declared in package.	6408.0
C000002	Task creation/terminate, task type declared in procedure.	6443.8
C000003	Task creation/terminate, task type declared in block.	6409.9
D000001	Dynamic array, use and deallocation.	629.3
D000002	Dynamic array elaboration and initialization.	52302.9
D000003	Dynamic record allocation and deallocation.	669.7
E000001	Raise and handle an exception locally.	14.2
E000002	Raise and handle an exception in a package.	3908.1
E000004	Raise and handle an exception nested four deep.	7367.6
F000001	Set a BOOLEAN flag using a logical equation.	10.4
F000002	Set a BOOLEAN flag using an "if" test.	9.6
L000001	Simple "for" loop.	8.4
L000002	Simple "while" loop.	7.4
L000003	Simple "exit" loop.	7.4
P000001	Procedure call and return (inlineable), no parameters.	15.4
P000002	Procedure call and return (not inlineable), no parameters.	27.1
P000003	Procedure call and return (compiled separately).	15.4
P000004	Procedure call and return (Pragma INLINE used).	0.1
P000005	Procedure call and return (one parameter, in INTEGER).	18.0
P000006	Procedure call and return (one parameter, out INTEGER).	21.6
P000007	Procedure call and return (one parameter, in out INTEGER).	22.2
P000010	Procedure call and return (ten parameters, in INTEGER).	47.5
P000011	Procedure call and return (twenty parameters, in INTEGER).	77.9
P000012	Procedure call and return (ten parameters, in record_type).	97.3
P000013	Procedure call and return (twenty parameters, in record_type).	178.9
T000001	Minimum rendezvous, entry call and return.	87.2
T000002	Task entry call and return (one task, one entry).	86.5
T000003	Task entry call and return (two tasks, one entry each).	116.2
T000004	Task entry call and return (one task, two entries).	115.4
T000005	Active entry and return (ten tasks, one entry each).	78.9
T000006	Task entry call and return (one task, ten entries).	77.9

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SoftTech PIWG results for 80286P/80287. Clock : 10MHz, 0 wait-states. PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	602.3
Whetstone	"Whetstone" benchmark.	183*
C000001	Task creation/terminate, task type declared in package.	2675.5
C000002	Task creation/terminate, task type declared in procedure.	2674.9
C000003	Task creation/terminate, task type declared in block.	2656.6
D000001	Dynamic array, use and deallocation.	380.7
D000002	Dynamic array elaboration and initialization.	16019.3
D000003	Dynamic record allocation and deallocation.	398.5
E000001	Raise and handle an exception locally.	5.3
E000002	Raise and handle an exception in a package.	1580.3
E000004	Raise and handle an exception nested 4 deep in procedures.	2940.1
F000001	Set a BOOLEAN flag using a logical equation.	3.0
F000002	Set a BOOLEAN flag using an "if" test.	3.1
L000001	Simple "for" loop.	2.8
L000002	Simple "while" loop.	2.3
L000003	Simple "exit" loop.	2.3
P000001	Procedure call and return (inlineable), no parameters.	7.9
P000002	Procedure call and return (not inlineable), no parameters.	12.9
P000003	Procedure call and return (compiled separately).	7.9
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	9.1
P000006	Procedure call and return (one parameter, out INTEGER).	9.8
P000007	Procedure call and return (one parameter, in out INTEGER).	8.9
P000010	Procedure call and return (ten parameters, in INTEGER).	16.6
P000011	Procedure call and return (twenty parameters, in INTEGER).	24.1
P000012	Procedure call and return (ten parameters, in record_type).	39.3
P000013	Procedure call and return (twenty parameters, in record_type).	70.6
T000001	Minimum rendezvous, entry call and return.	1313.1
T000002	Task entry call and return (one task, one entry).	1312.9
T000003	Task entry call and return (two tasks, one entry each).	1323.9
T000004	Task entry call and return (one task, two entries).	1834.6
T000005	Active entry and return (ten tasks, one entry each).	1309.7
T000006	Task entry call and return (one task, ten entries).	4155.1
T000007	Minimum rendezvous, entry call and return (one task, one entry).	1089.1

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SoTech PIWG results for 80286P/80287. Clock : 10MHz, 0 wait-states. (Tests were compiled with FAST TASKING pragmas). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhystone	"Dhystone" benchmark.	602.3
Whetstone	"Whetstone" benchmark.	183*
C000001	Task creation/terminate, task type declared in package.	2675.5
C000002	Task creation/terminate, task type declared in procedure.	2674.9
C000003	Task creation/terminate, task type declared in block.	2656.6
D000001	Dynamic array, use and deallocation.	380.7
D000002	Dynamic array elaboration and initialization.	16019.3
D000003	Dynamic record allocation and deallocation.	398.5
E000001	Raise and handle an exception locally.	5.3
E000002	Raise and handle an exception in a package.	1580.3
E000004	Raise and handle an exception nested four deep.	2940.1
F000001	Set a BOOLEAN flag using a logical equation.	3.0
F000002	Set a BOOLEAN flag using an "if" test.	3.1
L000001	Simple "for" loop.	2.8
L000002	Simple "while" loop.	2.3
L000003	Simple "exit" loop.	2.3
P000001	Procedure call and return (inlineable), no parameters.	7.9
P000002	Procedure call and return (not inlineable), no parameters.	12.9
P000003	Procedure call and return (compiled separately).	7.9
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	9.1
P000006	Procedure call and return (one parameter, out INTEGER).	9.8
P000007	Procedure call and return (one parameter, in out INTEGER).	8.9
P000010	Procedure call and return (ten parameters, in INTEGER).	16.6
P000011	Procedure call and return (twenty parameters, in INTEGER).	24.1
P000012	Procedure call and return (ten parameters, in RECORD).	39.3
P000013	Procedure call and return (twenty parameters, in RECORD).	70.6
T000001	Minimum rendezvous, entry call and return.	68.3
T000002	Task entry call and return (one task, one entry).	68.4
T000003	Task entry call and return (two tasks, one entry each).	80.1
T000004	Task entry call and return (one task, two entries).	80.1
T000005	Active entry and return (ten tasks, one entry each).	65.4
T000006	Task entry call and return (one task, ten entries).	65.4

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SoFTech PIWG results for 80286R/80287. Clock : 10MHz, 0 wait-states. PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhystone	"Dhystone" benchmark.	463.1
Whetstone	"Whetstone" benchmark.	192*
C000001	Task creation/terminate, task type declared in package.	1884.7
C000002	Task creation/terminate, task type declared in procedure.	1891.7
C000003	Task creation/terminate, task type declared in block.	1878.7
D000001	Dynamic array, use and deallocation.	183.0
D000002	Dynamic array elaboration and initialization.	14303.6
D000003	Dynamic record allocation and deallocation.	194.4
E000001	Raise and handle an exception locally.	3.0
E000002	Raise and handle an exception in a package.	1106.6
E000004	Raise and handle an exception nested four deep.	2087.3
F000001	Set a BOOLEAN flag using a logical equation.	2.0
F000002	Set a BOOLEAN flag using an "if" test.	3.1
L000001	Simple "for" loop.	2.9
L000002	Simple "while" loop.	2.3
L000003	Simple "exit" loop.	2.3
P000001	Procedure call and return (inlineable), no parameters.	5.4
P000002	Procedure call and return (not inlineable), no parameters.	8.9
P000003	Procedure call and return (compiled separately).	5.4
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	5.0
P000006	Procedure call and return (one parameter, out INTEGER).	6.9
P000007	Procedure call and return (one parameter, in out INTEGER).	6.1
P000010	Procedure call and return (ten parameters, in INTEGER).	13.6
P000011	Procedure call and return (twenty parameters, in INTEGER).	22.4
P000012	Procedure call and return (ten parameters, in record_type).	24.8
P000013	Procedure call and return (twenty parameters, in record_type).	44.3
T000001	Minimum rendezvous, entry call and return.	991.6
T000002	Task entry call and return (one task, one entry).	991.1
T000003	Task entry call and return (two tasks, one entry each).	999.1
T000004	Task entry call and return (one task, two entries).	1417.8
T000005	Active entry and return (ten tasks, one entry each).	988.6
T000006	Task entry call and return (one task, ten entries).	3340.5
T000007	Minimum rendezvous, entry call and return.	831.9

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SofTech PIWG results for 80286R/80287. Clock : 10MHz, 0 wait-states. (Tests were compiled with FAST TASKING pragmas). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	463.1
Whetstone	"Whetstone" benchmark.	192*
C000001	Task creation/terminate, task type declared in package.	1884.7
C000002	Task creation/terminate, task type declared in procedure.	1891.7
C000003	Task creation/terminate, task type declared in block.	1878.7
D000001	Dynamic array, use and deallocation.	183.0
D000002	Dynamic array elaboration and initialization.	14303.6
D000003	Dynamic record allocation and deallocation.	194.4
E000001	Raise and handle an exception locally.	3.0
E000002	Raise and handle an exception in a package.	1106.6
E000004	Raise and handle an exception nested four deep.	2087.3
F000001	Set a BOOLEAN flag using a logical equation.	2.0
F000002	Set a BOOLEAN flag using an "if" test.	3.1
L000001	Simple "for" loop.	2.9
L000002	Simple "while" loop.	2.3
L000003	Simple "exit" loop.	2.3
P000001	Procedure call and return (inlineable), no parameters.	5.4
P000002	Procedure call and return (not inlineable), no parameters.	8.9
P000003	Procedure call and return (compiled separately).	5.4
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	5.0
P000006	Procedure call and return (one parameter, out INTEGER).	6.9
P000007	Procedure call and return (one parameter, in out INTEGER).	6.1
P000010	Procedure call and return (ten parameters, in INTEGER).	13.6
P000011	Procedure call and return (twenty parameters, in INTEGER).	22.4
P000012	Procedure call and return (ten parameters, in record_type).	24.8
P000013	Procedure call and return (twenty parameters, in record_type).	44.3
T000001	Minimum rendezvous, entry call and return.	24.7
T000002	Task entry call and return (one task, one entry).	24.6
T000003	Task entry call and return (two tasks, one entry each).	33.4
T000004	Task entry call and return (one task, two entries).	33.0
T000005	Active entry and return (ten tasks, one entry each).	21.6
T000006	Task entry call and return (one task, ten entries).	21.3

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SofTech PIWG results for 80386P/80287. Clock : 16MHz, 0 - 3 wait-states. PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	426.8
Whetstone	"Whetstone" benchmark.	531*
C000001	Task creation/terminate, task type declared in package.	1968.1
C000002	Task creation/terminate, task type declared in procedure.	1940.6
C000003	Task creation/terminate, task type declared in block.	1939.0
D000001	Dynamic array, use and deallocation.	248.3
D000002	Dynamic array elaboration and initialization.	10022.6
D000003	Dynamic record allocation and deallocation.	260.4
E000001	Raise and handle an exception locally.	2.9
E000002	Raise and handle an exception in a package.	1063.9
E000004	Raise and handle an exception nested four deep.	1972.6
F000001	Set a BOOLEAN flag using a logical equation.	1.5
F000002	Set a BOOLEAN flag using an "if" test.	1.6
L000001	Simple "for" loop.	1.7
L000002	Simple "while" loop.	1.6
L000003	Simple "exit" loop.	1.6
P000001	Procedure call and return (inlineable), no parameters.	5.9
P000002	Procedure call and return (not inlineable), no parameters.	9.1
P000003	Procedure call and return (compiled separately).	5.9
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	6.4
P000006	Procedure call and return (one parameter, out INTEGER).	6.5
P000007	Procedure call and return (one parameter, in out INTEGER).	6.5
P000010	Procedure call and return (ten parameters, in INTEGER).	11.4
P000011	Procedure call and return (twenty parameters, in INTEGER).	17.9
P000012	Procedure call and return (ten parameters, in record_type).	28.1
P000013	Procedure call and return (twenty parameters, in record_type).	51.2
T000001	Minimum rendezvous, entry call and return.	865.9
T000002	Task entry call and return (one task, one entry).	855.5
T000003	Task entry call and return (two tasks, one entry each).	874.3
T000004	Task entry call and return (one task, two entries).	1134.4
T000005	Active entry and return (ten tasks, one entry each).	865.2
T000006	Task entry call and return (one task, ten entries).	2336.1
T000007	Minimum rendezvous, entry call and return.	708.8

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SoftTech PIWG results for 80386P/80287. Clock : 16MHz, 0 - 3 wait-states. (Tests were compiled with FAST TASKING pragmas). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	426.8
Whetstone	"Whetstone" benchmark.	531*
C000001	Task creation/terminate, task type declared in package.	1968.1
C000002	Task creation/terminate, task type declared in procedure.	1940.6
C000003	Task creation/terminate, task type declared in block.	1939.0
D000001	Dynamic array, use and deallocation.	248.3
D000002	Dynamic array elaboration and initialization.	10022.6
D000003	Dynamic record allocation and deallocation.	260.4
E000001	Raise and handle an exception locally.	2.9
E000002	Raise and handle an exception in a package.	1063.9
E000004	Raise and handle an exception nested four deep.	1972.6
F000001	Set a BOOLEAN flag using a logical equation.	1.5
F000002	Set a BOOLEAN flag using an "if" test.	1.6
L000001	Simple "for" loop.	1.7
L000002	Simple "while" loop.	1.6
L000003	Simple "exit" loop.	1.6
P000001	Procedure call and return (inlineable), no parameters.	5.9
P000002	Procedure call and return (not inlineable), no parameters.	9.1
P000003	Procedure call and return (compiled separately).	5.9
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	6.4
P000006	Procedure call and return (one parameter, out INTEGER).	6.5
P000007	Procedure call and return (one parameter, in out INTEGER).	6.5
P000010	Procedure call and return (ten parameters, in INTEGER).	11.4
P000011	Procedure call and return (twenty parameters, in INTEGER).	17.9
P000012	Procedure call and return (ten parameters, in record_type).	28.1
P000013	Procedure call and return (twenty parameters, in record_type).	51.2
T000001	Minimum rendezvous, entry call and return.	51.2
T000002	Task entry call and return (one task, one entry).	52.3
T000003	Task entry call and return (two tasks, one entry each).	60.5
T000004	Task entry call and return (one task, two entries).	61.2
T000005	Active entry and return (ten tasks, one entry each).	49.5
T000006	Task entry call and return (one task, ten entries).	50.6

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SofTech PIWG results for 80386R/80287. Clock : 16MHz, 0 - 3 wait-states. PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhrystone	"Dhrystone" benchmark.	330.1
Whetstone	"Whetstone" benchmark.	593*
C000001	Task creation/terminate, task type declared in package.	1421.0
C000002	Task creation/terminate, task type declared in procedure.	1405.5
C000003	Task creation/terminate, task type declared in block.	1402.7
D000001	Dynamic array, use and deallocation.	120.8
D000002	Dynamic array elaboration and initialization.	8503.4
D000003	Dynamic record allocation and deallocation.	127.8
E000001	Raise and handle an exception locally.	2.3
E000002	Raise and handle an exception in a package.	745.5
E000004	Raise and handle an exception nested four deep.	1401.2
F000001	Set a BOOLEAN flag using a logical equation.	1.5
F000002	Set a BOOLEAN flag using an "if" test.	1.6
L000001	Simple "for" loop.	1.7
L000002	Simple "while" loop.	1.6
L000003	Simple "exit" loop.	1.6
P000001	Procedure call and return (inlineable), no parameters.	4.2
P000002	Procedure call and return (not inlineable), no parameters.	6.5
P000003	Procedure call and return (compiled separately).	4.2
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	4.6
P000006	Procedure call and return (one parameter, out INTEGER).	4.9
P000007	Procedure call and return (one parameter, in out INTEGER).	4.9
P000010	Procedure call and return (ten parameters, in INTEGER).	9.8
P000011	Procedure call and return (twenty parameters, in INTEGER).	16.2
P000012	Procedure call and return (ten parameters, in record_type).	17.1
P000013	Procedure call and return (twenty parameters, in record_type).	30.6
T000001	Minimum rendezvous, entry call and return.	671.5
T000002	Task entry call and return (one task, one entry).	662.0
T000003	Task entry call and return (two tasks, one entry each).	678.9
T000004	Task entry call and return (one task, two entries).	888.6
T000005	Active entry and return (ten tasks, one entry each).	674.0
T000006	Task entry call and return (one task, ten entries).	1888.1
T000007	Minimum rendezvous, entry call and return (one task, one entry).	554.9

* WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

SoFTech PIWG results for 80386R/80287. Clock : 16MHz, 0 - 3 wait-states. (Tests were compiled with FAST TASKING pragmas). PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
Dhystone	"Dhystone" benchmark.	330.1
Whetstone	"Whetstone" benchmark.	593*
C000001	Task creation/terminate, task type declared in package.	1421.0
C000002	Task creation/terminate, task type declared in procedure.	1405.5
C000003	Task creation/terminate, task type declared in block.	1402.7
D000001	Dynamic array, use and deallocation.	120.8
D000002	Dynamic array elaboration and initialization.	8503.4
D000003	Dynamic record allocation and deallocation.	127.8
E000001	Raise and handle an exception locally.	2.3
E000002	Raise and handle an exception in a package.	745.5
E000004	Raise and handle an exception nested four deep.	1401.2
F000001	Set a BOOLEAN flag using a logical equation.	1.5
F000002	Set a BOOLEAN flag using an "if" test.	1.6
L000001	Simple "for" loop.	1.7
L000002	Simple "while" loop.	1.6
L000003	Simple "exit" loop.	1.6
P000001	Procedure call and return (inlineable), no parameters.	4.2
P000002	Procedure call and return (not inlineable), no parameters.	6.5
P000003	Procedure call and return (compiled separately).	4.2
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	4.6
P000006	Procedure call and return (one parameter, out INTEGER).	4.9
P000007	Procedure call and return (one parameter, in out INTEGER).	4.9
P000010	Procedure call and return (ten parameters, in INTEGER).	9.8
P000011	Procedure call and return (twenty parameters, in INTEGER).	16.2
P000012	Procedure call and return (ten parameters, in record_type).	17.1
P000013	Procedure call and return (twenty parameters, in record_type).	30.6
T000001	Minimum rendezvous, entry call and return.	20.1
T000002	Task entry call and return (one task, one entry).	20.4
T000003	Task entry call and return (two tasks, one entry each).	27.2
T000004	Task entry call and return (one task, two entries).	27.1
T000005	Active entry and return (ten tasks, one entry each).	18.5
T000006	Task entry call and return (one task, ten entries).	18.6

* WHETSTONE : units are in KWIPS not in microseconds.

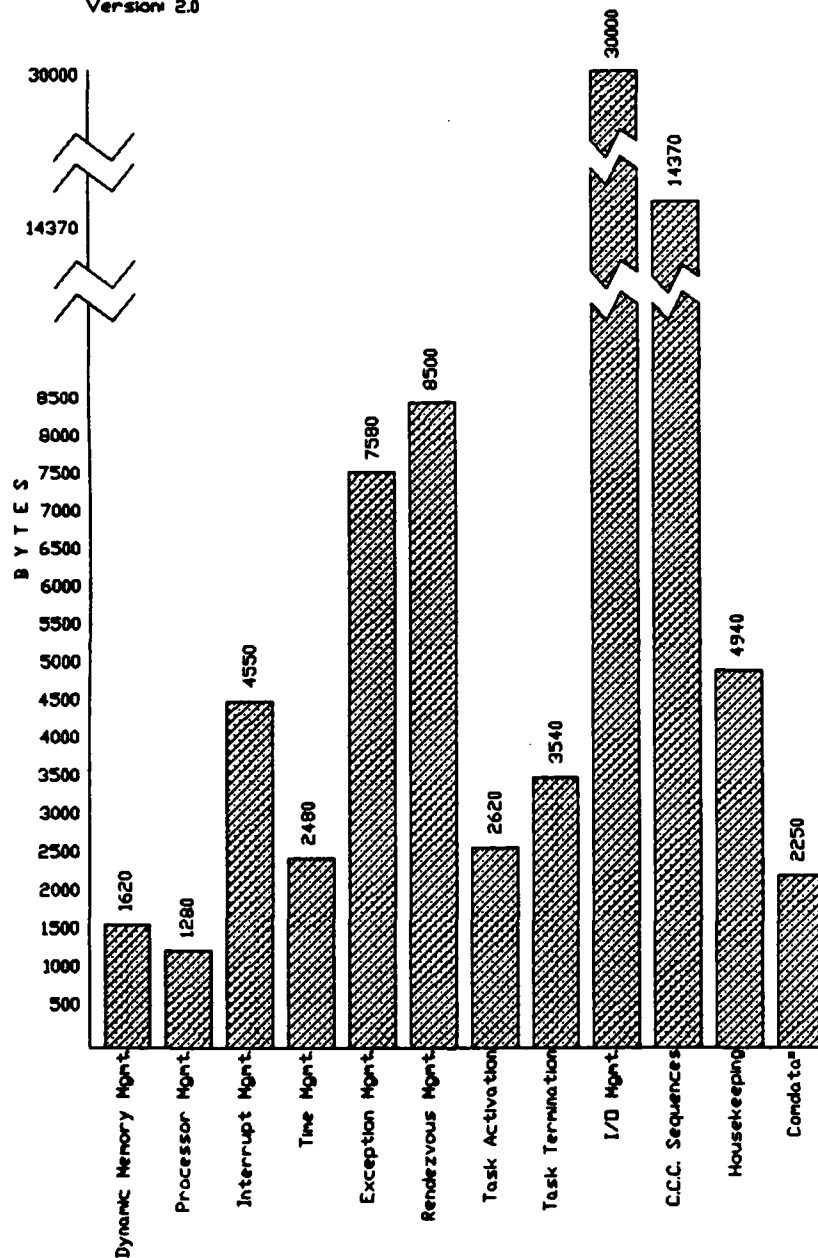
Guidelines to Select, Configure and Use an Ada Runtime Environment

SofTech, Inc.

Host: VAX/VMS

Target: IAPX 8086, 80186, 80286R, 80286P, 80386

Version: 2.0



Sum of ALL components = 83730 bytes

Figure sizes given are for the maximum overhead

(Refer to Degree of Configurability / Linker capability on previous page)

- Condata was described by the vendor as the combined read/write area for all of the Ada libraries.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Appendix F Notes

The following excerpts are taken from the Ada Compiler Validation Summary Report for SofTech Ada86 Version 1.34. [29]

Representation Clause Restrictions

Address Clauses

Address clauses are supported for the following items:

1. Scalar or composite objects with the following restrictions:
 - (a) The object must not be nested within a subprogram or task directly or indirectly.
 - (b) The size of the object must be determinable at the time of compilation.
2. Subprograms with the following restrictions:
 - (a) The subprogram can not be a library subprogram (LRM requirement).
 - (b) Any subprogram declared within a subprogram having an address clause will be placed in relocatable sections.
3. Entries - An address clause may specify a hardware interrupt with which the entry is to be associated.

Length Clause

TSTORAGE_SIZE for task type T specifies the number of bytes to be allocated for the runtime stack of each task object of type T.

Enumeration Representation Clause

In the absence of a representation specification for an enumeration type T, the internal representation of TFIRST is 0. The default SIZE for a stand-alone object of enumeration type T will be the smallest of the values 8, 16, or 32, such that the internal representation of TFIRST and TLAST both fall within the range:

$$-2^{**}(T\text{SIZE} - 1) .. 2^{**}(T\text{SIZE} - 1) - 1.$$

Length specifications of the form:

for TSIZE use N;

and/or enumeration representations of the form:

for T use aggregate

Are permitted for N in 2..32, provided the representations and the SIZE conform to the relationship specified above, or else for N in 1..31, provided that

Guidelines to Select, Configure and Use an Ada Runtime Environment

the internal representation of $TFIRST \geq 0$ and the representation of $TLAST = 2^{**}(TSIZE)-1$.

for components of enumeration types within packed composite objects, the smaller of the default stand-alone SIZE and the SIZE from a length specification is used.

In accordance with the rules of Ada, and the implementation of package STANDARD, enumeration representation on types derived from the predefined type BOOLEAN are not accepted, but length specifications are accepted.

Record Representation Clause

A length specification of the form

for TSIZE use N;

Will cause arrays and records to be packed, if required, to accommodate the length specification.

The PACK pragma may be used to minimize wasted space between components of arrays and records. The pragma causes the type representation to be chosen such that storage space requirements are minimized at the possible expense of data access time and code space.

A record type representation specification may be used to describe the allocation of components in a record. Bits are numbered 0..7 from the right. (Bit 8 starts at the right of the next higher-numbered byte).

The alignment clause of the form:

at mod N

can specify alignment of 1 (byte) or 2 (word).

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the iAPX286 Operating in Real Address Mode

```
package SYSTEM is

  type WORD is range 0..16#FFFF#;
  for WORD'SIZE use 16;

  type BYTE is range 0..255;
  for BYTE'SIZE use 8;

  subtype REGISTER          is SYSTEM.WORD;
  subtype SEGMENT_REGISTER is SYSTEM.REGISTER;
  subtype OFFSET_REGISTER  is SYSTEM.REGISTER;

  type ADDRESS is
    record
      SEGMENT:  SYSTEM.SEGMENT_REGISTER;
      OFFSET :  SYSTEM.OFFSET_REGISTER;
    end record;

  for ADDRESS'SIZE use 32;

  for ADDRESS use
    record
      OFFSET at 0 range 0..15;
      SEGMENT at 2 range 0..15;
    end record;

  NULL_ADDRESS : constant SYSTEM.ADDRESS := (0,0);

  subtype TO_ADDRESS is SYSTEM.REGISTER;

  type ABSOLUTE_ADDRESS is range 0..16#FFFF#;
  for ABSOLUTE_ADDRESS'SIZE use 20;

  type NAME is (VAX780_VMS, iAPX86, iAPX186, iAPX286R);

  SYSTEM_NAME : constant SYSTEM.NAME := (SYSTEM.iAPX286R);
               --Intel 80286 in real address mode.

  STORAGE_UNIT : constant := 8;
  MEMORY_SIZE  : constant := (2**20)-1;

  MIN_INT      : constant := -(2**31);
  MAX_INT      : constant := (2**31)-1;

  MAX_DIGITS   : constant := 15;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the iAPX286 Operating in Real Address Mode (Continued)

```

MAX_MANTISSA : constant := 31;
FINE_DELTA   : constant := 4.656_612_873_077_392_578_125E-10;

type INTERRUPT_TYPE_NUMBER is range 0..255;

DIVIDE_ERROR_INTERRUPT      : constant := 0;
SINGLE_STEP_INTERRUPT       : constant := 1;
NON_MASKABLE_INTERRUPT     : constant := 2;
OVERFLOW_INTERRUPT         : constant := 4;
RSL_CLOCK_INTERRUPT        : constant := 64;
-- for iAPX86, iAPX286R
-- for iAPX186, it is 18
DELAY_EXPIRY_INTERRUPT     : constant := 8;
-- for iAPX186 only
NUMERIC_PROCESSOR_INTERRUPT : constant := 16;
-- for iAPX286 only
-- for iAPX86 it is 71
-- for iAPX186 it is 15
DISPATCH_CODE_INTERRUPT   : constant := 32;
CHECK_STACK_INTERRUPT      : constant := 48;
ENTER_SUBPROGRAM_WITHOUT_LPP_INTERRUPT : constant := 49;
ENTER_SUBPROGRAM_INTERRUPT : constant := 50;
PROGRAM_ERROR_INTERRUPT    : constant := 53;
CONSTRAINT_ERROR_INTERRUPT : constant := 54;
NUMERIC_ERROR_INTERRUPT    : constant := 55;
ALLOCATE_OBJECT_INTERRUPT  : constant := 56;
BOUND_EXCEPTION_INTERRUPT  : constant := 05;
UNDEFINED_OPCODE_EXCEPTION_INTERRUPT : constant := 6;
PROCESSOR_EXTENSION_NOT_AVAILABLE_INTERRUPT : constant := 7;
ENTER_INNOCUOUS_CRITICAL_REGION_INTERRUPT : constant := 33;
LEAVE_INNOCUOUS_CRITICAL_REGION_INTERRUPT : constant := 34;

type ENTRY_KIND is
( ORDINARY_INTERRUPT_ENTRY, -- ordinary interrupt entry
  PROMPT                     , -- fast interrupt entry
  SIMPLE_QUICK                , -- quick interrupt entry
  NO_NDP_SIMPLE_QUICK         , -- quick interrupt entry
  SIGNALLING_QUICK            , -- quick interrupt entry
  NON_MASKABLE                , -- non-maskable interrupt entry
  NO_NDP_NON_MASKABLE         , -- non-maskable interrupt entry );

TICK : constant := 6.510_415_666_666_666_666_667E-6;
-- for iAPX86 only

TICK : constant := 0.000_015; -- 15 microseconds
-- for iAPX186 only

```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the iAPX286 Operating in Real Address Mode (Continued)

```
TICKS_PER_SECOND : constant := 20163.93442_62209_52836_06557;
                        -- approximate
                        -- for iAPX286 only

type DIRECTION_TYPE is ( AUTO_INCREMENT, AUTO_DECREMENT );

type PARITY_TYPE is ( ODD, EVEN );

type FLAGS_REGISTER is
  record
    NESTED_TASK          : BOOLEAN          := FALSE;
                                -- for iAPX286 only
    IO_PRIVILEGE_LEVEL   : NATURAL range 0..3 := 0;
                                -- for iAPX286 only
    OVERFLOW             : BOOLEAN          := FALSE;
    DIRECTION            : SYSTEM.DIRECTION_TYPE := SYSTEM.AUTO_INCREMENT;
    INTERRUPT            : BOOLEAN          := TRUE;
    TRAP                 : BOOLEAN          := FALSE;
    SIGN                 : BOOLEAN          := FALSE;
    ZERO                 : BOOLEAN          := TRUE;
    AUXILIARY            : BOOLEAN          := FALSE;
    PARITY                : SYSTEM.PARITY_TYPE := SYSTEM.EVEN;
    CARRY                : BOOLEAN          := FALSE;
  end record;

for FLAGS_REGISTER use
  record
    NESTED_TASK          at 0 range 14..14; -- for iAPX286 only
    IO_PRIVILEGE_LEVEL   at 0 range 12..13; -- for iAPX286 only
    OVERFLOW             at 0 range 11..11; -- for iAPX286 only
    DIRECTION            at 0 range 10..10; -- for iAPX286 only
    INTERRUPT            at 0 range 9..9;   -- for iAPX286 only
    TRAP                 at 0 range 8..8;   -- for iAPX286 only
    SIGN                 at 0 range 7..7;   -- for iAPX286 only
    ZERO                 at 0 range 6..6;   -- for iAPX286 only
    AUXILIARY            at 0 range 4..4;   -- for iAPX286 only
    PARITY                at 0 range 2..2;   -- for iAPX286 only
    CARRY                at 0 range 0..0;   -- for iAPX286 only
  end record;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the iAPX286 Operating in Real Address Mode (Continued)

```
NORMALIZED_FLAGS_REGISTER : constant SYSTEM.FLAGS_REGISTER :=
  ( NESTED_TASK           => FALSE, -- for iAPX286 only
    IO_PRIVILEGE_LEVEL    => 0,     -- for iAPX286 only
    OVERFLOW              => FALSE,
    DIRECTION             => SYSTEM.AUTO_INCREMENT,
    INTERRUPT             => TRUE,
    TRAP                  => FALSE,
    SIGN                  => FALSE,
    ZERO                  => TRUE,
    AUXILIARY             => FALSE,
    PARITY                => SYSTEM.EVEN,
    CARRY                 => FALSE );

subtype PRIORITY is INTEGER range 1..15;

UNRESOLVED_REFERENCE : exception;
SYSTEM_ERROR         : exception;

function EFFECTIVE_ADDRESS
  ( A : in SYSTEM.ADDRESS)
return SYSTEM.ABSOLUTE_ADDRESS;

function FAST_EFFECTIVE_ADDRESS
  ( A : in SYSTEM.ADDRESS)
return SYSTEM.ABSOLUTE_ADDRESS;

function TWOS_COMPLEMENT_OF
  ( W : in SYSTEM.WORD)
return SYSTEM.WORD;

procedure ADD_TO_ADDRESS
  ( ADDR      : in out SYSTEM.ADDRESS;
    OFFSET    : in SYSTEM.OFFSET_REGISTER );

procedure SUBTRACT_FROM_ADDRESS
  ( ADDR      : in out SYSTEM.ADDRESS;
    OFFSET    : in SYSTEM.OFFSET_REGISTER );

function INTERRUPT_TYPE_NUMBER_OF
  ( A : in SYSTEM.ADDRESS)
return SYSTEM.INTERRUPT_TYPE_NUMBER;

procedure GET_ADDRESS_FROM_INTERRUPT_TYPE_NUMBER
  ( A      : out SYSTEM.ADDRESS;
    ITN    : in SYSTEM.INTERRUPT_TYPE_NUMBER );
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the iAPX286 Operating in Real Address Mode (Continued)

```
function GREATER_THAN
  ( A1 : in SYSTEM.ADDRESS;
    A2 : in SYSTEM.ADDRESS )
return BOOLEAN;

function MINUS
  ( A1 : in SYSTEM.ADDRESS;
    A2 : in SYSTEM.ADDRESS )
return LONG_INTEGER;

function ">"
  ( A1 : in SYSTEM.ADDRESS;
    A2 : in SYSTEM.ADDRESS )
return BOOLEAN renames SYSTEM.GREATER_THAN;

function "-"
  ( A1 : in SYSTEM.ADDRESS;
    A2 : in SYSTEM.ADDRESS )
return LONG_INTEGER renames SYSTEM.MINUS;

procedure ADJUST_FOR_UPWARD_GROWTH
  ( OLD_ADDRESS : in SYSTEM.ADDRESS ;
    ADJUSTED_ADDRESS : out SYSTEM.ADDRESS );
-- Transforms the given SYSTEM.ADDRESS into a representation
-- yielding the same effective address, but in which the
-- SEGMENT component is as large as possible.

procedure ADJUST_FOR_DOWNWARD_GROWTH
  ( OLD_ADDRESS : in SYSTEM.ADDRESS ;
    ADJUSTED_ADDRESS : out SYSTEM.ADDRESS );
-- Transforms the given SYSTEM.ADDRESS into a representation
-- yielding the same effective address, but in which the
-- OFFSET component is as large as possible.

procedure ELAB_SYSTEM;

end SYSTEM;
```

Package STANDARD for the iAPX286 Operating in Real Address Mode

package STANDARD is

...

-147-

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Systems Designers Software, Inc. Compiler version 2B.00	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX series (under VAX/VMS 4.5 or MicroVMS 4.5)	1750A, Ferranti Computer System 100A (bare machine) * Derived *

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

See following pages.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

**System Designers Software PIWG results for 1750A (Fairchild 9450). Clock : 15MHz.
PIWG test suite 1987.**

PIWG Test Name	Description	Micro - seconds
C000001	Task creation/terminate, task type declared in package.	3150
C000002	Task creation/terminate, task type declared in procedure.	3549
C000003	Task creation/terminate, task type declared in block.	3574
D000001	Dynamic array, use and deallocation.	40
D000002	Dynamic array elaboration and initialization.	33399
D000003	Dynamic record allocation and deallocation.	4074
D000004	Dynamic record elaboration and initialization.	40600
E000001	Raise and handle an exception locally.	67
E000002	Raise and handle an exception in a package.	114
E000003	Raise and handle an exception nested 3 deep in procedures.	184
F000001	Set a BOOLEAN flag using a logical equation.	7
F000002	Set a BOOLEAN flag using an "if" test.	7
L000001	Simple "for" loop.	7
L000002	Simple "while" loop.	9
L000003	Simple "exit" loop.	6
P000001	Procedure call and return (inlineable), no parameters.	27
P000002	Procedure call and return (not inlineable), no parameters.	30
P000003	Procedure call and return (compiled separately).	27
P000004	Procedure call and return (Pragma INLINE used).	27
P000005	Procedure call and return (one parameter, in INTEGER).	30
P000006	Procedure call and return (one parameter, out INTEGER).	35
P000007	Procedure call and return (one parameter, in out INTEGER).	38
P000010	Procedure call and return (ten parameters, in INTEGER).	52
P000011	Procedure call and return (twenty parameters, in INTEGER).	75
P000012	Procedure call and return (ten parameters, in record_type).	72
P000013	Procedure call and return (twenty parameters, in record_type).	115
T000001	Minimum rendezvous, entry call and return.	687
T000002	Task entry call and return (one task, one entry).	650
T000003	Task entry call and return (two tasks, one entry each).	681
T000004	Task entry call and return (one task, two entries).	1068
T000005	Active entry and return (ten tasks, one entry each).	655
T000006	Task entry call and return (one task, ten entries).	2730

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Systems Designers Software, Inc. Compiler version 2C.00	DEC VAX 8600 (under VMS 4.5)	68010, MC68010 implemented on the MVME 117-3FP board (bare machine)

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

See following pages.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

System Designers Software PIWG results for Motorola MC68010. Clock : 10MHz, 1 wait-state. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
C000001	Task creation/terminate, task type declared in package.	4844
C000002	Task creation/terminate, task type declared in procedure.	7382
C000003	Task creation/terminate, task type declared in block.	8867
D000001	Dynamic array, use and deallocation.	18
D000002	Dynamic array elaboration and initialization.	32578
D000003	Dynamic record allocation and deallocation.	7570
D000004	Dynamic record elaboration and initialization.	47734
E000001	Raise and handle an exception locally.	48
E000002	Raise and handle an exception in a package.	78
E000003	Raise and handle an exception nested 3 deep in procedures.	119
F000001	Set a BOOLEAN flag using a logical equation.	6
F000002	Set a BOOLEAN flag using an "if" test.	7
L000001	Simple "for" loop.	8
L000002	Simple "while" loop.	9
L000003	Simple "exit" loop.	8
P000001	Procedure call and return (inlineable), no parameters.	30
P000002	Procedure call and return (not inlineable), no parameters.	34
P000003	Procedure call and return (compiled separately).	32
P000004	Procedure call and return (Pragma INLINE used).	32
P000005	Procedure call and return (one parameter, in INTEGER).	36
P000006	Procedure call and return (one parameter, out INTEGER).	39
P000007	Procedure call and return (one parameter, in out INTEGER).	42
P000010	Procedure call and return (ten parameters, in INTEGER).	61
P000011	Procedure call and return (twenty parameters, in INTEGER).	90
P000012	Procedure call and return (ten parameters, in record_type).	87
P000013	Procedure call and return (twenty parameters, in record_type).	141
T000001	Minimum rendezvous, entry call and return.	791
T000002	Task entry call and return (one task, one entry).	791
T000003	Task entry call and return (two tasks, one entry each).	815
T000004	Task entry call and return (one task, two entries).	1406
T000005	Active entry and return (ten tasks, one entry each).	777
T000006	Task entry call and return (one task, ten entries).	3469

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the System Designers MC68010 Bare Machine Target

```
package SYSTEM is

  type ADDRESS is private

  type NAME is (MC68000, MC68010);

  SYSTEM_NAME : constant NAME := MC68010;
  STORAGE_UNIT : constant      := 8;
  MEMORY_SIZE : constant       := 1677721;
  MIN_INT : constant           := -2147483648;
  MAX_INT : constant           := 2147483647;
  MAX_DIGITS : constant        := 6;
  MAX_MANTISSA : constant       := 31;
  FINE_DELTA : constant        := 2#1.0E-30;
  TICK : constant              := 2#1.0#E-7;

  subtype PRIORITY is INTEGER range 0..15;

  type UNIVERSAL_INTEGER is range MIN_INT..MAX_INT;

  subtype EXTERNAL_ADDRESS is STRING;

  subtype BYTE is INTEGER range -128..127;

  type LONG_WORD is array (0..3) of BYTE;

  pragma PACK(LONG_WORD);

  function CONVERT_ADDRESS(ADDR : EXTERNAL_ADDRESS)
    return ADDRESS;

  function CONVERT_ADDRESS(ADDR : ADDRESS)
    return EXTERNAL_ADDRESS;

  function CONVERT_ADDRESS(ADDR : LONG_WORD)
    return ADDRESS;

  function CONVERT_ADDRESS(ADDR : ADDRESS)
    return LONG_WORD;

  function "+" (ADDR : ADDRESS; OFFSET : UNIVERSAL_INTEGER)
    return ADDRESS;

  private
    -- type ADDRESS is system-dependent
end SYSTEM;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for the System Designers MC68010 Bare Machine Target

package STANDARD is

type BOOLEAN is (FALSE, TRUE);

type INTEGER is range
-2147483648..2147483647;

type FLOAT is digits 6 range
-16#FFFFFF#E32..16#0.FFFFFFF#E32;

type CHARACTER is

...

for CHARACTER use

...

package ASCII is

...

end ASCII;

-- Predefined subtypes:

subtype NATURAL is INTEGER range 0..INTEGER'LAST;

subtype POSITIVE is INTEGER range 1..INTEGER'LAST;

-- Predefined string type:

type STRING is array (POSITIVE range <>) of CHARACTER;

type DURATION is delta 2#1.0E-7 range -16777216.0..16777215.0;

-- The predefined exceptions:

...

end STANDARD;

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Systems Designers Software, Inc. Compiler version 2C.00	DEC VAX 8600 (under VMS 4.5)	68020, MC68020, implemented on the MVME 133 board with a MC68881 floating point co-processor (bare machine)
Compiler version 2C.00	DEC VAX-11/7xx VAX 8xxx, VAX station, and MicroVAX series (under VAX/VMS 4.5 or MicroVMS 4.5)	68020, MC68020, implemented on the MVME 133 board with a MC68881 floating point co-processor (bare machine)
Compiler version 3A.00	DEC VAX-11/7xx VAX 8xxx, VAX station (under VMS 4.6), MicroVAX series (under MicroVMS 4.5)	68020, MC68020, implemented on the MVME 133 board with a MC68881 floating point co-processor (bare machine)

DEGREE OF CONFIGURABILITY

The software supplied is configured for the standard hardware, but it is supplied in source form to enable users to reconfigure it for their own target hardware.

The following is a list of the common target differences and what is affected:

Target Timer

A macro definition file defines symbols and macros used to control the timer device of the SD standard MC68020 target. This file must be modified for targets using a different timer or one located elsewhere.

Target I/O

A macro definition file is used to provide symbols and macros dependent upon the location and type of I/O device used for the host/target link. As supplied, it is

Guidelines to Select, Configure and Use an Ada Runtime Environment

configured for the standard MC68020 target and may require modification for targets using a different I/O device.

Target Initialization

A macro file is provided for certain target-dependent actions that may be required to initialize the target following a "reset" or "power-up".

Context Switching

A macro file is provided which will allow the user the option of saving/not saving the entire MC68020 register set and floating point coprocessor context when context switching occurs as the result of an interrupt.

Interrupt Vectors

With the software as supplied, all unused interrupt vectors are initialized to pass control to the handler `UNEXPECTED_INTERRUPT`, which reports the interrupt and then raises `PROGRAM_ERROR`. The user can place interrupt vectors at a specific location, but a particular module must be specifically located there when the program is built. The user can also modify a module which handles unexpected interrupts as required.

Unhandled Exceptions

A catchall handler is provided for exceptions that are propagated out of an Ada program.

Deadlock

If the tasking system detects a deadlock situation, a module is called to output a deadlock message. This routine should be changed to perform the required actions.

PIWG RESULTS

See next page.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

System Designers Software PIWG results for Motorola MC68020. Clock : 12.5MHz, zero wait-states. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
C000001	Task creation/terminate, task type declared in package.	8089
C000002	Task creation/terminate, task type declared in procedure.	8125
C000003	Task creation/terminate, task type declared in block.	8125
D000001	Dynamic array, use and deallocation.	13
D000002	Dynamic array elaboration and initialization.	12500
D000003	Dynamic record allocation and deallocation.	1689
D000004	Dynamic record elaboration and initialization.	17734
E000001	Raise and handle an exception locally.	31
E000002	Raise and handle an exception in a package.	47
E000003	Raise and handle an exception nested 3 deep in procedures.	67
F000001	Set a BOOLEAN flag using a logical equation.	9
F000002	Set a BOOLEAN flag using an "if" test.	9
J000001	Simple "for" loop.	4
L000002	Simple "while" loop.	3
L000003	Simple "exit" loop.	4
P000001	Procedure call and return (inlineable), no parameters.	8.8
P000002	Procedure call and return (not inlineable), no parameters.	10.9
P000003	Procedure call and return (compiled separately).	21.9
P000004	Procedure call and return (Pragma INLINE used).	17.3
P000005	Procedure call and return (one parameter, in INTEGER).	18.3
P000006	Procedure call and return (one parameter, out INTEGER).	20.7
P000007	Procedure call and return (one parameter, in out INTEGER).	25.9
P000010	Procedure call and return (ten parameters, in INTEGER).	29.6
P000011	Procedure call and return (twenty parameters, in INTEGER).	51.2
P000012	Procedure call and return (ten parameters, in record_type).	30.5
P000013	Procedure call and return (twenty parameters, in record_type).	56.2
T000001	Minimum rendezvous, entry call and return.	191
T000002	Task entry call and return (one task, one entry).	189
T000003	Task entry call and return (two tasks, one entry each).	202
T000004	Task entry call and return (one task, two entries).	317
T000005	Active entry and return (ten tasks, one entry each).	188
T000006	Task entry call and return (one task, ten entries).	539

Guidelines to Select, Configure and Use an Ada Runtime Environment

Appendix F Notes

The following excerpts are from System Designers Ada-Plus VAX/VMS = > MC68020 compiler documentation. [12]

Restrictions on Representation Clauses

Length Clauses

Attribute SIZE

The value specified for SIZE must not be less than the minimum number of bits required to represent all values in the range of the associated type or subtype. Otherwise, a compiler restriction is reported.

Attribute SMALL

There are no restrictions for this attribute.

Attribute STORAGE_SIZE

For access types the limit is governed by the address range of the target machine and the maximum value is determined by SYSTEM.ADDRESS'LAST.

For task types the limit is also SYSTEM.ADDRESS'LAST.

Record Representation Clauses

Alignment Clause

The `static_simple_expression` used to align records onto storage unit boundaries must deliver the values 0 (bit aligned), 1 (byte aligned), 2 (word aligned) or 4 (long word aligned).

Component Clause

Non-scalar types must be aligned and sized correctly.

The component size defined by the static range must not be less than the minimum number of bits required to hold every allowable value of the component. For a component of non-scalar type, the size may not be larger than that chosen by the compiler for the type.

Address Clauses

Address clauses are implemented as assignments of the address expressions to objects of an appropriate access type.

An object being given an address is assumed to provide a means of accessing memory external to the Ada program. An object declaration with an address clause is treated by the compiler as an access object whose access type is the same as the type of the

Guidelines to Select, Configure and Use an Ada Runtime Environment

object declaration. This access object is initialized with the given address at the point of elaboration of the corresponding address clause, for example:

```
X : INTEGER;  
.  
.  
.  
for X'ADDRESS use at CONVERT_ADDRESS("FF00");  
  
is equivalent to:  
  
type X_P is access INTEGER;  
X : X_P;  
.  
.  
.  
X := new_AT_ADDRESS(X_P, "FF00");  
-- where function new_AT_ADDRESS claims no store but  
-- returns the address given.
```

Note: The expressions in an address clause for an object are interpreted as addresses absolute addresses on the target. Address clauses for subprograms, packages and tasks are not implemented.

It is the responsibility of some external agent to initialize the area at a given address. The Ada program may fail unpredictably if the storage area is initialized prior to the elaboration of the address clause. The access object can be used for reading from and writing to the memory normally, but only after the elaboration of the address clause.

Address clauses can only be given for objects and task entries. Address clauses are not supported for other entities.

Unchecked storage deallocation will not work for objects with address clauses.

Object Addresses

For objects with an address clause, a pointer is declared which points to the object at the given address. There is a restriction however that the object cannot be initialized either explicitly or implicitly (i.e. the object cannot be an access type).

Subprogram, Package and Task Unit Addresses

Address clauses for subprograms, packages and task units are not supported by this version of the compiler.

Entry Addresses

Address clauses for are supported; the address given is the address of an interrupt vector.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Example:

```
task INTERRUPT_HANDLER is
  entry DONE;
  for DONE use at SYSTEM.CONVERT_ADDRESS ("7C");
end INTERRUPT_HANDLER;
```

Note that it is only possible to define an address clause for an entry of a single task.

Implementation-Generated Names

There are no implementation-generated names denoting implementation-dependent components.

Interpretation of Expressions in Address Clauses

The expressions in an address clause are interpreted as absolute addresses on the target. Address clauses for subprograms, packages and tasks are not implemented.

Unchecked Conversions

The implementation imposes the restriction on the use of the generic function `UNCHECKED_CONVERSION` that the size of the target type must not be less than the size of the source type.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the SD Ada - MC68020

package SYSTEM is

type ADDRESS is private;

type NAME is (MC68020);

SYSTEM_NAME : constant NAME := MC68020;

STORAGE_UNIT : constant := 8;

MEMORY_SIZE : constant := 2**32;

MIN_INT : constant := -(2**31);

MAX_INT : constant := (2**31)-1;

MAX_DIGITS : constant := 15;

MAX_MANTISSA : constant := 31;

FINE_DELTA : constant := 2#1.0E-31;

TICK : constant := 2#1.0#E-7;

subtype PRIORITY is INTEGER range 0 .. 126;

type UNIVERSAL_INTEGER is range MIN_INT .. MAX_INT;

subtype EXTERNAL_ADDRESS is STRING;

function CONVERT_ADDRESS (ADDR : EXTERNAL_ADDRESS)
return ADDRESS;

function CONVERT_ADDRESS (ADDR : ADDRESS)
return EXTERNAL_ADDRESS;

function "+" (ADDR : ADDRESS;
OFFSET : UNIVERSAL_INTEGER)
return ADDRESS;

private

-- Implementation-dependent type ADDRESS

end SYSTEM;

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for the MC68020

```
package STANDARD is
  type BOOLEAN is (FALSE, TRUE);
  type SHORT_INTEGER is range
    -128 .. 127;
  type INTEGER is range
    -32_768 .. 32_767;
  type LONG_INTEGER is range
    -2_147_483_648 .. 2_147_483_647;
  type FLOAT is digits 6 range
    -16#0.FFFFFFF#e32 .. 16#0.FFFFFFF#E32;
  type LONG_FLOAT is digits 15 range
    - 16#0.FFFFFFFFF_FFFFFFF#E44 ..
      16#0.FFFFFFFFF_FFFFFFF#E44;
  type DURATION IS DELTA 2#1.0#E-7 range
    - 16_777_216.0 .. 16_777_215.0;
end STANDARD;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Tartan Laboratories Inc. Compiler version V9	VAX-11/750 (under VMS 4.1)	1750A, Fairchild F9450 1750A, Mikros MKS1750/SO 1750A, Unisys S1636- (MIL-STD-1750A) (all bare machines)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Components are linked selectively and are only included if required.

II. Customization of the Runtime:

- By the use of pragmas
- By the use Compiler Switches
- Modifying-Replacing the source to selective runtime routines provided with the purchase of the compiler (i.e. Device Drivers)
- By modifying the source to the entire runtime (after purchasing it)

Storage Layout - The main memory of a 1750A computer running an Ada program is divided into four regions: static code and data (low memory), a stack for the main program, a stack for interrupt handlers (high memory), and all remaining free storage. The exact layout of the static area is determined by the linker control file.

User-Defined Actions - This is a collection of small procedures that are invoked by the runtime when unusual conditions arise. It consists of the following:

- System Idle
- Program Termination
- Abnormal Termination Diagnostics
- Lowest Level Output
- Text I/O Routines
- Simple I/O Routines

Interrupts - The Ada runtime handles the floating-point overflow, fixed-point overflow, floating-point underflow, and timer B interrupts of the MIL-STD- 1750A. The remaining interrupts are available for application use. There are several ways in which handling of a particular interrupt may be added to the runtime:

- An assembly code handler may be used that transparently services the interrupt and returns to the point of interruption.

Guidelines to Select, Configure and Use an Ada Runtime Environment

- A handler may be added to the Ada runtime. Such a handler, following a standard template, can share the runtime interrupt stack and invoke runtime task scheduling and interrupt services. The body for such a handler may be written in Ada.
- A task entry may be connected to an interrupt.

Interrupt Vectors - There is a file which statically initialized the interrupt vectors. Users may add their own vector initialization to this file.

Transparent Interrupt Handlers - Interrupts which occur very frequently and require rapid service may be serviced by transparent interrupt handlers. Such a handler is divorced from the Ada runtime data structures. The handler is written in assembly code. Runtime tasking and interrupt services may not be called.

Standard Interrupt Handlers - A less restrictive form of interrupt handler may be constructed using the template code provided. Standard handlers share the runtime's interrupt stack and have access to runtime tasking and interrupt services. The body of a standard interrupt handler may be written in Ada as a normal procedure. The following restrictions should be observed:

- No Ada tasking operations should be done.
- Access types should not be declared nor allocations done. Doing so would cause invocation of storage manager functions with the potential for lock conflicts.
- Up-level addressing of nonstatic objects cannot be done. Interrupt service should be done by outer level routines.

Direct Connection of Task Entries - A task entry may be directly connected to some hardware interrupts by use of an address clause. The direct connection of an entry to a hardware interrupt requires the alteration of the appropriate interrupt vector by the runtime when the task is created.

The following excerpts are from Tartan's User Manual for the Runtime Client Package.
[27]

Tartan provides a Client package which is a "sideways" interface to the Ada runtime's tasking mechanism. The Client package allows the Ada programmer to access the tasking data structures and operations that are used to implement the Ada language requirements. The access is directly into the runtime support, therefore providing considerably greater power and generality than is available from the ordinary language operations. It also places greater responsibility upon the user to insure that these operations are used correctly.

In general, the following capabilities are provided by the Client package:

- It allows the user to examine and modify the control block associated with each task, or to associate additional user-defined data structures with a given task.
- It allows the user to define nested global critical sections within which all context switching is disabled.
- It allows the user to suspend, resume, delay, abort, or force exceptions into arbitrary tasks.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Use of these capabilities in combination allows the user to implement a variety of executive control functions in a multiple task application.

Runtime Tasking Data Structures (which the user has access to):

- Task Control Block. This is a data structure that is created when the corresponding task is elaborated.
- Major States of a Task - Not Used, Await Dependents Termination, Caller Await Rendezvous, Caller In Rendezvous, Child Activating, Delayed, Executing, Finished, I/O Wait, Not Started, Waiting To Accept.
- TCB Extension. This field may be used to store whatever scalar or pointer information the particular application wishes.
- Priorities of the Task. - static and dynamic priority fields
- Parent TCB. This field contains the TCB of the task on which this task is dependent.
- Exception ID. This field contains the exception code for an exception that is to be raised in a currently non-executing task.
- Execution Context.

Critical Sections

A critical section is a sequence of code in which context switches (such as task switching or interrupts) must be prohibited due to the nature of the operations being performed. Usually, this is done to prohibit corruption of some data structure by "simultaneous" access by multiple tasks. The beginning of a critical section is indicated by a call to the procedure `Enter_Critical_Section`; the end of the critical section is indicated by a call to the procedure `Leave_Critical_Section`.

Critical sections may be nested. The runtime software has the notion of a critical section level global to all executing tasks on a single processor.

Tasking Interactions

The Client package supplies a number of procedures that allow the user to affect the scheduling of tasks.

Scheduling Policy

The runtime contains a replaceable "scheduling" module. The default is a preemptive, priority-based algorithm that uses order of arrival to break ties. Variants of this module may use other criteria such as round-robin, fairness, or cyclic time-slicing.

Context Switch

The context switch function performs the actual swap of processor state. In essence, it implements the decision of the scheduling policy module.

Tasking Control Procedures and Functions

The following are individual procedures related to tasking control:

Guidelines to Select, Configure and Use an Ada Runtime Environment

- **Reschedule.** This procedure causes the scheduler to reexamine the set of runnable tasks and dispatch one based on the scheduling policy currently in effect. Usually the task with the highest priority is dispatched.

The current task will be treated like any other runnable task by the scheduler unless the task has been suspended by procedure `Suspend`. If and when the current task resumes execution, control returns to the task immediately after the call to `Reschedule`.

- **Suspend.** This procedure causes the task T to be removed from the set of runnable tasks. The effect of `Suspend` is undone by a call to `Resume`. Calling `Suspend` does not cause a context switch; the current task continues to execute until such time as a reschedule is done explicitly or implicitly.

- **Suspend and Reschedule.** This procedure suspends the task T and can cause a context switch to occur.

- **Suspend Current Task and Reschedule.** This procedure suspends the current task and causes a context switch to occur.

- **Suspend Current Task.** This procedure suspends the current task.

Execution of the current task continues until a context switch occurs, which will happen if `Reschedule` is called but may happen asynchronously if an interrupt occurs. Therefore if this procedure is used in preference to `SuspendMe_and_Reschedule`, it should probably be called only within a critical section.

- **Resume and Reschedule.** This procedure cancels the effect of a previous `Suspend` on task T; that is, makes task T runnable. In addition, the scheduler makes a new selection of a runnable task, a selection in which T will participate. If task T has higher priority than the current task, and if the scheduler is implementing a priority-driven policy, the current task is preempted.

- **Resume.** This procedure cancels the effect of a previous `Suspend` on task T; that is, it makes task T runnable.

Unlike procedure `Resume_and_Reschedule`, procedure `Resume` does not cause a context switch and therefore the current task will continue to run until the scheduler regains control via a call to `Reschedule` or through the receipt of an external interrupt. If this procedure is called in preference to `Resume_and_Reschedule`, it should either be called from within a critical section or the caller should know that this task will not be preempted (e.g., because it has a higher priority than any other task).

- **Resume after Delay.** This procedure cancels the effect of `Suspend` on task T, if any, and places it on the delay queue as if a delay statement had been executed by the task. No context switch is made in conjunction with this procedure unless task T is the current task.

It is possible that task T is the current task, in which case the effect is exactly as if a delay statement had been executed.

Guidelines to Select, Configure and Use an Ada Runtime Environment

- **Set Priority.** This procedure alters the priority of the specified task. This interface is procedural because a scheduler policy decision must be made based upon the new priorities. However, no context switch is performed. In the absence of interrupts, return will always be to the calling task.
- **Set Priority and Reschedule.** This procedure acts like `Set_Priority` except that a context switch may occur after T's priority is changed.
- **Abort Task.** This procedure aborts task T as if an Ada abort statement had been executed on it.
- **Set Exception.** This procedure posts an exception in the indicated task. Any previously posted exception is superseded. The exception will be raised when the task reaches a synchronization point, that is, the next time it is dispatched.
- **Identify Current Task.** This function returns the TCB (task control block) of the current task.
- **Entry Caller for a Task.** This function returns the TCB for the task engaged in a rendezvous with the current task. It should be called only from within an accept body. If nested rendezvous are in progress, the task corresponding to the innermost rendezvous is returned.
- **Runnable.** This function returns TRUE if the task T can be executed immediately, that is, if it is in the scheduler's "ready queue". A suspended task is *not* runnable. To avoid a possible race condition, call this function only from within a critical section; otherwise the status of task T could change unpredictably.
- **Suspended.** This function returns TRUE if task T has been suspended. To avoid a race condition, call this function only from within a critical section.

The following excerpts are from Tartan's User Manual for the Expanded Memory Package.
[28]

The following modules may be customized to a particular application.

User-Defined Actions

This is a collection of small procedures that are invoked by the runtime when unusual conditions arise.

- System Idle
- Program Termination
- Abnormal Termination Diagnostics
- Lowest Level Output
- Text I/O Routines
- Simple I/O Routines

Interrupts

Guidelines to Select, Configure and Use an Ada Runtime Environment

There are several ways in which handling of a particular interrupt may be added to the runtimes:

- An assembly code handler may be used that transparently services the interrupt and returns to the point of interruption.
- A handler may be added to the Ada runtime. Such a handler, following a standard template, can share the runtime interrupt stack and invoke runtime task scheduling and interrupt services. The body for such a handler may be written in Ada.
- A task entry may be connected to an interrupt.

Interrupt Vectors

Files are used to statically initialize the interrupt vectors by the runtime. Users can add their own vector initialization to this file.

Transparent Interrupt Handlers

Interrupts that occur very frequently and require rapid services may be serviced by transparent interrupt handlers. Since such a handler is divorced from the Ada runtime data structures, entry and exit code can be kept to a minimum. The handler is written in assembly code in the usual manner except that it must obey the following restrictions:

- During interrupt service, certain interrupts should be masked (i.e. `PREEMPTER_MASK` in `EXDEFS.ASM`).
- Return from interrupt must be to the point of interruption.
- Runtime tasking, interrupt, and storage allocation services may not be called.

Standard Interrupt Handlers

A less restrictive form of interrupt handler may be constructed using the template code. Standard handlers share the runtime's interrupt stack and have access to runtime tasking and interrupt services.

The body of a standard interrupt handler may be written in Ada as a normal procedure. `Pragma Linkage_Name` can be used to provide a tractable name for the entry of the procedure. The compiled procedure is then called from the interrupt handler stub. When Ada procedures are used in such a manner, the following restrictions should be observed:

- No Ada tasking operations should be performed. The tasking control operations in the package `ARTClient` may be used.
- Access types should not be declared nor allocations done. Doing so would cause invocation of storage manager functions with the potential for lock conflicts.
- Up-level addressing of nonstatic objects cannot be done. Interrupt service should be done by outer-level routines.

Direct Connection of Task Entries

Guidelines to Select, Configure and Use an Ada Runtime Environment

A task entry may be directly connected to some hardware interrupts by use of an address clause.

The direct connection of an entry to a hardware interrupt requires the alteration of the appropriate interrupt vector by the runtimes when the task is created.

Software Interrupts

The user may cause an interrupt entry call to occur to an associated task entry by use of the DoInterrupt runtime routine.

III. Documentation provided to help user configure runtime:

- TADA (Tartan Ada Manual)
- Runtime Implementers Guide (with source code purchase)

IV. Services to customize the runtime:

- Provided by Tartan
- Cost: To be negotiated, approximately \$100.00/Hour.

V. Cost of runtime source code:

- \$50,000
- Expanded memory option: Between \$3,000.00 - \$10,000.00 Depending on host.

VI. Source of Information: Vendor input and compiler manuals.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Tartan Laboratories Inc. PIWG results for 1750A, Fairchild.
Clock : 16MHz, zero wait-states. PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks.	662.0
A000093	Whetstone benchmarks.	482*
C000001	Task creation/terminate, task type declared in package.	1106.5
C000002	Task creation/terminate, task type declared in procedure.	1106.5
C000003	Task creation/terminate, task type declared in block.	1091.8
D000001	Dynamic array, use and deallocation.	14.0
D000002	Dynamic array elaboration and initialization.	7297.7
D000003	Dynamic record allocation and deallocation.	25.1
D000004	Dynamic record elaboration and initialization.	8534.8
E000001	Raise and handle an exception locally.	11.6
E000002	Raise and handle an exception in a package.	31.1
E000003	Raise and handle an exception nested 3 deep in procedures.	18.5
E000004	Raise and handle an exception nested four deep.	18.5
E000005	Raise and handle an exception in a rendezvous.	120.4
F000001	Set a BOOLEAN flag using a logical equation.	2.1
F000002	Set a BOOLEAN flag using an "if" test.	2.5
G000005	TEXT_IO.Get an INTEGER from a local string.	340.3
G000006	TEXT_IO.Get a FLOAT from a local string.	950.1
H000001	BOOLEAN operations on entire PACKed array.	23.2
H000002	BOOLEAN operations on entire array (not packed).	573.0
H000003	BOOLEAN operations on components of a PACKed array.	1178.5
H000004	BOOLEAN operations on components of a array (not packed).	295.1
H000005	Move INTEGER to INTEGER (Unchecked_Conversion).	0.0
H000006	Move array of 10 Floats to record (Unchecked_Conversion).	21.9
H000007	Store and extract bit fields, defined by representation clauses.	80.8
L000001	Simple "for" loop.	2.5
L000002	Simple "while" loop.	2.6
L000003	Simple "exit" loop.	2.5
L000004	Loop of 5 iterations with pragma OPTIMIZE (Time).	2.2
L000005	Loop of 5 iterations with pragma OPTIMIZE (Space).	2.2
P000001	Procedure call and return (inlineable), no parameters.	0.0
P000002	Procedure call and return (not inlineable), no parameters.	4.0
P000003	Procedure call and return (compiled separately).	9.2
P000004	Procedure call and return (Pragma INLINE used).	9.2

Guidelines to Select, Configure and Use an Ada Runtime Environment

Tartan Laboratories Inc. PIWG results for 1750A, Fairchild (Continued). Clock : 16MHz, zero wait-states. PIWG test suite 1988.

PIWG Test Name	Description	Micro - seconds
P000005	Procedure call and return (one parameter, in INTEGER).	10.7
P000006	Procedure call and return (one parameter, out INTEGER).	11.5
P000007	Procedure call and return (one parameter, in out INTEGER).	12.1
P000010	Procedure call and return (ten parameters, in INTEGER).	22.0
P000011	Procedure call and return (twenty parameters, in INTEGER).	36.0
P000012	Procedure call and return (ten parameters, in record_type).	33.3
P000013	Procedure call and return (twenty parameters, in record_type).	53.3
T000001	Minimum rendezvous, entry call and return.	150.5
T000002	Task entry call and return (one task, one entry).	149.2
T000003	Task entry call and return (two tasks, one entry each).	159.3
T000004	Task entry call and return (one task, two entries).	371.8
T000005	Active entry and return (ten tasks, one entry each).	146.9
T000006	Task entry call and return (one task, ten entries).	1022.9
T000007	Minimum rendezvous, entry call and return	150.5
T000008	Parameter pass from producer task through buffer task to consumer task	726.7

* WHETSTONE : units are in KWIPS not in microseconds.

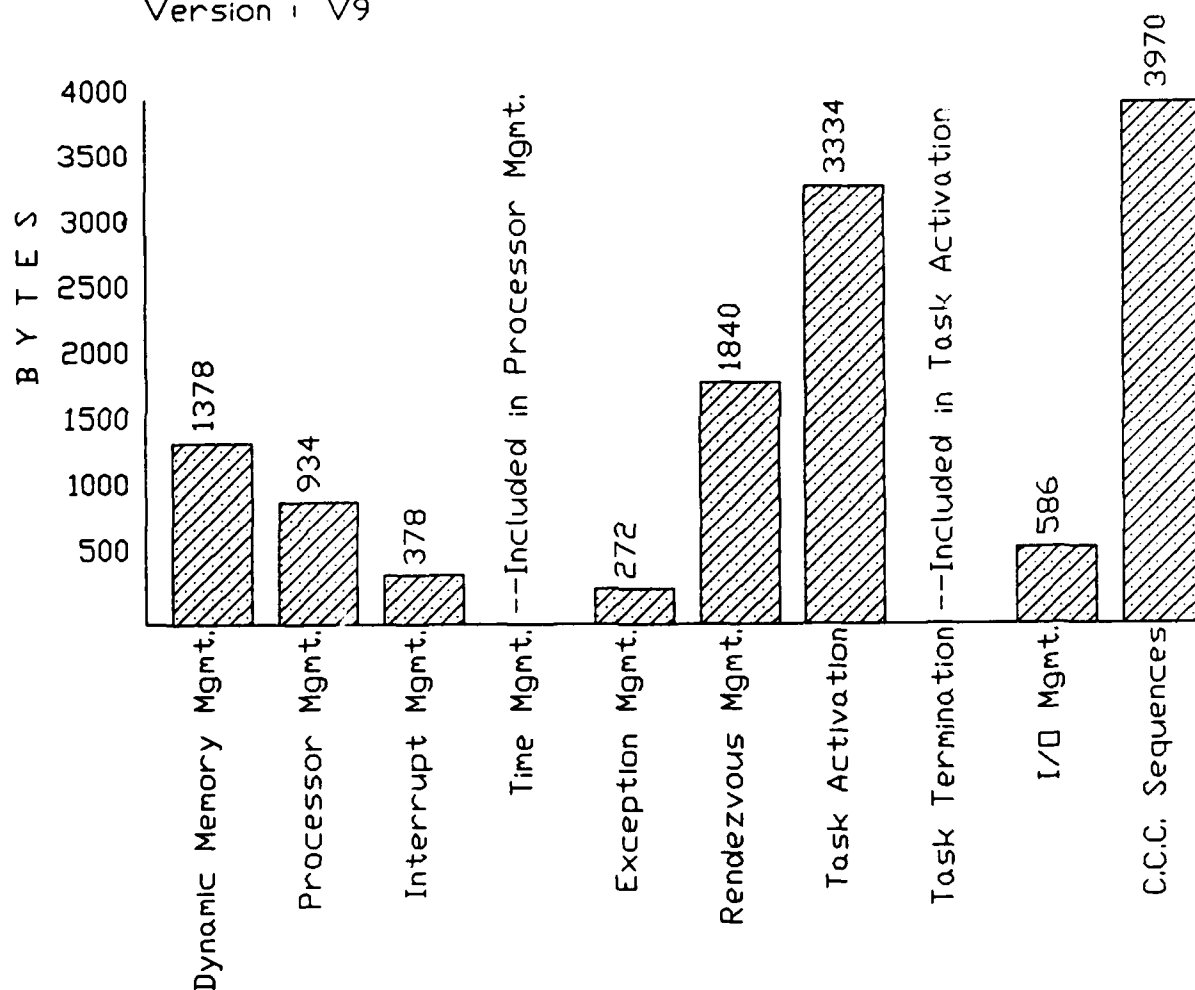
Guideline to Select, Configure, and Use an Ada Runtime Environment

Tartan Laboratories Incorporated

Host : VAX-Unix

Target : 1750A Mikros, Fairchild, or Unisys

Version : V9



- Sum of ALL Components = 12,692 bytes

Guidelines to Select, Configure and Use an Ada Runtime Environment

Appendix F Notes

The following excerpts are from User Manual for Tartan Ada VMX/1750A. [11]

Restrictions on Representation Clauses

Length Clauses

A length clause to T'SIZE is permitted for any type or first subtype T for which the size can be computed at compile time. A length clause for a composite type cannot be used to force a smaller size for components than established by the default the mapping or by length clauses for the component types.

There are no restrictions on other forms of length clauses except the restrictions specified in LRM 13.2. The size specified for T'SORAGE_SIZE of an access type or task type T is assumed to include a small amount of hidden administrative storage.

Enumeration Representation Clauses

All integer codes in the representation aggregate must be between INTEGER'FIRST and INTEGER'LAST.

Record Representation Clauses

Record representation clauses are permitted only for record types all of whose components have a size known at compile time.

Representation specifications may be specified for some components of a record without supplying representation specifications for all components. The compiler will freely allocate the unspecified components.

Address Clauses

When applied to an object, an address clause becomes a linker directive to allocate the object at the given logical address. For any object not declared immediately within top-level library package, the address clause is meaningless, with the possible exception of objects declared inside a task, if the target permits a task to run in a separate address space.

Address clauses applied to local packages are not supported by Tartan Ada.

Address clauses applied to subprograms and tasks are implemented according to the LRM rules.

When applied to an entry, the specified value identifies an interrupt in a manner customary for the target. Immediately after a task is created, a runtime call is made for each of its entries having an address clause, establishing the proper binding between the entry and the interrupt.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the MIL-STD-1750a

package SYSTEM is

type ADDRESS is new integer;;

type NAME is (VAX, MIL_STD_1750A, MC68000);

system_name : constant name := MIL_STD_1750A;

storage_unit : constant := 16;

memory_size : constant := 1000000;

min_int : constant := -max_int - 1;

max_int : constant := 32767;

max_digits : constant := 9;

max_mantissa : constant := 31;

fine_delta : constant := 2#1.0E-14;

tick : constant := 0.0001;

subtype priority is integer range 10 .. 200;

default_priority : constant priority := priority'first;

runtime_error : exception;

end SYSTEM;

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
TeleSoft, Inc. Compiler version 3.22	MicroVAX II (under VMS, version 4.6)	1750A, MIL-STD-1750A ECSP0 RAID simulator version 4.0 executing on the host (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

This information was not supplied by the vendor.

II. Customization of the Runtime:

- By compiler switches
- By linker switches

III. Documentation provided to help user configure runtime:

- TeleGen2 VAX/1750A Users Guide

IV. Services to customize the runtime:

- TeleSoft has a custom products division geared toward assisting the customer, customizing the runtime and/or compiler.

V. Cost of runtime source code:

- \$50,000

VI. Source of Information: Vendor input, selected compiler documentation.

Guidelines to Select, Configure and Use an Ada Runtime Environment

TeleSOFT Inc. PIWG results for Fairchild 1750A. Clock : 15MHz, zero wait-states, (Tests were compiled with OPTIMIZE pragma). PIWG test suite 1986.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks.	1050.2
A000093	Whetstone benchmarks.	450*
C000001	Task creation/terminate, task type declared in package.	2961.1
C000002	Task creation/terminate, task type declared in procedure.	2965.7
C000003	Task creation/terminate, task type declared in block.	2930.9
D000001	Dynamic array, use and deallocation.	16.7
D000002	Dynamic array elaboration and initialization.	23576.7
D000003	Dynamic record allocation and deallocation.	313.5
D000004	Dynamic record elaboration and initialization.	24776.0
E000001	Raise and handle an exception locally.	9.1
L000001	Simple "for" loop.	5.6
L000002	Simple "while" loop.	5.5
L000003	Simple "exit" loop.	4.9
P000001	Procedure call and return (inlineable), no parameters.	0.0
P000002	Procedure call and return (not inlineable), no parameters.	35.8
P000003	Procedure call and return (compiled separately).	34.7
P000004	Procedure call and return (Pragma INLINE used).	0.0
P000005	Procedure call and return (one parameter, in INTEGER).	34.7
P000006	Procedure call and return (one parameter, out INTEGER).	35.4
P000007	Procedure call and return (one parameter, in out INTEGER).	35.4
P000010	Procedure call and return (ten parameters, in INTEGER).	37.7
P000011	Procedure call and return (twenty parameters, in INTEGER).	54.1
P000012	Procedure call and return (ten parameters, in record_type).	44.7
P000013	Procedure call and return (twenty parameters, in record_type).	71.2
T000001	Minimum rendezvous, entry call and return.	961.4
T000002	Task entry call and return (one task, one entry).	959.9
T000003	Task entry call and return (two tasks, one entry each).	978.9
T000004	Task entry call and return (one task, two entries).	1177.5
T000006	Task entry call and return (one task, ten entries).	1770.7
T000007	Minimum rendezvous, entry call and return.	617.9

* WHETSTONE : units are in KWIPS not in microseconds.

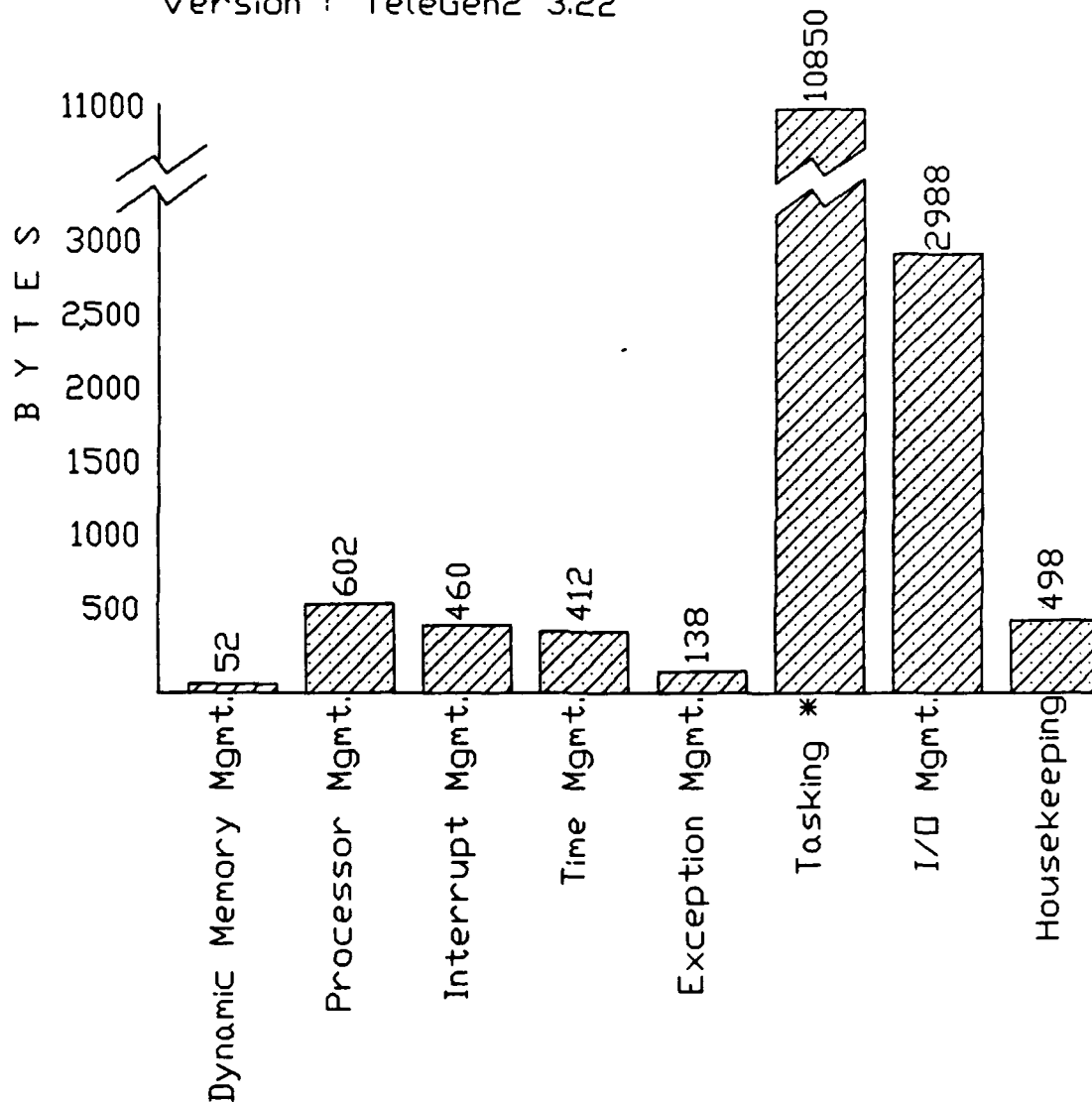
Guidelines to Select, Configure and Use an Ada Runtime Environment

TeleSoft, Inc.

Host : VAX - VMS

Target : 1750A

Version : TeleGen2 3.22



- Sum of ALL Components = 16,000 bytes

- * Tasking includes :
1. Rendezvous Management
 2. Task Activation
 3. Task Termination

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: The 1750A uses 10KHz timer (Timer B).

Q2: How long, and for what reasons are interrupts disabled?

A2: Interrupts are disabled during runtime structure update.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: (1) Function mapped interrupts - All processing associated with handling the interrupt occurs during the rendezvous (in the body of the accept statement) and no interactions with other tasks occur during the rendezvous. (2) Simple rendezvous.

Q4: What are the restrictions for representation clauses?

A4: For the 1750A (the following are excerpts from TeleGen2 User Guide documentation [25]):

(LRM 13.1) This release supports a limited use of pragma Pack.

(LRM 13.2) Telegen2 allows user specification of storage for a task activation using the Storage_Size attribute in a length clause. The default stack size is 768 words.

(LRM 13.5) For address clauses applied to objects, a simple expression of type Address is interpreted as a position within the linear address space of the 1750A. Unchecked_Conversion to the private type System.Address must be used to specify address constants.

(LRM 13.5.1) For interrupt entries, the address of a TeleSoft-defined interrupt descriptor can be given. Address clauses for subprograms, packages, tasks, and literal constants are not supported.

(LRM 13.6) Changes of representation are not supported for types with record representation clauses.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Event driven (interrupts included) and Run-until-blocked or pre-empted.

Q6: What are the restrictions on pragma INLINE?

A6: The following are excerpts from TeleGen2 User Guide documentation. [25]

The optimizer supports inlining of calls to subprograms that the user identifies through pragma INLINE. As specified by the Ada language reference manual, the pragma must be placed in the same declarative region as the declaration of the subprogram to be inlined and must follow the subprogram declaration. In the following example the package Drag_Coef is placed after the declaration of the function:

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package Drag_Calc is

type Plane_Type is (B707,B727,B747);

function Drag_Coef (Plane: Plane_Type) return float;
Pragma INLINE (Drag_Coef);

end Drag_Calc;

The inlining of subprograms is transitive. For example, if inlined subprogram A is called by inlined subprogram B, and B is called by subprogram C, then optimization will result in A being inlined in B and then B being inlined in C.

The one exception to the rule is that a subprogram will not be automatically inlined into a subprogram that itself marked for inlining using pragma INLINE. Autoinlining is inhibited to ensure that the user has full control over the inlining process. This feature prevents any significant and undesired overhead introduced by the automatic inlining of a called subprogram. Any subprogram that is to be inlined into another inlined subprogram must be explicitly marked with Pragma INLINE.

Inline expansion is the one type of optimization currently implemented for which a time/space tradeoff is an issue. A subprogram that the user has marked for inline expansion and that is called from more than one place can cause object code to be larger after optimization than before, if the inlined subprogram has significant size. Subprograms identified for inlining should be small enough so that expansion takes little if any additional space than the call it replaces. Inline subprogram designations will be honored regardless of the space the code uses. It is the users responsibility to evaluate potential tradeoffs.

The following conditions must be met for a subprogram to be inlined:

1. The subprogram must be designated in an INLINE pragma or be subject to automatic inlining.
2. The unit containing the subprogram to be inlined must be optimized.
3. The units that call the inlined subprogram must be optimized.

Conditions two and three indicate that both the called subprogram and the code that calls it must be optimized for inlining to take place.

4. Full intermediate code forms of the unit containing the subprogram to be inlined must be present in the Ada library.

The Optimizer works on the intermediate code forms of compiled units. The compiler,/NOSQUEEZE qualifier must be used to ensure that these forms are stored in the Ada library when compilation is complete.

5. A unit that contains the body of an inlined subprogram must be compiled before the compilation of any units that call the inlined subprogram.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Inlining consists of inserting the code for the inlined subprogram into the calling program. The code for the inlined subprogram must already exist for the insertion to occur.

If any of these conditions are not met, inlining will not take place and a normal call to a non-inlined copy of the subprogram will take place.

Inlining may create new unit dependencies. The user needs to anticipate the consequences of inlining certain subprograms with a given configuration.

Due to the possibility that a caller has been compiled prior to compilation and optimization of an inlined body, the optimizer will always create a callable body for a `pragma INLINE` program that is externally visible. Generation of a callable body can be avoided by declaring the subprogram where it is not externally visible (i.e. in the body of a package) or by designating the unit a hidden unit of a collection that includes all of the subprogram's callers.

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: Yes.

Q9: What object types are supported by `pragma SHARED`?

A9: None.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

- Dynamic Task Priority
- Timer Resolution
- Exception Trace
- Default Stack Sizes
- Fast Interrupt Entry
- Optional Numeric Co-processor

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the TeleSoft 1750A Target

The current specification of package System is provided below. Note that the named number Tick is not used by any component of the Ada compiler or runtime system. Similarly, Memory_Size is not used.

package SYSTEM is

```
type Address is private;
Null_Address : constant Address;

type Physical_Address is private;

type Subprogram_Value is private;

type Name is (TeleGen2);
System_Name : constant Name := TeleGen2;

Storage_Unit : constant := 16;
Memory_Size : constant := 65536;

Min_Int : constant := - (2**31);
Max_Int : constant := (2 ** 31) - 1;
Max_Digits : constant := 6;
Max_Mantissa : constant := 31;
Fine_Delta : constant := 1.0 / (2 ** (Max_Mantissa - 1));
Tick : constant := 0.0001;

subtype Priority is Integer range 0..15;

Max_Object_Size : constant := Max_Int;
Max_Record_Count : constant := Max_Int;
Max_Text_Io_Count : constant := Max_Int-1;
Max_Text_Io_Field : constant := 1000;
```

private

```
type Address is Access Integer;
Null_Address : constant Address := null;

type Physical_Address is range 16#0#..16#7FFFFFFF#;

type Subprogram_Value is record
  Logical_Address : Target_Logical_Address;
  Address_State : Target_Address_State;
  Static_Base : Target_Logical_Address;
end record;
```

end SYSTEM;

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
TeleSoft, Inc. Compiler version 3.22	DEC VAX family (MicroVAX VAX station, VAX server, VAX 8xxx, models) (under VMS 4.5 and 4.6)	68020, MC68020 implemented on a Motorola MVME 133A-20 board with a MC68881 floating-point coprocessor (bare machine)
Compiler version 3.22	MicroVAX II (under VMS 4.6)	68020, MC68020 implemented on a Motorola MVME 133A-20 board with a MC68881 floating-point coprocessor (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

This information was not supplied by the vendor.

II. Customization of the Runtime:

- By compiler switches
- By linker switches

III. Documentation provided to help user configure runtime:

- TeleGen2 VAX/68K Users Guide

IV. Services to customize the runtime:

- TeleSoft has a custom products division geared toward assisting the customer, customizing the runtime and/or compiler.

V. Cost of runtime source code:

-\$50,000

VI. Source of Information: Vendor input, user input, selected compiler documentation.

Guidelines to Select, Configure and Use an Ada Runtime Environment

TeleSoft Inc. PIWG results for Motorola MC68020/68881. Clock : 20MHz, one wait-state, cache enabled, global optimization used, no suppresses, 68020 code generation option selected. PIWG test suite 1986.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks.	320.0
A000092	Whetstone benchmarks*.	769*
C000001	Task creation/terminate, task type declared in package.	846.7
C000002	Task creation/terminate, task type declared in procedure.	860.3
C000003	Task creation/terminate, task type declared in block.	849.6
D000001	Dynamic array, use and deallocation.	10.5
D000002	Dynamic array elaboration and initialization.	5632.7
D000003	Dynamic record allocation and deallocation.	910.2
D000004	Dynamic record elaboration and initialization.	7890.5
E000001	Raise and handle an exception locally.	16.6
E000002	Raise and handle an exception in a package.	64.2
E000004	Raise and handle an exception nested 4 deep in procedures.	424.3
L000001	Simple "for" loop.	1.1
L000002	Simple "while" loop.	0.1
L000003	Simple "exit" loop.	0.0
P000001	Procedure call and return (inlineable), no parameters.	3.8
P000002	Procedure call and return (not inlineable), no parameters.	5.6
P000003	Procedure call and return (compiled separately).	8.2
P000004	Procedure call and return (Pragma INLINE used).	3.1
P000005	Procedure call and return (one parameter, in INTEGER).	11.0
P000006	Procedure call and return (one parameter, out INTEGER).	5.2
P000007	Procedure call and return (one parameter, in out INTEGER).	4.0
P000010	Procedure call and return (ten parameters, in INTEGER).	12.7
P000011	Procedure call and return (twenty parameters, in INTEGER).	24.5
P000012	Procedure call and return (ten parameters, in record_type).	16.8
P000013	Procedure call and return (twenty parameters, in record_type).	23.9
T000001	Minimum rendezvous, entry call and return.	279.3
T000002	Task entry call and return (one task, one entry).	281.3
T000003	Task entry call and return (two tasks, one entry each).	283.9
T000004	Task entry call and return (one task, two entries).	439.9
T000005	Active entry and return (ten tasks, one entry each).	277.7
T000006	Task entry call and return (one task, ten entries).	843.7
T000007	Minimum rendezvous, entry call and return.	187.5

* This version of the WHETSTONE uses manufacturers' math routines. WHETSTONE : units are in KWIPS not in microseconds.

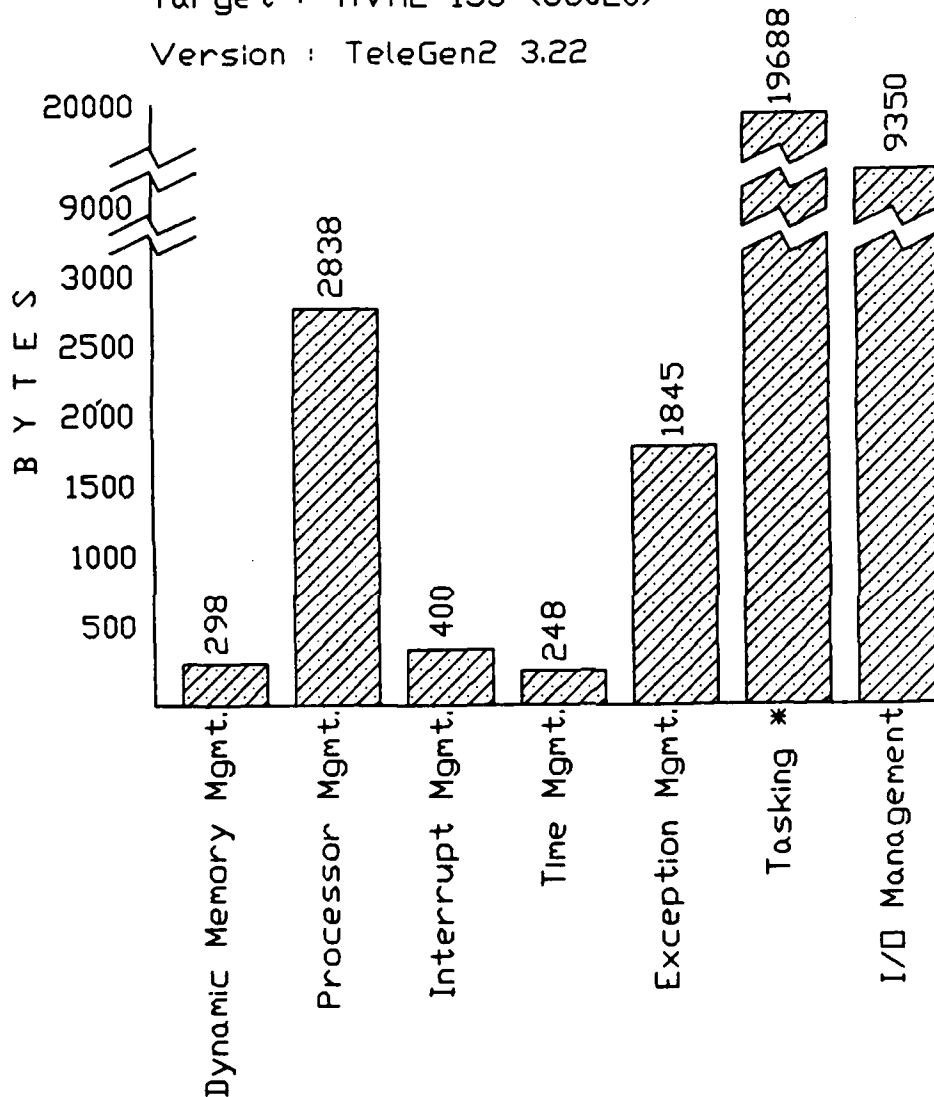
Guidelines to Select, Configure and Use an Ada Runtime Environment

TeleSoft, Inc.

Host : VAX - VMS

Target : MVME 133 (68020)

Version : TeleGen2 3.22



- Sum of ALL Components = 34,667 bytes

* Tasking includes :

1. Rendezvous Management
2. Task Activation
3. Task Termination
4. Housekeeping (Includes unchecked conversion)

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: For the MC68000 it is hardware dependent.

Q2: How long, and for what reasons are interrupts disabled?

A2: Interrupts are disabled during runtime structure update.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: (1) Function mapped interrupts - All processing associated with handling the interrupt occurs during the rendezvous (in the body of the accept statement) and no interactions with other tasks occur during the rendezvous. (2) Simple rendezvous.

Q4: What are the restrictions for representation clauses?

A4: For the MC68000 (the following are excerpts from TeleGen2 User Guide [26]):

(LRM 13.1) Records that are packed using pragma PACK follow these conventions:

1. The allocated size of each component is always a power of two (1,2,4...).
2. Components of records may cross word boundaries,
3. Components that are composite types (arrays and records) are always allocated on a System.Storage_Unit (8-bit or word) boundary.

(LRM 13.2) TeleGen2 allows user specification of storage for a task activation by use of the 'STORAGE_SIZE attribute in a length clause. The default stack size is 5000 storage units (bytes). 'STORAGE_SIZE is not supported for collections.

(LRM 13.3) Enumeration representation clauses on BOOLEAN types are not supported.

(LRM 13.4) Record representation clauses are supported, within the following constraints :

1. Each component of the record must be specified with a component clause.
2. The alignment of the record is restricted to mod 2, word alignment.
3. The ordering of bits within a byte is right to left.
4. Components may cross word boundaries.
5. Any object of a discrete type of size larger than 8 bits requires a sign bit. In the example below, the type Actually_11_bits appears to be representable in ten bits:

```
type Actually_11_bits is new Integer range 0..2**10-1;
```

```
type Small_rec is record  
  Isit_10_Bits : Actually_11_bits;  
end record;
```

```
for Small_Rec use record at mod 2;  
  Isit_10_Bits : at 0 range 0..9; --error! Invalid size.  
end record;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

Since `Actually_11_bits` are used because of the sign bit, the component clause in the example is illegal.

There are no implementation-dependent names to denote implementation-dependent components.

(LRM 13.5) Address clauses for subprograms, packages, and tasks are not supported. For address clauses applied to objects, a simple expression of type `Address` is interpreted as a position within the linear address space of the MC680x0. `Unchecked_Conversion` to the private type `System.Address` must be used to specify address constants.

(LRM 13.5.1) For interrupt entries, the address of a TeleSoft-defined interrupt descriptor can be given.

(LRM 13.6) Changes of representation are not supported for types with record representation clauses.

(LRM 13.7) Pragmas `System_Name`, `Storage_Unit`, and `Memory_Size` are not supported.

(LRM 13.7.2) `'Address` is not supported for packages or labels.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Event driven (interrupts included) and Run-until-blocked or pre-empted.

Q6: What are the restrictions on `pragma INLINE`?

A6: (The following are excerpts from TeleGen2 User Guide [26])

The optimizer supports inlining of calls to subprograms that the user identifies through `pragma INLINE`. As specified by the Ada language reference manual, the pragma must be placed in the same declarative region as the declaration of the subprogram to be inlined and must follow the subprogram declaration. In the following example the package `Drag_Coef` is placed after the declaration of the function:

Package `Drag_Calc` is

type `Plane_Type` is (B707,B727,B747);

function `Drag_Coef` (`Plane`: `Plane_Type`) return float;
`Pragma INLINE` (`Drag_Coef`);

end `Drag_Calc`;

The inlining of subprograms is transitive. For example, if inlined subprogram A is called by inlined subprogram B, and B is called by subprogram C, then optimization will result in A being inlined in B and then B being inlined in C.

The one exception to the rule is that a subprogram will not be automatically inlined into a subprogram that itself marked for inlining using `pragma INLINE`. Autoinlining is inhibited

Guidelines to Select, Configure and Use an Ada Runtime Environment

to ensure that the user has full control over the inlining process. This feature prevents any significant and undesired overhead introduced by the automatic inlining of a called subprogram. Any subprogram that is to be inlined into another inlined subprogram must be explicitly marked with Pragma `INLINE`.

Inline expansion is the one type of optimization currently implemented for which a time/space tradeoff is an issue. A subprogram that the user has marked for inline expansion and that is called from more than one place can cause object code to be larger after optimization than before, if the inlined subprogram has significant size. Subprograms identified for inlining should be small enough that expansion takes little if any additional space than the call it replaces. Inline subprogram designations will be honored regardless of the space the code uses. It is the users responsibility to evaluate potential tradeoffs.

The following conditions must be met for a subprogram to be inlined:

1. The subprogram must be designated in an `INLINE` pragma or be subject to automatic inlining.
2. The unit containing the subprogram to be inlined must be optimized.
3. The units that call the inlined subprogram must be optimized.

Conditions two and three indicate that both the called subprogram and the code that calls it must be optimized for inlining to take place.

4. Full intermediate code forms of the unit containing the subprogram to be inlined must be present in the Ada library.

The Optimizer works on the intermediate code forms of compiled units. The compiler/`NOSQUEEZE` qualifier must be used to ensure that these forms are stored in the Ada library when compilation is complete.

5. A unit that contains the body of an inlined subprogram must be compiled before the compilation of any units that call the inlined subprogram.

Inlining consists of inserting the code for the inlined subprogram into the calling program. The code for the inlined subprogram must already exist for the insertion to occur.

If any of these conditions are not met, inlining will not take place and a normal call to a non-inlined copy of the subprogram will take place.

Inlining may create new unit dependencies. The user needs to anticipate the consequences of inlining certain subprograms with a given configuration.

Due to the possibility that a caller has been compiled prior to compilation and optimization of an inlined body, the optimizer will always create a callable body for a pragma `INLINE` program that is externally visible. Generation of a callable body can be avoided by declaring the subprogram where it is not externally visible (i.e. in the body of a package) or by designating the unit a hidden unit of a collection that includes all of the subprogram's callers.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: Yes.

Q9: What object types are supported by `pragma SHARED`?

A9: None.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

- Dynamic Task Priority
- Timer Resolution
- Exception Trace
- Default Stack Sizes
- Fast Interrupt Entry
- Optional Numeric Co-processor

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the TeleSoft Embedded MC680X0 Targets

The current specification of package System is provided below. Note that the named number Tick is not used by any component of the Ada compiler or runtime system. Similarly, Memory_Size is not used.

package SYSTEM is

type Address is access integer;

type Name is (TeleGen2);

System_Name : constant Name := TeleGen2;

Storage_Unit : constant := 8;

Memory_Size : constant := (2**31) - 1;

-- System-Dependent Named Numbers:

Min_Int : constant := - (2**31);

Max_Int : constant := (2**31) - 1;

Max_Digits : constant := 15;

Max_Mantissa : constant := 31;

Fine_Delta : constant := 1.0 / (2 ** (Max_Mantissa - 1));

Tick : constant := 10.0E-3;

-- Other System-Dependent Declarations:

subtype Priority is Integer range 0..63;

Max_Object_Size : constant := Max_Int;

Max_Record_Count : constant := Max_Int;

Max_Text_Io_Count : constant := Max_Int - 1;

Max_Text_Io_Field : constant := 1000;

-- Other TeleSoft Declarations:

private

type Subprogram_Value is private;

RECORD

Proc_addr : Address;

Static_link : Address;

Global_frame : Address;

END_RECORD;

end SYSTEM;

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
TLD Systems Ltd. Compiler version 1.3.2	VAX-11 VMS	1750A, MIL-STD-1750A (bare machine)
Compiler version 1.3.2	HP9000 - 350	1750A, MIL-STD-1750A (bare machine)
Compiler version 1.3.2	DG AOS/VS	1750A, MIL-STD-1750A (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Package bodies are loaded as needed, procedures and functions are always loaded.

II. Customization of the Runtime:

- By pragmas
- By compiler switches
- By linker switches
- By modifying the source to the entire runtime

III. Documentation provided to help user configure runtime:

- Interface Control Document

IV. Services to customize the runtime:

- Provided by TLD
- Cost: Charges are negotiated for each case, depending on the complexity of the customization.

V. Cost of runtime source code:

- The runtime source code is included in the price of the compiler.

VI. Source of Information: Vendor Input.

Guidelines to Select, Configure and Use an Ada Runtime Environment

TLD Systems Ltd. PIWG results for MDC281. Clock : 15MHz.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks.	1429.9
A000092	Whetstone benchmarks.	255*
A000093	Whetstone benchmarks.	262*
C000001	Task creation/terminate, task type declared in package.	4848.6
C000002	Task creation/terminate, task type declared in procedure.	4832.2
C000003	Task creation/terminate, task type declared in block.	4811.4
D000001	Dynamic array, use and deallocation.	370.2
D000002	Dynamic array elaboration and initialization.	19279.8
D000003	Dynamic record allocation and deallocation.	383.8
E000001	Raise and handle an exception locally.	37.9
E000002	Raise and handle an exception in a package.	64.1
E000003	Raise and handle an exception nested 3 deep in procedures.	18.4
E000004	Raise and handle an exception nested 4 deep in procedures.	12.7
E000005	Raise and handle an exception in a rendezvous.	198.4
F000001	Set a BOOLEAN flag using a logical equation.	6.6
F000002	Set a BOOLEAN flag using an "if" test.	6.5
L000001	Simple "for" loop.	8.5
L000002	Simple "while" loop.	8.5
L000003	Simple "exit" loop.	7.8
L000004	Loop of 5 iterations with pragma OPTIMIZE (Time).	7.5
L000005	Loop of 5 iterations with pragma OPTIMIZE (Space).	7.6
P000001	Procedure call and return (inlineable), no parameters.	12.5
P000002	Procedure call and return (not inlineable), no parameters.	78.9
P000003	Procedure call and return (compiled separately).	12.5
P000004	Procedure call and return (Pragma INLINE used).	12.5
P000005	Procedure call and return (one parameter, in INTEGER).	12.5
P000006	Procedure call and return (one parameter, out INTEGER).	20.4
P000007	Procedure call and return (one parameter, in out INTEGER).	18.1
P000010	Procedure call and return (ten parameters, in INTEGER).	41.4
P000011	Procedure call and return (twenty parameters, in INTEGER).	93.1
P000012	Procedure call and return (ten parameters, in record_type).	48.4
P000013	Procedure call and return (twenty parameters, in record_type).	104.8
T000001	Minimum rendezvous, entry call and return.	1078.3
T000002	Task entry call and return (one task, one entry).	1078.1
T000003	Task entry call and return (two tasks, one entry each).	1125.6

Guidelines to Select, Configure and Use an Ada Runtime Environment

TLD Systems Ltd. PIWG results for MDC281. Clock : 15MHz.

PIWG Test Name	Description	Micro - seconds
T000004	Task entry call and return (one task, two entries).	1856.8
T000005	Active entry and return (ten tasks, one entry each).	1070.3
T000006	Task entry call and return (one task, ten entries).	3816.7
T000007	Minimum rendezvous, entry call and return.	461.5
T000008	Parameter pass from producer task through buffer task to	3541.7

* WHETSTONE : units are in KWIPS not in microseconds.

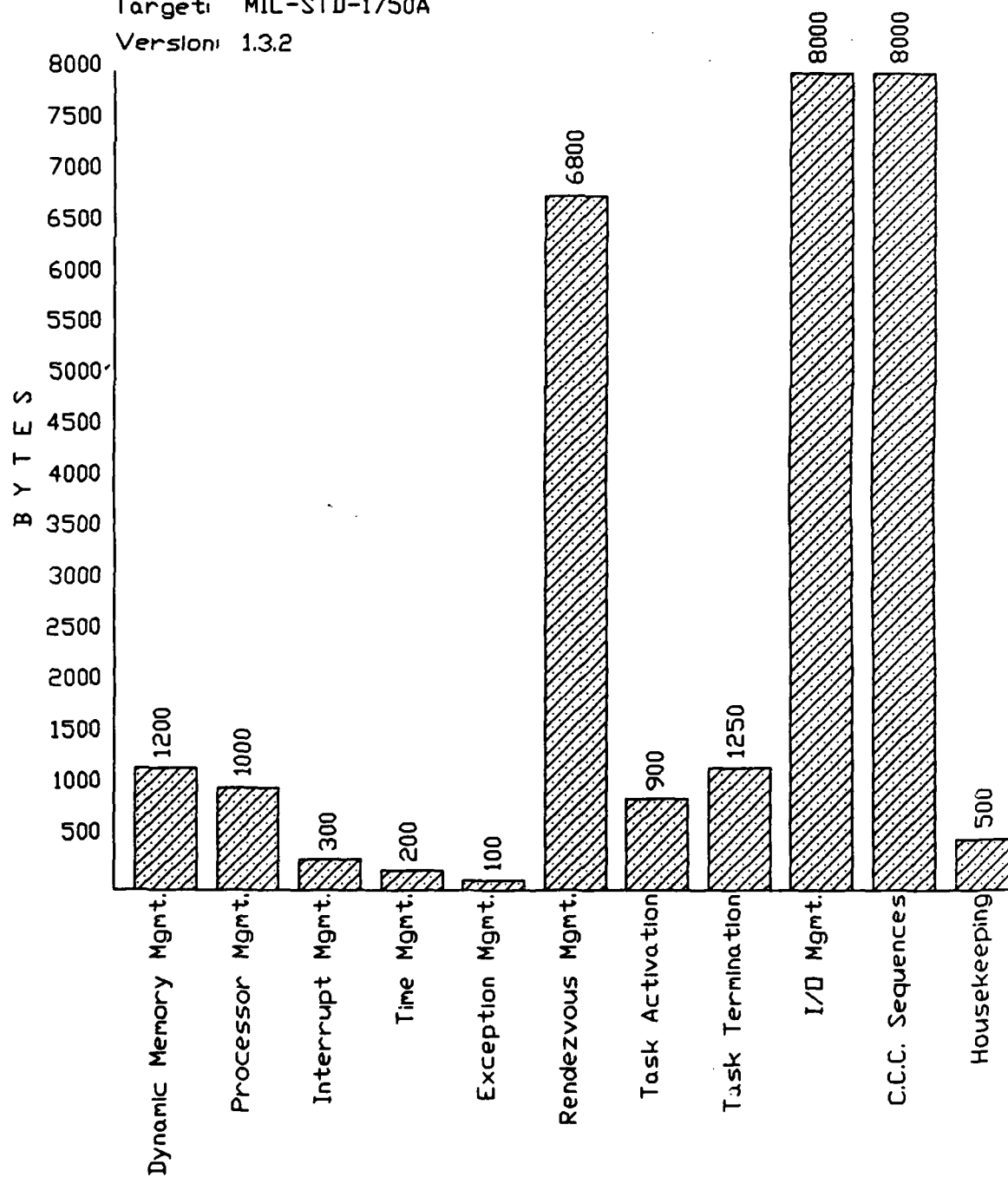
Guidelines to Select, Configure and Use an Ada Runtime Environment

TLD Systems Ltd.

Host: VAX - 11/VMS, HP9000-350, DG ADS/VS

Target: MIL-STD-1750A

Version: 1.3.2



- Sum Of All Components = 28,250 bytes

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: 100 microseconds.

Q2: How long, and for what reasons are interrupts disabled?

A2: Maximum of 200 Microseconds to 1.)update queues, pointers, and other runtime system global variables and 2.) handle interrupts and change context.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: Parameterless, bodyless rendezvous' are optimized. (See T000001)

Q4: What are the restrictions for representation clauses?

A4: 'SMALL is not supported. Nested representation specifications are not supported.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Preemptive priority based scheduling with optional time slicing.

Q6: What are the restrictions on pragma INLINE?

A6: Pragma INLINE is not implemented.

Q7: Is code "ROM"able?

A7: Yes.

Q8: Are machine code inserts supported?

A8: Yes.

Q9: What object types are supported by pragma SHARED?

A9: Scalars types are not supported by pragma SHARED.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

- Maximum number of tasks (No limit)
- Dynamic task priority
- Task time slice default
- Default stack sizes
- Default task priority
- Terminal I/O

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package SYSTEM for the MIL-STD-1750A

package SYSTEM is

```
-- Note: The order of the elements in the OPERATING_SYSTEMS and NAME
-- enumerations cannot be changed--they must correspond with the values
-- in the CONFIG.CFG file.
```

```
type operating_systems is (unix, netos, vms, os_x, msdos, bare);
```

```
type NAME is (none, ns16000, vax, af1750, z8002, z8001, gould, pdp11,
              m68000, pe3200, caps, amdahl, i8086, i80286, i80386, z80000,
              ns32000, ibms1, m68020, nebula, name_x, hp);
```

```
system_name : constant name := name'target;
```

```
os_name      : constant operating_systems := operating_systems'system;
```

```
subtype priority is integer range 1..16#3FEE#; -- one is default priority
subtype interrupt_priority is integer range 16#3FF0#..16#3FFF#;
```

```
pragma put_line('>', '>', '>', ' ', system_name,
                ' ', '/', ' ', ' ', os_name, ' ', '<', '<', '<');
```

```
type address is range 0..65535;
for address'size use 16;
```

```
for unsigned is range 0..65535;
for unsigned size use 16;
```

-- Language defined constants

```
storage_unit : constant := 16;
memory_size  : constant := 65535;
min_int       : constant := -2**31;
max_int       : constant := 2**31-1;
max_digits    : constant := 9; -- 11 digits internally
max_mantissa  : constant := 31;
fine_delta    : constant := 2.0**(-31);
tick          : constant := 1.0/10_000.0; -- Clock ticks are 100 usecs.
rtc_tps       : constant := 10_000;      -- # of counts in one second
-- for systems rtc
min_delay     : constant := rtc_tps * tick; -- Minimum value of Ada delay
address_0     : constant address := 0;      -- Zero address
```

end SYSTEM;

Guidelines to Select, Configure and Use an Ada Runtime Environment

Package STANDARD for the MIL-STD-1750A

```
pragma runtime;  
package STANDARD is  
  
    Package ascii is  
  
        ...  
  
    end ascii;  
  
    subtype Natural is Integer range 0 .. Integer'Last;  
    subtype Positive is Integer range 1 .. Integer'Last;  
  
    type String is array (Positive range <>) of Character;  
    -- Pragma Pack (String);  
  
    -- 32 bits with 12 bits for fractional part.  
  
    type duration is delta 2.0**(-14) range -86_400.0..86_400.0;  
  
    constraint_error : exception;  
    numeric_error : exception;  
    storage_error : exception;  
    tasking_error : exception;  
  
end Standard;
```

Notes: Float is 6 digits, Long_Float is 9 digits.
Integer is 16 bits, Long_Integer is 32 bits.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Verdix Corp. Compiler version 5.5	MicroVAX II (under VMS Version 4.7)	1750A, Fairchild 9450 under Tektronics emulation (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Any part of a library unit being required loads entire unit.
- Available 6/89: Individual subprograms and/or data objects may be extracted from packages only.

II. Customization of the Runtime:

- by pragmas
- by compiler switches
- by linker switches
- by modifying/replacing the source to selective runtime routines provided by the Verdix with purchase of the compiler (i.e. device drivers, etc).
- by modifying the source of the entire runtime (after purchasing it)

III. Documentation provided to help user configure runtime:

VADS User Manual, Configuring VADS <versions number>

IV. Services to customize the runtime:

Verdix supplies support for runtime configuration as part of level 1 and level 2 support contracts. The support is usually by telephone, but for large customers or where the problems may be a fault of Verdix, Verdix may come on site.

V. Cost of runtime source code:

\$25,000 to \$50,000

VI. Source of Information: Vendor input.

PIWG RESULTS

This information was not supplied by the vendor.

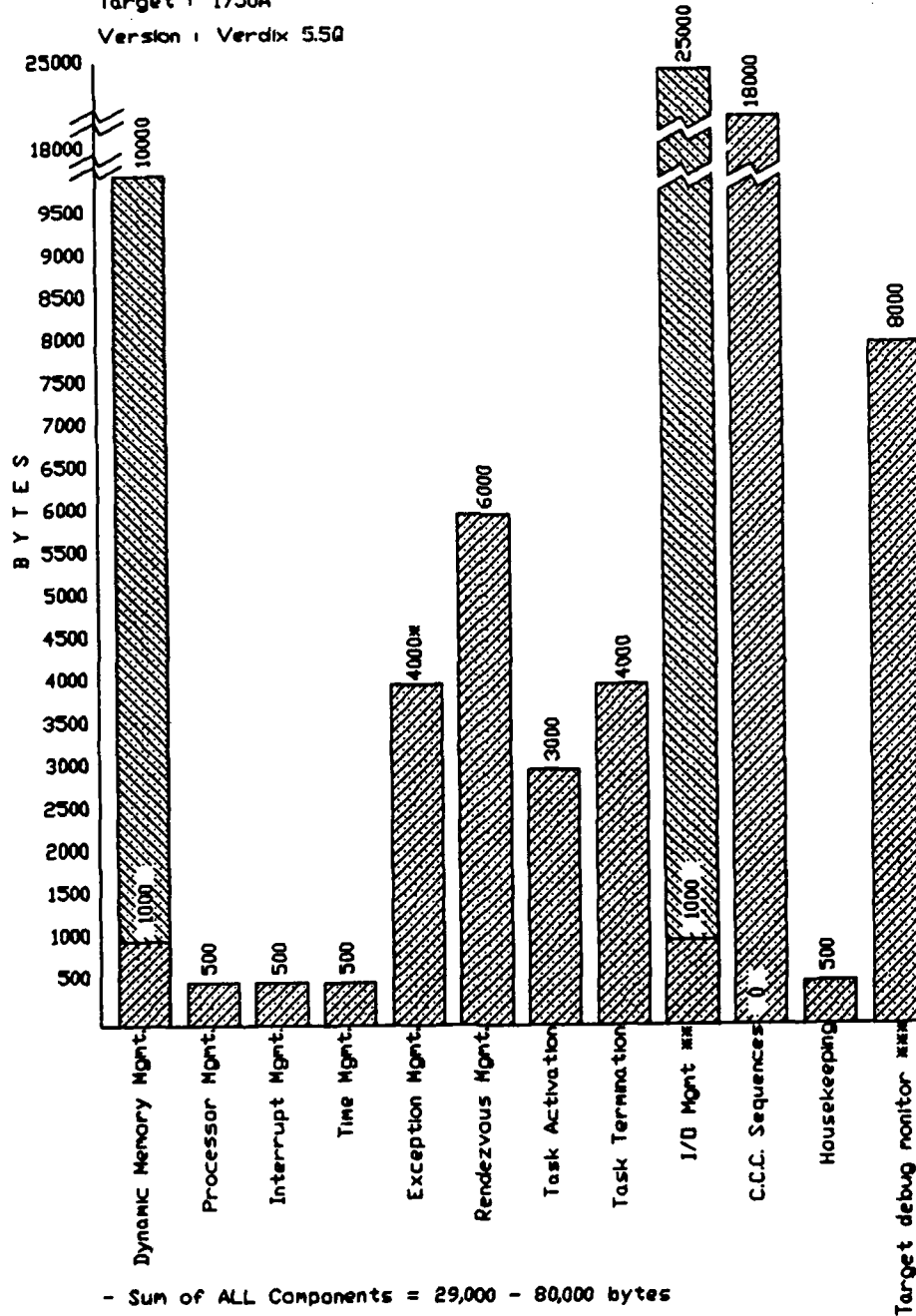
Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix Corporation

Host : VAX - VMS

Target : 1750A

Version : Verdix 5.50



- Sum of ALL Components = 29,000 - 80,000 bytes

= 4,000 bytes + 5-10% of program for tables.

== Text_ID = 25,000 bytes

RS232 = 1,000 bytes

Direct_ID = 8,000 bytes

Sequential_ID = 8,00 bytes

=== Component was supplied by vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: For V5.7 1750A 'Timer B' is used.

Q2: How long, and for what reasons are interrupts disabled?

A2: Not available.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: Pragma PASSIVE - Rendezvous is implemented as procedure call protected by semaphores
Pragma PASSIVE(Interrupt) - Rendezvous implemented as direct hardware interrupt handler protected by interrupt masking hardware.

Simple (trivial) accepts are implemented as "resumes".

Other specific optimizations are also detected.

Q4: What are the restrictions for representation clauses?

A4: Array element sizes are packed only to power-of-two bits, below 16 bits.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Time slicing, Run-until-blocked, dynamic priorities (Only after next addition to pending queue), or priority-inheritance. Time slicing may be applied to individual tasks. Queues are FIFO by priority.

Q6: What are the restrictions on pragma INLINE?

A6: None. However if nobody is available then a call will be generated. A pragma INLINE_ONLY will suppress generation of an out-of-line body, but will force availability of the inline body.

Q7: Is code "ROM"able?

A7: Yes. It is not yet position-independent however, and so must be relinked to be moved.

Q8: Are machine code inserts supported?

A8: Verdex has complete assembler-level machine code for all cross and self-hosted VADS products. In addition, tools such as the optimizer and debugger can operate on machine code (Pragma IMPLICIT_CODE(OFF)) will inhibit optimization and prologue/epilogue Ada support, for "What you see is what you get" machine code.

Q9: What object types are supported by pragma SHARED?

A9: Pragma SHARED inhibits the representation of variables in registers or other non-write-through memory. Only scalars and other register-sized values are affected.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

Maximum number of tasks:	Memory dependent
Task time slice default:	Max clock value
Timer resolution:	Min clock value or about 10 microseconds
Default stack sizes:	Memory dependent
Default task priority:	0-99

Guidelines to Select, Configure and Use an Ada Runtime Environment

Optional numeric coprocessor: unknown
Dynamic task priority: 0-99
Semaphore operations: Yes
Exception trace: Unhandled interrupts
Fast interrupt entry: Yes
Terminal I/O: RS232
Runtime system variations: Yes

Additional items:

- Mailboxes
- Delay-Until
- User-suppliable memory management
- Timed Semaphores
- Suspend/Resume
- Dynamic task priority/Time-slice
- User-supplied task/program creation/switch/destroy "Call Outs"
- Multi-program support
- Multi-processor support (Remote semaphores, Suspend/Resume, Signal, Memory mapping, Memory allocation, Cataloging)
- Emulator support
- Target debug monitor support

Guideline to Select, Configure, and Use an Ada Runtime Environment

Package SYSTEM for the MIL-STD-1750A

package SYSTEM is

type NAME is (m1750a);

SYSTEM_NAME : constant NAME := m1750a ;
EXTENDED_MEMORY : BOOLEAN := FALSE;
STORAGE_UNIT : constant := 16;
MEMORY_SIZE : constant := 2097152;

-- System-Dependent Named Numbers

MIN_INT : constant := -2_147_483_648;
MAX_INT : constant := 2_147_483_647;
MAX_DIGITS : constant := 9;
MAX_MANTISSA : constant := 31;
FINE_DELTA : constant := 2.0**(-31);
TICK : constant := 0.01;

-- Other System-dependent Declarations

subtype PRIORITY is INTEGER range 1..99;

MAX_REC_SIZE : integer := 1*1024;
type SHORT_ADDRESS is private;
type ADDRESS is private;

NO_ADDR : constant ADDRESS;
NO_SHORT_ADDR : constant SHORT_ADDRESS;
subtype SEGMENT is INTEGER range 0..INTEGER'LAST;
function PHYSICAL_ADDRESS (I : INTEGER) return ADDRESS;
function ADDR_GT (A, B: ADDRESS) return BOOLEAN;
function ADDR_LT (A, B: ADDRESS) return BOOLEAN;
function ADDR_GE (A, B: ADDRESS) return BOOLEAN;
function ADDR_LE (A, B: ADDRESS) return BOOLEAN;
function ADDR_DIFF (A, B: ADDRESS) return INTEGER;
function INCR_ADDR (A: ADDRESS; INCR: INTEGER) return ADDRESS;
function DECR_ADDR (A: ADDRESS; DECR: INTEGER) return ADDRESS;

function ">" (A, B: ADDRESS) return BOOLEAN renames ADDR_GT;
function "<" (A, B: ADDRESS) return BOOLEAN renames ADDR_LT;
function ">=" (A, B: ADDRESS) return BOOLEAN renames ADDR_GE;
function "<=" (A, B: ADDRESS) return BOOLEAN renames ADDR_LE;
function "-" (A, B: ADDRESS) return INTEGER renames ADDR_DIFF;
function "+" (A: ADDRESS; INCR: INTEGER) return ADDRESS renames INCR_ADDR;
function "-" (A: ADDRESS; DECR: INTEGER) return ADDRESS renames DECR_ADDR;

Guideline to Select, Configure, and Use an Ada Runtime Environment

Package SYSTEM for the MIL-STD-1750A (Continued)

```
function OFFSET_OF(A : ADDRESS) return SHORT_ADDRESS;
function SEGMENT_OF(A : ADDRESS) return SEGMENT;
function SEGMENT_OF return SEGMENT;
function MAKE_ADDRESS(A : SHORT_ADDRESS; SEG : SEGMENT) return ADDRESS;

function PHYSICAL_ADDRESS(I : LONG_INTEGER) return SHORT_ADDRESS;
function ADDR_GT(A, B : SHORT_ADDRESS) return BOOLEAN;
function ADDR_LT(A, B : SHORT_ADDRESS) return BOOLEAN;
function ADDR_GE(A, B : SHORT_ADDRESS) return BOOLEAN;
function ADDR_LE(A, B : SHORT_ADDRESS) return BOOLEAN;
function ADDR_DIFF(A, B : SHORT_ADDRESS) return INTEGER;
function INCR_ADDR(A : SHORT_ADDRESS; INCR : INTEGER) return SHORT_ADDRESS;
function DECR_ADDR(A : SHORT_ADDRESS; DECR : INTEGER) return SHORT_ADDRESS;

function ">" (A, B: SHORT_ADDRESS) return BOOLEAN renames ADDR_GT;
function "<" (A, B: SHORT_ADDRESS) return BOOLEAN renames ADDR_LT;
function ">=" (A, B: SHORT_ADDRESS) return BOOLEAN renames ADDR_GE;
function "<=" (A, B: SHORT_ADDRESS) return BOOLEAN renames ADDR_LE;
function "-" (A, B: SHORT_ADDRESS) return INTEGER renames ADDR_DIFF;
function "+" (A: SHORT_ADDRESS; INCR: INTEGER) return SHORT_ADDRESS
    renames INCR_ADDR;
function "-" (A: SHORT_ADDRESS; DECR: INTEGER) return SHORT_ADDRESS
    renames DECR_ADDR;

pragma inline (ADDR_GT);
pragma inline (ADDR_LT);
pragma inline (ADDR_GE);
pragma inline (ADDR_LE);
pragma inline (ADDR_DIFF);
pragma inline (INCR_ADDR);
pragma inline (DECR_ADDR);
pragma inline (OFFSET_OF);
pragma inline (SEGMENT_OF);
pragma inline (MAKE_ADDRESS);
pragma inline (PHYSICAL_ADDRESS);

private

    type ADDRESS is new integer;
    type SHORT_ADDRESS is new address;
    for ADDRESS'size use 16;
    for SHORT_ADDRESS'size use 16;

end SYSTEM;
```

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Verdix Corp. Compiler version 5.5	MicroVAX II (under MicroVMS, Version 4.4)	80386, iSBC 386/20P Intel using file- server support from the Host (bare machine)
Compiler version 5.5	MicroVAX II (under VMS, Version 4.7)	80386, iSBC 386/20P Intel using file-server support from the Host (bare machine)
Compiler version 5.5	VAX 8800, 87000 8650, 8600, 8500, 8300, 8200 VAX 11/785, 782, 780, 750, 730, & MicroVAX II (under VMS 4.4)	80386, iSBC 386/20P Intel using file-server support from the Host (bare machine) *Derived*

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Any part of a library unit being required loads entire unit.
- Available 6/89: Individual subprograms and/or data objects may be extracted from packages only.

II. Customization of the Runtime:

- by pragmas
- by compiler switches
- by linker switches
- by modifying/replacing the source to selective runtime routines provided by the Verdix with purchase of the compiler (i.e. device drivers, etc).
- by modifying the source of the entire runtime (after purchasing it)

III. Documentation provided to help user configure runtime:

VADS User Manual, Configuring VADS <versions number>

IV. Services to customize the runtime:

Verdix supplies support for runtime configuration as part of level 1 and level 2 support contracts. The support is usually by telephone, but for large customers or where the problems may be a fault of Verdix, Verdix may come on site.

Guidelines to Select, Configure and Use an Ada Runtime Environment

V. Cost of runtime source code:

\$25,000 to \$50,000

VI. Source of Information: Vendor input.

PIWG RESULTS

This information was not supplied by the vendor.

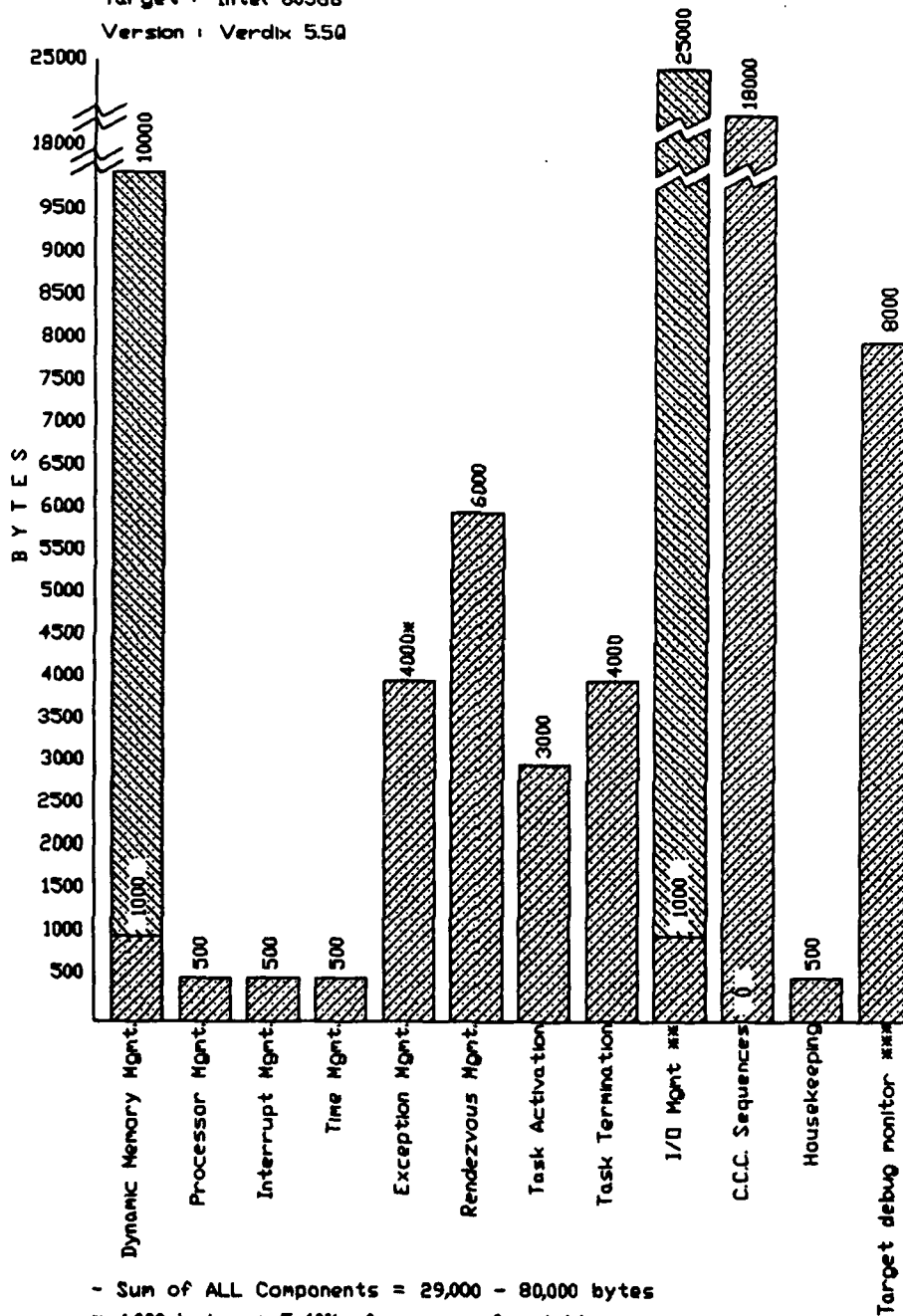
Guidelines to Select, Configure and Use an Ada Runtime Environment

Vendix Corporation

Host : VAX - VMS

Target : Intel 80386

Version : Vendix 5.50



- Sum of ALL Components = 29,000 - 80,000 bytes

* 4,000 bytes + 5-10% of program for tables.

** Text_ID = 25,000 bytes

RS232 = 1,000 bytes

Direct_ID = 8,000 bytes

Sequential_ID = 8,000 bytes

*** Component was supplied by vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: Intel configurable, interrupt service chip, 10 microseconds.

Q2: How long, and for what reasons are interrupts disabled?

A2: Not available.

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

**A3: Pragma PASSIVE - Rendezvous is implemented as procedure call protected by semaphores
Pragma PASSIVE(Interrupt) - Rendezvous implemented as direct hardware interrupt handler protected by interrupt masking hardware.
Simple (trivial) accepts are implemented as "resumes".
Other specific optimizations are also detected.**

Q4: What are the restrictions for representation clauses?

A4: Not supplied.

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Time slicing, Run-until-blocked, dynamic priorities (Only after next addition to pending queue), or priority-inheritance. Time slicing may be applied to individual tasks. Queues are FIFO by priority.

Q6: What are the restrictions on pragma INLINE?

A6: None. However if nobody is available then a call will be generated. A pragma INLINE_ONLY will suppress generation of an out-of-line body, but will force availability of the inline body.

Q7: Is code "ROM"able?

A7: Yes. It is not yet position-independent however, and so must be relinked to be moved.

Q8: Are machine code inserts supported?

A8: Verdix has complete assembler-level machine code for all cross and self-hosted VADS products. In addition, tools such as the optimizer and debugger can operate on machine code (Pragma IMPLICIT_CODE(OFF)) will inhibit optimization and prologue/epilogue Ada support, for "What you see is what you get" machine code.

Q9: What object types are supported by pragma SHARED?

A9: Pragma SHARED inhibits the representation of variables in registers or other non-write-through memory. Only scalars and other register-sized values are affected.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

Maximum number of tasks:	Memory dependent
Task time slice default:	Max clock value
Timer resolution:	Min clock value or about 10 microseconds
Default stack sizes:	Memory dependent
Default task priority:	0-99

Guidelines to Select, Configure and Use an Ada Runtime Environment

Optional numeric coprocessor: unknown
Dynamic task priority: 0-99
Semaphore operations: Yes
Exception trace: Unhandled interrupts
Fast interrupt entry: Yes
Terminal I/O: RS232
Runtime system variations: Yes

Additional items:

- Mailboxes
- Delay-Until
- User-suppliable memory management
- Timed Semaphores
- Suspend/Resume
- Dynamic task priority/Time-slice
- User-supplied task/program creation/switch/destroy "Call Outs"
- Multi-program support
- Multi-processor support (Remote semaphores, Suspend/Resume, Signal, Memory mapping, Memory allocation, Cataloging)
- Emulator support
- Target debug monitor support

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Verdix Corp. Compiler version 5.5	Sun Microsystems Sun-3/160 (under Sun UNIX 4.2, Release 3.2)	68020, Microbar GBC68020 using file-server support from the Host (bare machine)
Compiler version 5.5	MicroVAX II (under UNIX 4.2 BSD)	68020, Microbar GPC-68020 (bare machine)
Compiler version 5.5	MicroVAX II (under MicroVMS 4.4)	68020, Microbar GPC-68020 (bare machine)

DEGREE OF CONFIGURABILITY

I. Linker Capability:

- Any part of a library unit being required loads entire unit.
- Available 6/89: Individual subprograms and/or data objects may be extracted from packages only.

II. Customization of the Runtime:

- by pragmas
- by compiler switches
- by linker switches
- by modifying/replacing the source to selective runtime routines provided by the Verdix with purchase of the compiler (i.e. device drivers, etc).
- by modifying the source of the entire runtime (after purchasing it)

RTS Configuration Parameters

The following excerpts are from Verdix Ada Development System VADS documentation. [13]

ALLOWED_WASTE - Heap memory management parameter. This parameter specifies how much larger the space allocated to a memory request can be than the size of the request.

DISABLE_MASK - The mask used to disable the interrupts when in a critical region in the runtime system. This mask will be stored in the status register when a critical region is entered.

Guidelines to Select, Configure and Use an Ada Runtime Environment

ENABLE_MASK - The mask used to initialize the status register. The **CONFIG_INIT** routine should place this value in the status register.

HEAP_BASE - This is the start address of the heap. The heap will grow from this address toward high memory. The heap is the storage area from which memory is obtained during the execution of an Ada allocator.

HEAP_TOP - This is the address of the last storage unit in the heap. If, during the execution of an Ada allocator, there is not enough space in the heap to allocate an object, the exception **STORAGE_ERROR** will be raised.

MAX_TIME_SLICED_PRIORITY - The range of priorities a task can have is defined by the subtype **PRIORITY** in the package **SYSTEM**. **MAX_TIME_SLICED_PRIORITY** must be in this range.

Any task with a priority higher than **MAX_TIME_SLICED_PRIORITY** will not be preempted by a time slice.

CAUTION: This configuration parameter may be eliminated in a future release. Verdex recommends that **MAX_TIME_SLICED_PRIORITY** not be changed.

NUM_SMALL_BLOCK_SIZES - Heap memory management parameter. This parameter declares the number of small object sizes to be handled by subpools.

SMALL_BLOCK_SIZES - Heap memory management parameter. For each small-block subpool, this array gives the size of blocks in the pool. The sizes must be in ascending order and each size must be a multiple of eight. When the user allocates a small object, the heap memory management routines will use a block from the smallest small-block subpool large enough to handle the request.

STACK_BASE - This parameter defines the initial value of the stack pointer. The stack grows from this address toward low memory.

STACK_LIMIT - The value of this variable is the lowest address the stack pointer may assume. This only applies to the stack of the main program; each task will have its own **STACK_BASE** and **STACK_LIMIT**. The size of a task stack can be specified using the **T'SORAGE_SIZE** length clause.

TIME_SLICING_ENABLED - If **TIME_SLICING_ENABLED** is true, then tasks will be preempted by time slicing. If false, then each task will keep the processor until it executes a delay, enters a rendezvous, or is preempted by an interrupt.

TIME_SLICE_MSECS - This is an array of integers. It must be declared exactly as it is shown in the default configuration package, except that the upper bound may be different and the initial values may be different, but must be static. The upper bound must be at least as large as **MAX_TIME_SLICED_PRIORITY**. If time slicing is enabled, this array is consulted by the RTS to determine the length of a timeslice for a task having a priority in the range 1 .. **MAX_TIME_SLICED_PRIORITY**. Tasks whose priorities are greater than **MAX_TIME_SLICED_PRIORITY** will not be timesliced. The value of an element of the array is the number of milliseconds in a timeslice for a task having that priority.

Guidelines to Select, Configure and Use an Ada Runtime Environment

CAUTION: This configuration parameter may be eliminated in a future release. Verdix recommends that `MAX_TIME_SLICED_PRIORITY` not be used.

VECTOR_BASE - The value of this variable is the physical address of the 680x0 interrupt vector. On the 68000, this will be 0. On the 68010 and 68020, this should be the value the VBR register will contain during the execution of the program.

The RTS uses the following three interrupt vectors:

- number 5, offset 014 hex, used for zero divide
- number 6, offset 018 hex, used for CHK, CHK2 instruction
- number 7, offset 01C hex, used for `cpTRAPcc`, `TRAPcc`, `TRAPV` instructions

The RTS initializes these vectors to convert these interrupts into Ada exceptions.

RTS Configuration Subprograms

Configuration routines for which user implemented routines may be substituted are:

AA_POOL_NEW - Allocates space from the named pool.

COMPACT - Compaction is expensive: it amounts to a sort of a pool free list by address followed by a traversal to coalesce all adjacent memory areas. Compaction can remedy fragmentation. It is called automatically if storage is exhausted. Users may wish to call it explicitly if they want to avoid a random long delay when an arbitrary allocation exhausts memory.

CONFIG_INIT - This procedure is called very early during the initialization of the RTS environment after the stack pointer has been initialized and after necessary ROM has been copied into RAM. `CONFIG_INIT` is called by the RTS startup procedure before any of the configuration parameters are used by the runtime system. Therefore, it is capable of setting the values of RTS configuration variables, if this is desirable (i.e., the configuration variables do not have to be defined statically).

CREATE_POOL - `CREATE_POOL` creates an internal data structure for a pool and returns a descriptor or identifier for the pool. Pools generally use contiguous memory where possible to prevent fragmentation.

CURRENT_POOL - Returns the pool identifier for the current pool.

CURRENT_TIME - This function returns the current time in milliseconds since the `INIT_CLOCK` procedure was called. The package `CALENDAR` calls this function to determine the current time. See *The Clock-Timer* below.

DEALLOCATE_POOL - `DEALLOCATE_POOL` causes all blocks of storage that have been obtained for the named pool to be returned to the heap's free list. Such memory is no longer reserved for that pool and may be used for any heap activity. The heap pool can never be deallocated.

Guidelines to Select, Configure and Use an Ada Runtime Environment

GET_MEMORY - The GET_MEMORY procedure defined in the configuration package is called by the RTS when the heap memory becomes exhausted.

HALT - HALT is called at the very end of the program, after the main Ada subprogram has returned. The default HALT procedure clears the register D0 and then performs a TRAP 15 instruction.

HEAP_POOL - Returns the pool identifier for the heap pool. It may be desirable to use the HEAP_POOL when allocating objects that must persist.

INIT_CLOCK - INIT_CLOCK is called from the RTS during initialization. The RTS passes in the address of a routine that must be called by the clock-handling code whenever the clock interrupts. This routine is also responsible for initializing the current time for the routine CURRENT_TIME. See also *The Clock-Timer* below.

PANIC - PANIC is called when an unhandled exception is detected. It is also called if an internal inconsistency is discovered in the runtime system or if a tasking deadlock is detected. A string parameter is passed into PANIC. If an unhandled exception is being reported, the parameter will contain the following message.

****MAIN PROGRAM ABANDONED -- EXCEPTION "name" RAISED****

name is the name of the exception. For the exceptions defined by the Ada RM (11.1) these names will be the following:

CONSTRAINT_ERROR
NUMERIC_ERROR
PROGRAM_ERROR
STORAGE_ERROR
TASKING_ERROR

SCHEDULE_ALARM - The SCHEDULE_ALARM procedure is called by the RTS to arrange for a clock interrupt at a specified number of milliseconds in the future. See *The Clock-Timer* below.

SWITCH_POOL - At system startup, the heap is the current pool. However, by calling SWITCH_POOL, a program can select an arbitrary pool as the current pool.

TURN_OFF_ALARM - Turn off the next scheduled alarm.

The Clock-Timer

The VADS RTS uses the clock for delays, calendar, and, if enabled, timeslicing.

A timer driver package must be written to control the system's clock device. The body of TIMER_SUPPORT depends on the following types, constants, and routines declared in a package named TIMER.

constant COUNTS_PER_MSEC - number of times the clock ticks per millisecond. This should be a positive integer value as close to 1 as possible. (More ticks per millisecond only serve to decrease the time between clock

Guidelines to Select, Configure and Use an Ada Runtime Environment

maintenance interrupts, since the minimum time-slice or delay time is 1 millisecond).

constant MAX_MSECS - Maximum number of milliseconds for which the clock can be set.

constant MAX_COUNTS - The product *COUNTS_PER_MSEC* * *MAX_MSECS*. *MAX_MSECS* should be selected to provide a value of *MAX_COUNTS* slightly less than the absolute maximum the clock can represent, so that *CURRENT_TIME* can detect overrun.

type COUNTER_T - Type used to represent a quantity of clock ticks.

procedure INIT_TIMER - Initializes the timer hardware. The timer should be initialized to *MAX_COUNTS*. The address passed to this routine should be written to the clock's interrupt vector.

procedure SET_TIMER - Sets the timer to interrupt after a specified number of clock ticks.

procedure READ_TIMER - Returns the number of ticks until the next clock interrupt.

Clock Operation

The default implementation of the *TIMER_SUPPORT* package assumes that a single countdown interval timer is used to support both the runtime system's time requirements (delays and time-slicing), and the calendar package. If this type of time source is available and acceptable, the *TIMER_SUPPORT* package might not have to be modified.

package CONFIG

This package provides definitions for objects used by the RTS and provides hooks into the RTS allowing the user to replace or modify target board dependent routines.

Package *MATH* in *verdixlib* provides mathematical constants, exponential, logarithmic, circular trigonometric, inverse circular trigonometric, hyperbolic trigonometric, polar conversion, *bessel* functions. It is not configurable.

III. Documentation provided to help user configure runtime:

VERDIX Ada Development System VADS Version 5.41 for SUN-3/UNIX => Motorola 68000 Family Processors.

IV. Services to customize the runtime:

Verdix supplies support for runtime configuration as part of level 1 and level 2 support contracts. The support is usually by telephone, but for large customers or where the problems may be a fault of Verdix, Verdix may come on site.

Guidelines to Select, Configure and Use an Ada Runtime Environment

V. Cost of runtime source code:

\$25,000 to \$50,000

VI. Source of Information: Vendor input and compiler documentation.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20. Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization with supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks*	398.0
A000093	Whetstone benchmarks*	580**
C000001	Task creation/terminate, task type declared in package.	1068.8
C000002	Task creation/terminate, task type declared in procedure.	1068.8
C000003	Task creation/terminate, task type declared in block.	1062.5
D000001	Dynamic array, use and deallocation.	16.0
D000002	Dynamic array elaboration and initialization.	1350.0
D000003	Dynamic record allocation and deallocation.	3750.0
D000004	Dynamic record elaboration and initialization.	5050.0
E000001	Raise and handle an exception locally.	318.8
E000002	Raise and handle an exception in a package.	725.0
E000003	Raise and handle an exception nested 3 deep in procedures.	1018.8
E000004	Raise and handle an exception nested 4 deep in procedures.	987.5
E000005	Raise and handle an exception in a rendezvous	1212.5
F000001	Set a BOOLEAN flag using a logical equation.	0.0
F000002	Set a BOOLEAN flag using an "if" test.	0.0
G000005	TEXT_IO.Get an INTEGER from a local string.	212.9
G000006	TEXT_IO.Get a FLOAT from a local string.	1343.8
H000001	BOOLEAN operations on entire PACKed array.	16.0
H000002	BOOLEAN operations on entire array (not packed).	165.6
H000003	BOOLEAN operations on components of a PACKed array.	600.0
H000004	BOOLEAN operations on components of an array (not packed).	179.7
H000005	Move INTEGER to INTEGER (Unchecked_Conversion).	0.0
H000006	Move array of 10 Floats to record (Unchecked_Conversion).	20.9
H000007	Store and extract bit fields, defined by representation clauses.	33.6
L000001	Simple "for" loop.	0.9
L000002	Simple "while" loop.	1.1
L000003	Simple "exit" loop.	1.1
L000004	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Time).	1.3
L000005	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Space).	1.3
P000001	Procedure call and return (inlineable), no parameters.	3.2
P000002	Procedure call and return (not inlineable), no parameters.	5.6
P000003	Procedure call and return (compiled separately).	5.9
P000004	Procedure call and return (Pragma INLINE used).	0.0

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20 (Continued). Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization with supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
P000005	Procedure call and return (one parameter, in INTEGER).	6.4
P000006	Procedure call and return (one parameter, out INTEGER).	6.3
P000007	Procedure call and return (one parameter, in out INTEGER).	7.3
P000010	Procedure call and return (ten parameters, in INTEGER).	15.2
P000011	Procedure call and return (twenty parameters, in INTEGER).	27.3
P000012	Procedure call and return (ten parameters, in record_type).	20.1
P000013	Procedure call and return (twenty parameters, in record_type).	36.3
T000001	Minimum rendezvous, entry call and return.	267.2
T000002	Task entry call and return (one task, one entry).	270.3
T000003	Task entry call and return (two tasks, one entry each).	271.9
T000004	Task entry call and return (one task, two entries).	337.5
T000005	Active entry and return (ten tasks, one entry each).	270.0
T000006	Task entry call and return (one task, ten entries).	417.5
T000007	Minimum rendezvous, entry call and return.	182.8
T000008	Measures time to pass integer from producer to consumer task.	775.0

* Using standard internal math routines.

** WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20. Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization, suppress checking enforced with supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks*	289.0
A000093	Whetstone benchmarks*	606**
C000001	Task creation/terminate, task type declared in package.	1075.0
C000002	Task creation/terminate, task type declared in procedure.	1068.8
C000003	Task creation/terminate, task type declared in block.	1062.5
D000001	Dynamic array, use and deallocation.	11.3
D000002	Dynamic array elaboration and initialization.	1137.5
D000003	Dynamic record allocation and deallocation.	3775.0
D000004	Dynamic record elaboration and initialization.	4875.0
E000001	Raise and handle an exception locally.	318.8
E000002	Raise and handle an exception in a package.	725.0
E000003	Raise and handle an exception nested 3 deep in procedures.	1018.8
E000004	Raise and handle an exception nested 4 deep in procedures.	981.3
E000005	Raise and handle an exception in a rendezvous	1212.5
F000001	Set a BOOLEAN flag using a logical equation.	1.6
F000002	Set a BOOLEAN flag using an "if" test.	1.5
G000005	TEXT_IO.Get an INTEGER from a local string.	216.8
G000006	TEXT_IO.Get a FLOAT from a local string.	1343.8
H000001	BOOLEAN operations on entire PACKed array.	15.0
H000002	BOOLEAN operations on entire array (not packed).	162.5
H000003	BOOLEAN operations on components of a PACKed array.	578.1
H000004	BOOLEAN operations on components of an array (not packed).	170.3
H000005	Move INTEGER to INTEGER (Unchecked_Conversion).	0.0
H000006	Move array of 10 Floats to record (Unchecked_Conversion).	17.6
H000007	Store and extract bit fields, defined by representation clauses.	33.6
L000001	Simple "for" loop.	0.9
L000002	Simple "while" loop.	0.9
L000003	Simple "exit" loop.	0.8
L000004	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Time).	1.3
L000005	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Space).	1.3
P000001	Procedure call and return (inlineable), no parameters.	5.6
P000002	Procedure call and return (not inlineable), no parameters.	7.3
P000003	Procedure call and return (compiled separately).	1.3
P000004	Procedure call and return (Pragma INLINE used).	0.0

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20 (Continued). Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization, suppress checking enforced with supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
P000005	Procedure call and return (one parameter, in INTEGER).	3.1
P000006	Procedure call and return (one parameter, out INTEGER).	4.0
P000007	Procedure call and return (one parameter, in out INTEGER).	4.6
P000010	Procedure call and return (ten parameters, in INTEGER).	11.1
P000011	Procedure call and return (twenty parameters, in INTEGER).	23.4
P000012	Procedure call and return (ten parameters, in record_type).	16.2
P000013	Procedure call and return (twenty parameters, in record_type).	31.3
T000001	Minimum rendezvous, entry call and return.	270.3
T000002	Task entry call and return (one task, one entry).	270.3
T000003	Task entry call and return (two tasks, one entry each).	270.3
T000004	Task entry call and return (one task, two entries).	337.5
T000005	Active entry and return (ten tasks, one entry each).	265.0
T000006	Task entry call and return (one task, ten entries).	420.0
T000007	Minimum rendezvous, entry call and return.	181.3
T000008	Measures time to pass integer from producer to consumer task.	775.0

* Using standard internal math routines.

** WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20. Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization, suppress checking enforced, register variables used (when register variables are not present, that includes both user code and kernel). Supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks*	241.0
A000093	Whetstone benchmarks*	631**
C000001	Task creation/terminate, task type declared in package.	987.5
C000002	Task creation/terminate, task type declared in procedure.	993.8
C000003	Task creation/terminate, task type declared in block.	987.5
D000001	Dynamic array, use and deallocation.	5.9
D000002	Dynamic array elaboration and initialization.	1125.0
D000003	Dynamic record allocation and deallocation.	3725.0
D000004	Dynamic record elaboration and initialization.	4875.0
E000001	Raise and handle an exception locally.	390.6
E000002	Raise and handle an exception in a package.	603.1
E000003	Raise and handle an exception nested 3 deep in procedures.	868.8
E000004	Raise and handle an exception nested 4 deep in procedures.	1225.0
E000005	Raise and handle an exception in a rendezvous	1125.0
F000001	Set a BOOLEAN flag using a logical equation.	3.4
F000002	Set a BOOLEAN flag using an "if" test.	3.3
G000005	TEXT_IO.Get an INTEGER from a local string.	199.2
G000006	TEXT_IO.Get a FLOAT from a local string.	1140.6
H000001	BOOLEAN operations on entire PACKed array.	15.2
H000002	BOOLEAN operations on entire array (not packed).	164.1
H000003	BOOLEAN operations on components of a PACKed array.	490.6
H000004	BOOLEAN operations on components of an array (not packed).	124.2
H000005	Move INTEGER to INTEGER (Unchecked_Conversion).	2.5
H000006	Move array of 10 Floats to record (Unchecked_Conversion).	14.6
H000007	Store and extract bit fields, defined by representation clauses.	33.6
L000001	Simple "for" loop.	2.1
L000002	Simple "while" loop.	2.4
L000003	Simple "exit" loop.	2.4
L000004	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Time).	0.2
L000005	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Space).	0.2
P000001	Procedure call and return (inlineable), no parameters.	3.0
P000002	Procedure call and return (not inlineable), no parameters.	4.2
P000003	Procedure call and return (compiled separately).	2.9
P000004	Procedure call and return (Pragma INLINE used).	1.7

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20 (Continued). Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization, suppress checking enforced, register variables used (when register variables are not present, that includes both user code and kernel). Supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
P000005	Procedure call and return (one parameter, in INTEGER).	4.0
P000006	Procedure call and return (one parameter, out INTEGER).	4.1
P000007	Procedure call and return (one parameter, in out INTEGER).	4.9
P000010	Procedure call and return (ten parameters, in INTEGER).	10.9
P000011	Procedure call and return (twenty parameters, in INTEGER).	14.5
P000012	Procedure call and return (ten parameters, in record_type).	10.2
P000013	Procedure call and return (twenty parameters, in record_type).	21.1
T000001	Minimum rendezvous, entry call and return.	262.5
T000002	Task entry call and return (one task, one entry).	264.1
T000003	Task entry call and return (two tasks, one entry each).	265.6
T000004	Task entry call and return (one task, two entries).	331.3
T000005	Active entry and return (ten tasks, one entry each).	257.5
T000006	Task entry call and return (one task, ten entries).	400.0
T000007	Minimum rendezvous, entry call and return.	175.0
T000008	Measures time to pass integer from producer to consumer task.	737.5

* Using standard internal math routines.

** WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20. Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization, suppress checking enforced, register variables used (when register variables are not present, that includes both user code and kernel), pragma passive (interrupts) used. Supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks*	241.0
A000093	Whetstone benchmarks*	631**
C000001	Task creation/terminate, task type declared in package.	987.5
C000002	Task creation/terminate, task type declared in procedure.	993.8
C000003	Task creation/terminate, task type declared in block.	987.5
D000001	Dynamic array, use and deallocation.	5.9
D000002	Dynamic array elaboration and initialization.	1125.0
D000003	Dynamic record allocation and deallocation.	3725.0
D000004	Dynamic record elaboration and initialization.	4875.0
E000001	Raise and handle an exception locally.	390.6
E000002	Raise and handle an exception in a package.	603.1
E000003	Raise and handle an exception nested 3 deep in procedures.	868.8
E000004	Raise and handle an exception nested 4 deep in procedures.	1225.6
E000005	Raise and handle an exception in a rendezvous	1125.0
F000001	Set a BOOLEAN flag using a logical equation.	3.4
F000002	Set a BOOLEAN flag using an "if" test.	3.3
G000005	TEXT_IO.Get an INTEGER from a local string.	199.2
G000006	TEXT_IO.Get a FLOAT from a local string.	1140.6
H000001	BOOLEAN operations on entire PACKed array.	15.2
H000002	BOOLEAN operations on entire array (not packed).	164.1
H000003	BOOLEAN operations on components of a PACKed array.	490.6
H000004	BOOLEAN operations on components of an array (not packed).	124.2
H000005	Move INTEGER to INTEGER (Unchecked_Conversion).	2.5
H000006	Move array of 10 Floats to record (Unchecked_Conversion).	14.6
H000007	Store and extract bit fields, defined by representation clauses.	33.6
L000001	Simple "for" loop.	2.1
L000002	Simple "while" loop.	2.4
L000003	Simple "exit" loop.	2.4
L000004	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Time).	0.2
L000005	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Space).	0.2
P000001	Procedure call and return (inlineable), no parameters.	3.0
P000002	Procedure call and return (not inlineable), no parameters.	4.2
P000003	Procedure call and return (compiled separately).	2.9
P000004	Procedure call and return (Pragma INLINE used).	1.7

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20 (Continued). Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization, suppress checking enforced, register variables used (when register variables are not present, that includes both user code and kernel), pragma passive (interrupts) used. Supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
P000005	Procedure call and return (one parameter, in INTEGER).	4.0
P000006	Procedure call and return (one parameter, out INTEGER).	4.1
P000007	Procedure call and return (one parameter, in out INTEGER).	4.9
P000010	Procedure call and return (ten parameters, in INTEGER).	10.9
P000011	Procedure call and return (twenty parameters, in INTEGER).	14.5
P000012	Procedure call and return (ten parameters, in record_type).	10.2
P000013	Procedure call and return (twenty parameters, in record_type).	21.1
T000001	Minimum rendezvous, entry call and return.	262.5
T000002	Task entry call and return (one task, one entry).	46.9
T000003	Task entry call and return (two tasks, one entry each).	47.7
T000004	Task entry call and return (one task, two entries).	50.0
T000005	Active entry and return (ten tasks, one entry each).	43.4
T000006	Task entry call and return (one task, ten entries).	60.6
T000007	Minimum rendezvous, entry call and return.	175.0
T000008	Measures time to pass integer from producer to consumer task.	184.4

* Using standard internal math routines.

** WHETSTONE : units are in KWIPS not in microseconds.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20. Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization, suppress checking enforced, register variables used (when register variables are not present, that includes both user code and kernel), pragma passive (interrupts) used, pragma passive (semaphores) used. Supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
A000091	Dhrystone benchmarks*	241.0
A000093	Whetstone benchmarks*	631**
C000001	Task creation/terminate, task type declared in package.	987.5
C000002	Task creation/terminate, task type declared in procedure.	993.8
C000003	Task creation/terminate, task type declared in block.	987.5
D000001	Dynamic array, use and deallocation.	5.9
D000002	Dynamic array elaboration and initialization.	1125.0
D000003	Dynamic record allocation and deallocation.	3725.0
D000004	Dynamic record elaboration and initialization.	4875.0
E000001	Raise and handle an exception locally.	390.6
E000002	Raise and handle an exception in a package.	603.1
E000003	Raise and handle an exception nested 3 deep in procedures.	868.8
E000004	Raise and handle an exception nested 4 deep in procedures.	1225.0
E000005	Raise and handle an exception in a rendezvous	1125.0
F000001	Set a BOOLEAN flag using a logical equation.	3.4
F000002	Set a BOOLEAN flag using an "if" test.	3.3
G000005	TEXT_IO.Get an INTEGER from a local string.	199.2
G000006	TEXT_IO.Get a FLOAT from a local string.	1140.6
H000001	BOOLEAN operations on entire PACKed array.	15.2
H000002	BOOLEAN operations on entire array (not packed).	164.1
H000003	BOOLEAN operations on components of a PACKed array.	490.6
H000004	BOOLEAN operations on components of an array (not packed).	124.2
H000005	Move INTEGER to INTEGER (Unchecked_Conversion).	2.5
H000006	Move array of 10 Floats to record (Unchecked_Conversion).	14.6
H000007	Store and extract bit fields, defined by representation clauses.	33.6
L000001	Simple "for" loop.	2.1
L000002	Simple "while" loop.	2.4
L000003	Simple "exit" loop.	2.4
L000004	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Time).	0.2
L000005	Unwrap of loop of 5 iterations with pragma OPTIMIZE(Space).	0.2
P000001	Procedure call and return (inlineable), no parameters.	3.0
P000002	Procedure call and return (not inlineable), no parameters.	4.2
P000003	Procedure call and return (compiled separately).	2.9
P000004	Procedure call and return (Pragma INLINE used).	1.7

Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix PIWG results for Motorola MVME 133A-20 (Continued). Clock: 20MHz, MC68020/MC68881, Memory Wait States not supplied. Full optimization, suppress checking enforced, register variables used (when register variables are not present, that includes both user code and kernel), pragma passive (interrupts) used, pragma passive (semaphores) used. Supervisor task enabled, time-slicing disabled, priority-inheritance disabled. PIWG test suite 1987.

PIWG Test Name	Description	Micro - seconds
P000005	Procedure call and return (one parameter, in INTEGER).	4.0
P000006	Procedure call and return (one parameter, out INTEGER).	4.1
P000007	Procedure call and return (one parameter, in out INTEGER).	4.9
P000010	Procedure call and return (ten parameters, in INTEGER).	10.9
P000011	Procedure call and return (twenty parameters, in INTEGER).	14.5
P000012	Procedure call and return (ten parameters, in record_type).	10.2
P000013	Procedure call and return (twenty parameters, in record_type).	21.1
T000001	Minimum rendezvous, entry call and return.	262.5
T000002	Task entry call and return (one task, one entry).	46.9
T000003	Task entry call and return (two tasks, one entry each).	48.0
T000004	Task entry call and return (one task, two entries).	49.6
T000005	Active entry and return (ten tasks, one entry each).	44.4
T000006	Task entry call and return (one task, ten entries).	61.9
T000007	Minimum rendezvous, entry call and return.	175.0
T000008	Measures time to pass integer from producer to consumer task.	184.4

* Using standard internal math routines.

** WHETSTONE : units are in KWIPS not in microseconds.

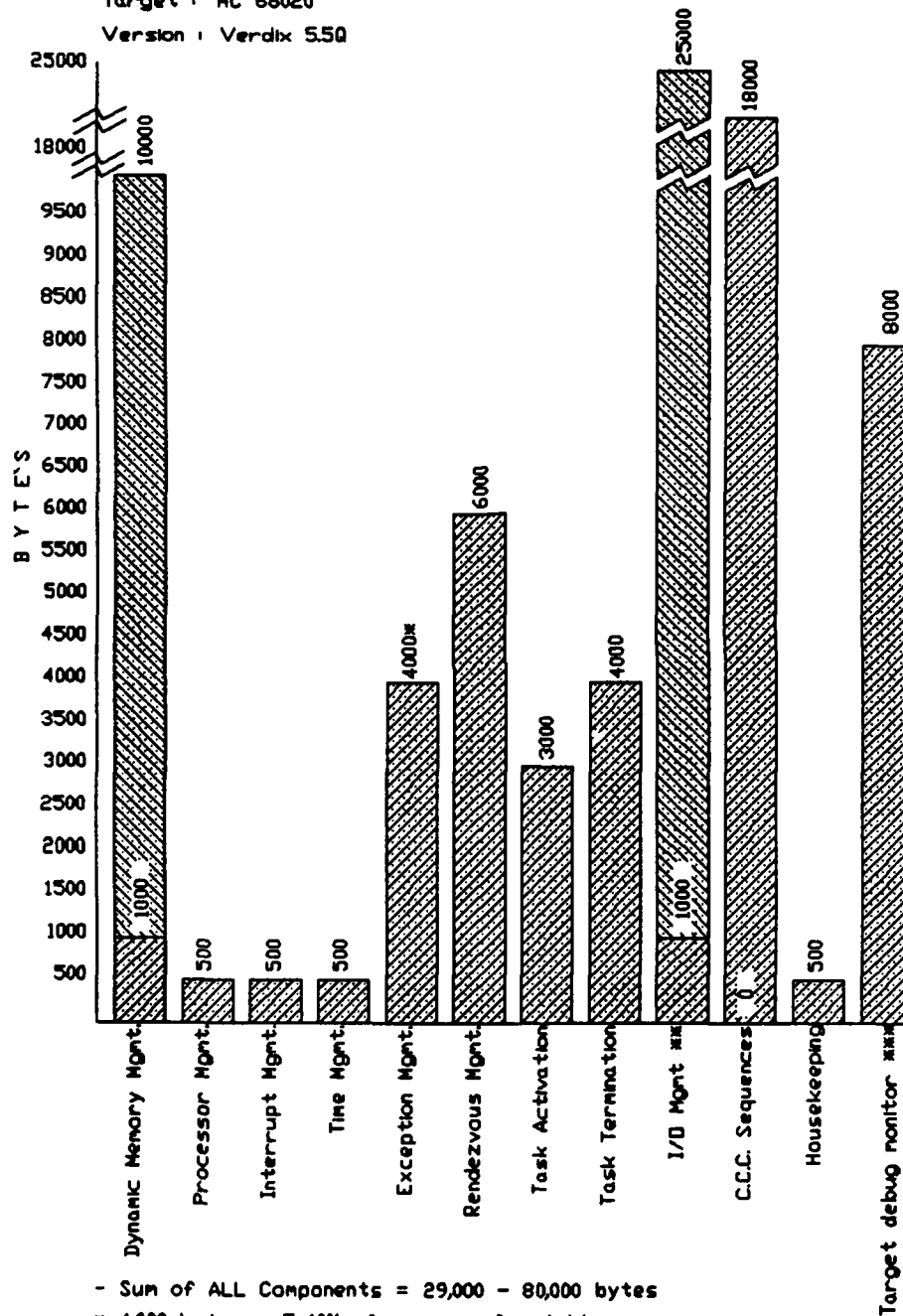
Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix Corporation

Host : VAX - VMS

Target : MC 68020

Version : Verdix 5.50



- Sum of ALL Components = 29,000 - 80,000 bytes

■ 4,000 bytes + 5-10% of program for tables.

■ Text_IO = 25,000 bytes

RS232 = 1,000 bytes

Direct_IO = 8,000 bytes

Sequential_IO = 8,000 bytes

■■■ Component was supplied by vendor.

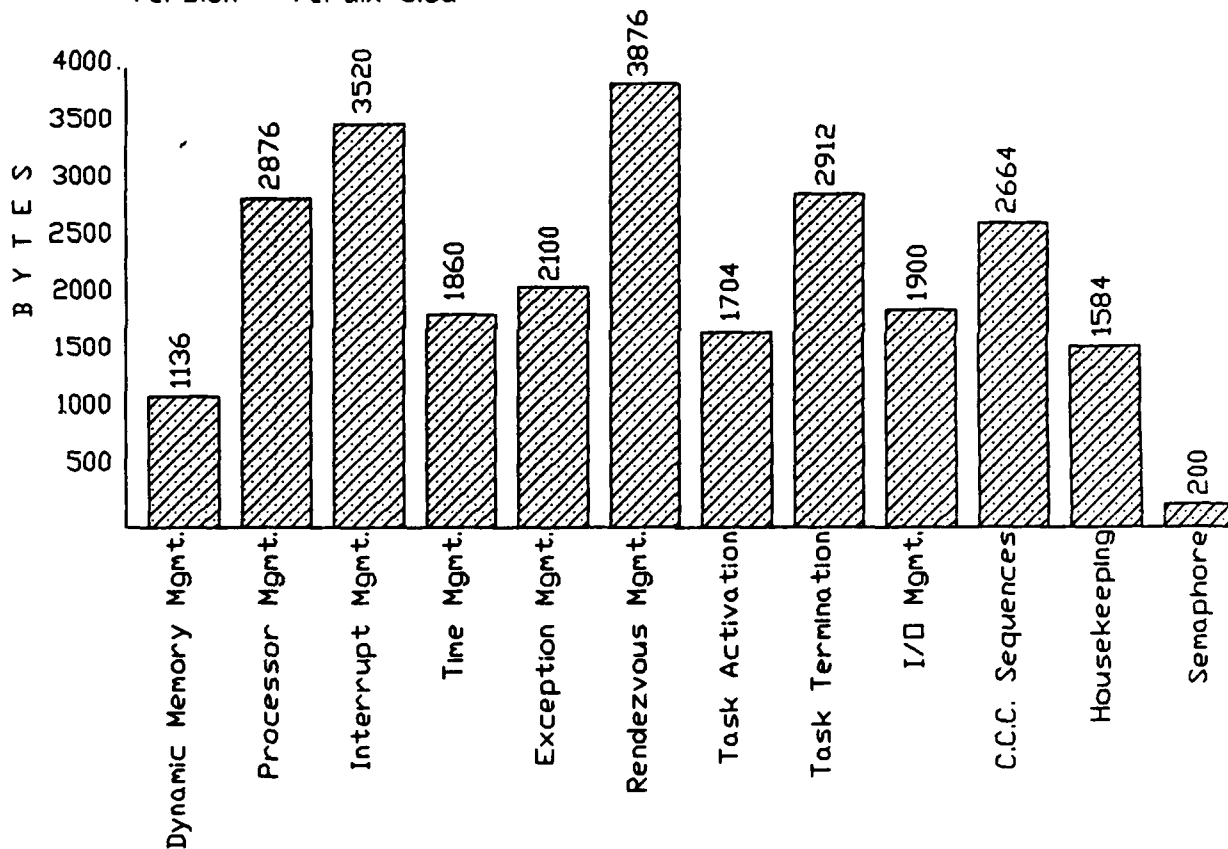
Guidelines to Select, Configure and Use an Ada Runtime Environment

Verdix Corporation

Host : VAX - VMS

Target : MC 68020

Version : Verdix 5.5Q



- Sum of ALL Components = 26,332 bytes

NOTE: User supplied information for a particular implementation.

Guidelines to Select, Configure and Use an Ada Runtime Environment

Response to Critical Questions

Q1: What is the resolution of the clock used for delay statements?

A1: For V5.7 68020 the duration is 1 millisecond. Clock is user configurable.

Q2: How long, and for what reasons are interrupts disabled?

A2: For V5.7 68020 the user can configure what interrupt level is disabled while in kernel at:

1. Adding/Deleting from PENDING queue $5 + 5 \cdot N/2$, Where N = number of tasks on pending queue.
2. Adjusting clock - (User configurable)
3. Leave kernel and return to user - (Approximately: 1 microseconds)
4. In passive interrupt tasks - (User configurable)
5. Interrupt handler as it enters the kernel - (Approximately: 3 microseconds)

Q3: What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

A3: **Pragma PASSIVE** - Rendezvous is implemented as procedure call protected by semaphores
Pragma PASSIVE(Interrupt) - Rendezvous implemented as direct hardware interrupt handler protected by interrupt masking hardware.

Simple (trivial) accepts are implemented as "resumes".

Other specific optimizations are also detected.

Q4: What are the restrictions for representation clauses?

A4: Array element sizes are packed only to power-of-two bits, below 16 bits. Thus a five bit element will be packed as 8 bits, a 1 bit element is packed as 1 bit. Record fields must be extractable entirely in one machine register, unless the hardware supplies bit field instructions (in that case, the instruction restrictions apply). Thus a field of 31 bits must begin on bit 0 or bit 1 for a 680x0, while a 7 or even 25 bit field may begin anywhere

Q5: What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

A5: Time slicing, Run-until-blocked, dynamic priorities (Only after next addition to pending queue), or priority-inheritance. Time slicing may be applied to individual tasks. Queues are FIFO by priority.

Q6: What are the restrictions on **pragma INLINE**?

A6: None. However if no body is available then a call will be generated. A **pragma INLINE_ONLY** will suppress generation of an out-of-line body, but will force availability of the inline body.

Q7: Is code "ROM"able?

A7: Yes. It is not yet position-independent however, and so must be relinked to be moved.

Q8: Are machine code inserts supported?

A8: Verdex has complete assembler-level machine code for all cross and self-hosted VADS products. In addition, tools such as the optimizer and debugger can operate on machine code (**Pragma IMPLICIT_CODE(OFF)**) will inhibit optimization and prologue/epilogue Ada support, for "What you see is what you get" machine code.

Q9: What object types are supported by **pragma SHARED**?

Guidelines to Select, Configure and Use an Ada Runtime Environment

A9: Pragma SHARED inhibits the representation of variables in registers or other non-write-through memory. Only scalars and other register-sized values are affected.

Q10: What items are configurable for the runtime system?

A10: The items below are configurable for the runtime system.

Maximum number of tasks:	Memory dependent
Task time slice default:	Max clock value
Timer resolution:	Min clock value or about 10 microseconds
Default stack sizes:	Memory dependent
Default task priority:	0-99
Optional numeric coprocessor:	68881, soon WEITEK
Dynamic task priority:	0-99
Semaphore operations:	Yes
Exception trace:	Unhandled interrupts
Fast interrupt entry:	Yes
Terminal I/O:	RS232
Runtime system variations:	Yes

Additional items:

- Mailboxes
- Delay-Until
- User-suppliable memory management
- Timed Semaphores
- Suspend/Resume
- Dynamic task priority/Time-slice
- User-supplied task/program creation/switch/destroy "Call Outs"
- Multi-program support
- Multi-processor support (Remote semaphores, Suspend/Resume, Signal, Memory mapping, Memory allocation, Cataloging)
- Emulator support
- Target debug monitor support

Guideline to Select, Configure, and Use an Ada Runtime Environment

Package SYSTEM for the MC68000 Operating

package SYSTEM is

type NAME is (m68k);

SYSTEM_NAME : constant NAME := m68k ;

STORAGE_UNIT : constant := 8;

MEMORY_SIZE : constant := 16_777_216;

-- System-Dependent Named Numbers

MIN_INT : constant := -2_147_483_648;

MAX_INT : constant := 2_147_483_647;

MAX_DIGITS : constant := 15;

MAX_MANTISSA : constant := 31;

FINE_DELTA : constant := 2.0**(-30);

TICK : constant := 0.1;

-- Other System-dependent Declarations

subtype PRIORITY is INTEGER range 1..99;

MAX_REC_SIZE : integer := 64*1024;

type ADDRESS is private;

NO_ADDR : constant ADDRESS;

function PHYSICAL_ADDRESS (I : INTEGER) return ADDRESS;

function ADDR_GT (A, B: ADDRESS) return BOOLEAN;

function ADDR_LT (A, B: ADDRESS) return BOOLEAN;

function ADDR_GE (A, B: ADDRESS) return BOOLEAN;

function ADDR_LE (A, B: ADDRESS) return BOOLEAN;

function ADDR_DIFF (A, B: ADDRESS) return INTEGER;

function INCR_ADDR (A: ADDRESS; INCR: INTEGER) return ADDRESS;

function DECR_ADDR (A: ADDRESS; DECR: INTEGER) return ADDRESS;

function ">" (A, B: ADDRESS) return BOOLEAN renames ADDR_GT;

function "<" (A, B: ADDRESS) return BOOLEAN renames ADDR_LT;

function ">=" (A, B: ADDRESS) return BOOLEAN renames ADDR_GE;

function "<=" (A, B: ADDRESS) return BOOLEAN renames ADDR_LE;

function "-" (A, B: ADDRESS) return INTEGER renames ADDR_DIFF;

function "+" (A: ADDRESS; INCR: INTEGER) return ADDRESS

renames INCR_ADDR;

function "-" (A: ADDRESS; DECR: INTEGER) return ADDRESS

renames DECR_ADDR;

Guideline to Select, Configure, and Use an Ada Runtime Environment

Package SYSTEM for the MC68000 Operating

```
pragma inline (ADDR_GT);
pragma inline (ADDR_LT);
pragma inline (ADDR_GE);
pragma inline (ADDR_LE);
pragma inline (ADDR_DIFF);
pragma inline (INCR_ADDR);
pragma inline (DECR_ADDR);
pragma inline (PHYSICAL_ADDRESS);

private

  type ADDRESS is new integer;
  NO_ADDR : constant ADDRESS := 0;

end SYSTEM;
```


Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Advanced Computer Techniques Corp. (InterACT) Compiler version 2.1	VAX-11/785 (under VMS 4.4)	1750A, Fairchild 9450/1750A in a HP 64000 workstation (bare machine)

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

This information was not supplied by the vendor.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Harris Corporation Compiler version 1.0	Harris HCX-7 Series (under HCX/UX, Version 2.2)	1750A, Tektronix 8540A (bare machine)
Compiler version 1.0	Harris H1200 (under VOS, Version 6.1)	1750A, Tektronix 8540A (bare machine)

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

This information was not supplied by the vendor.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Rockwell Int'l. Compiler version 1.0	VAX-11/8650 (under VMS, version 4.5)	CAPS/AAMP (bare machine)
Compiler version 2.0	DEC VAX 8650 (under VMS, version 4.7)	CAPS/AAMP (bare machine)

This compiler is not for sale to the general public, therefore the information was not provided.

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

This information was not supplied by the vendor.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Systems Designers Software, Inc. Compiler version 2C.00	DEC VAX-11/7xx, VAX 8xxx, VAX Station, and MicroVAX Series (under VAX/VMS 4.5 or MicroVMS 4.5)	68000, MC68000/10 implemented on the MVME 117-3FP board (bare machine) * Derived *
Compiler version 3A.00	DEC VAX-11/7xx, VAX 8xxx, VAX Station, (under VMS 4.6) and MicroVAX Series (under MicroVMS 4.5)	68000, MC68000/10 implemented on the MVME 177-3 FP board (bare machine) * Derived *

Note: Although a response was not provided for these targets, a response for a bare MC68020 was provided, and it is reasonable to expect similar capabilities (excluding performance) for these implementations. Refer to System Designers' bare MC68020 target processor response.

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

This information was not supplied by the vendor.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
TeleSoft/Intel Corp./TeleLOGIC Compiler version	VAX 8530 (under VMS, version 4.6)	80386, Intel 80386 on Intel 386-100 board (bare machine)

No information was provided due to the recent validation. Time did not permit inclusion of this in the report. Please contact the vendor for further information.

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

This information was not supplied by the vendor.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
TeleSoft, Inc. Compiler version 1.2	Sun Microsystems Sun-3/280 Workstation (under Sun UNIX version 4.2, release 3.2)	MC68000, implemented on a Motorola MVME 101 board (bare machine)
Compiler version 1.2	Sun Microsystems Sun-3 Workstations, Models: 260, 180, 160, 150, 140, 110, 75, 60, 50 and 52 (with soft- ware floating point); 50ME and 52 + 152A (with MC68881 FPC) (under Sun UNIX version 4.2, Releases 3.2 & 3.4)	MC68000, implemented on a Motorola MVME 101 board (bare machine) *Derived*
Compiler version 3.2	MicroVAX II (under VMS, version 4.6)	MC68000, implemented on a Motorola MVME 101 board (bare machine)
Compiler version 3.2	DEC VAX family (MicroVAX, VAX station, VAX server, VAX 8xxx, & VAX-11 models) (under VMS 4.5 and 4.6)	MC68000, implemented on a Motorola MVME 101 board (bare machine) * Derived*
Compiler version 1.2	Sun Microsystems Sun-3/280 Workstation (under Sun UNIX version 4.2, release 3.2)	MC68010, implemented on a Motorola MVME 117-4 board (bare machine)
Compiler version 1.2	Sun Microsystems Sun-3 Workstations, Models: 260, 180, 160, 150, 140, 110, 75, 60, 50 and 52 (with soft- ware floating point); 50ME and 52 + 152A (with MC68881 FPC) (under Sun UNIX version 4.2, Releases 3.2 & 3.4)	MC68010, implemented on a Motorola MVME 117-4 board (bare machine) *Derived*

(Continued on next page)

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
TeleSoft, Inc. Compiler version 3.2	MicroVAX II (under VMS, version 4.6)	MC68010, implemented on a Motorola MVME 117-4 board 117-4 board (bare machine)
Compiler version 3.2	DEC VAX family (MicroVAX, VAX station, VAX server, VAX 8xxx, & VAX-11 models) (under VMS 4.5 and 4.6)	MC68010, implemented on a Motorola MVME 117-4 board (bare machine) * Derived*
Compiler version 1.2	Sun Microsystems Sun-3 Workstations, Models: 260, 180, 160, 150, 140, 110, 75, 60, 50 and 52 (with soft- ware floating point); SOME and 52 + 152A (with MC68881 FPC) (under Sun UNIX version 4.2, Releases 3.2 & 3.4)	MC68010, implemented on a Motorola MVME 133A-20 board with a MC68881 floating point coprocessor (bare machine) *Derived*
Compiler version 1.2	Sun Microsystems Sun-3/280 Workstation (under Sun UNIX version 4.2, release 3.2)	MC68020, implemented on a Motorola MVME 133-A-20 board with a MC68881 floating-point coprocessor (bare machine)

Note: Although a response was not provided for these targets, a response for a bare MC68020 was provided, and it is reasonable to expect similar capabilities (except performance) for these implementations. Refer to TeleSoft's bare MC68020 target processor response.

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Verdix Corp.	Sequent Symetry S-27(under DYNIX, release 3.0)	80386, iSBC 386/20P Intel(bare machine) using file-server support from the Host
	Intel system 320 (under UNIX system V rel 3.0)	80386, iSBC 386/20 Intel(bare machine)

Note: Although a response was not provided for these host/target combinations, a response for a bare 80386 (with a different host) was provided, and it is reasonable to expect similar capabilities for these implementations. Refer to Verdix's bare 80386 target processor response.

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

This information was not supplied by the vendor.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guidelines to Select, Configure and Use an Ada Runtime Environment

COMPILER VENDOR	HOST PROCESSOR	TARGET PROCESSOR
Verdix Corp. Compiler version 5.42	MicroVAX II (under MicroVMS, Version 4.4)	32032, National DB32000 (NS32032) using file- server support from the Host (bare machine)
Compiler version 5.42	VAX 8800, 87000 8650, 8600, 8500, 8300, 8200 VAX 11/785, 782, 780, 750, 730, & MicroVAX II (under VMS 4.4)	32032, National DB32000 (NS32032) using file-server support from the Host (bare machine) *Derived*
Compiler version 5.42	SYS32/20 (under Opus5 (UNIX System V), release 2.0)	32032, National DB32000 (NS32032) using file- server support from the Host (bare machine)

DEGREE OF CONFIGURABILITY

This information was not supplied by the vendor.

PIWG RESULTS

This information was not supplied by the vendor.

RUNTIME STORAGE REQUIREMENTS

This information was not supplied by the vendor.

Guideline to Select, Configure, and Use an Ada Runtime Environment

5. Application Characteristics

The first part of this section describes requirements that can be imposed upon application software. The requirements can be chosen from the following list:

- 1). Real-time Response,
- 2). Hardware Interfacing,
- 3). Fault-tolerance,
- 4). Distributed Processing,
- 5). Multi-level Security,
- 6). Concurrency,
- 7). Periodic Processing,
- 8). Numeric Accuracy,
- 9). Continuous Operation,
- 10). Message Processing,
- 11). High Throughput Rate,
- 12). Program/Data Size Limitations.

The second part of this section will partition the application domain into various classes. The classification is not meant to be exhaustive, but rather characteristic. Then, the requirements outlined above can be mapped into the classes, along with the runtime components necessary to implement each one. This list will be prioritized based upon:

1. LabTek's experience.
2. Interviews with application engineers. (LabTek performed extensive interviews with application engineers under a previous contract. [1])
3. ARTEWG's, First Annual Survey of Mission Critical Application Requirements for Runtime Environments. [3]
4. Current literature.

Figure 3., "The Application Domain", presents the classes of applications that will be covered in detail. For each class of application the following information will be provided:

1. Brief description of the class.
2. Typical system requirements for an application of this nature.
3. A prioritized list of the runtime environment features needed for this class of applications.

Guideline to Select, Configure, and Use an Ada Runtime Environment

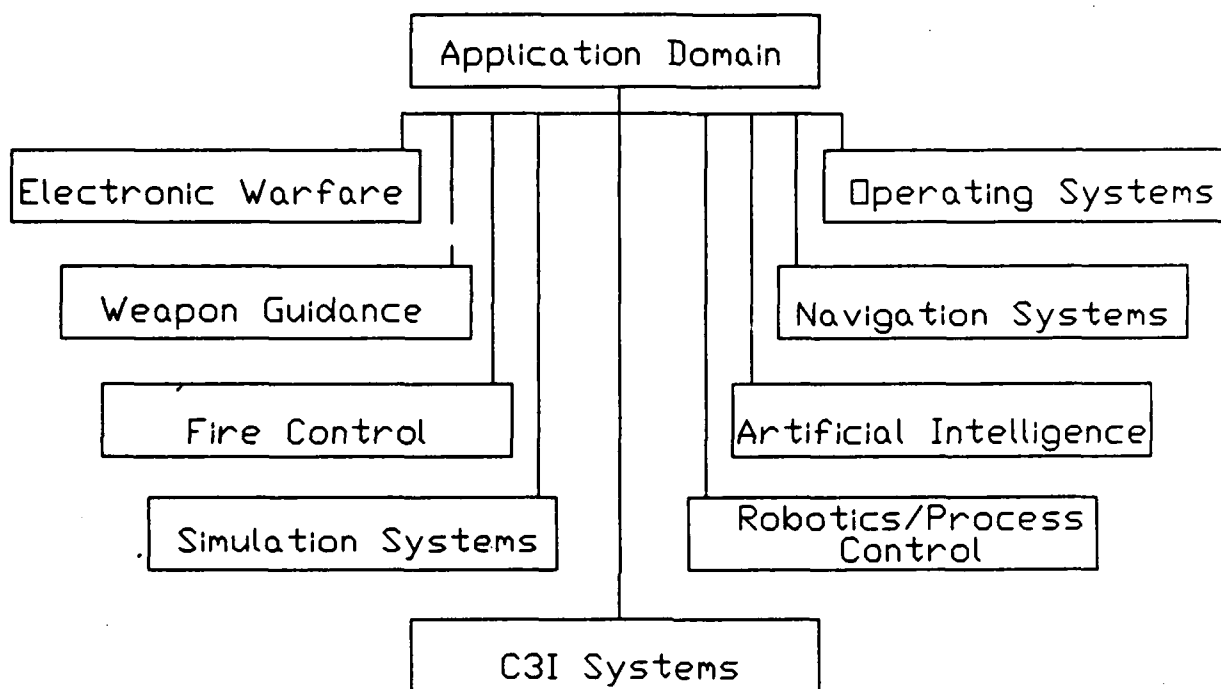


Figure 3. The Application Domain

5.1 Electronic Warfare

Brief description: EW systems are used to transmit and/or receive electronic or electro-optical radiation, usually for observation or communication; or to deny an adversary use of their electronic systems.

System requirements: Refer to subheadings below.

Runtime Environment Features Required (Prioritized): Refer to subheadings below.

5.1.1 Radar Systems

Brief description: Transmission of electronic signals which reflect off of objects and return to a sensor for the purposes of detecting and locating objects.

Guideline to Select, Configure, and Use an Ada Runtime Environment

System requirements: Real-Time Response, Hardware Interface, Distributed Processing, Concurrency, Periodic Processing, Numeric Accuracy, Message Processing (see also "Signal Processing" below).

Runtime Environment Features Required (Prioritized): Interrupt Management, Rendezvous Management, Time Management.

5.1.2 Electronic Counter Measures (ECM)

Brief description: ECM systems vary considerably. In general, they are designed to deny an adversary use of their communications or surveillance systems by transmitting radiation which "jams" the adversary receivers. They may be used in conjunction with Electronic Support Measures (ESM) which is used to receive and classify the adversary's transmissions. Electronic Counter Counter Measure (ECCM) systems in-turn are used to nullify the effect of ECM systems.

System requirements: Real-Time Response, Hardware Interface, Distributed Processing, Concurrency, Periodic Processing, Numeric Accuracy, Message Processing (see also "Signal Processing" below).

Runtime Environment Features Required (Prioritized): Interrupt Management, Rendezvous Management, Time Management.

5.1.3 Signal Processing

Brief description: Signal Processing systems are used to process the raw (digitized) return data received from sensor inputs. Typically data is received at rates exceeding one million bytes per second (1MB/s) and must be processed in very short periods of time. Custom processors are used to implement quick Multiply/Accumulate operations in support of Fast Fourier Transforms (FFTs), Convolutions, and other algorithms used to reduce or transform input data to meaningful values.

System requirements: High Throughput, Real-Time Response, Distributed Processing, Periodic Processing.

Runtime Environment Features Required (Prioritized): Processor Management, Commonly Called Code Sequences.

5.2 Weapon Guidance

Brief description: Weapon guidance can be autonomous (self-guided) or in conjunction with platform-based support. In either case, the intent is to direct the in-flight weapon to a (possibly moving) target. Target tracking and prediction, attitude control, and servo loop processing are included as major portions of the system. If the system includes a platform component, message processing would also be involved.

System requirements: Real-Time Response, High Speed Interface, Concurrency, Periodic Processing, Program/Data Size Limitations, Message Processing.

Runtime Environment Features Required (Prioritized): Interrupt Management, Time Management, Rendezvous Management, I/O Management.

Guideline to Select, Configure, and Use an Ada Runtime Environment

5.3 Fire Control

Brief description: Fire Control systems determine the elevation, azimuth, and range values for ballistic trajectories. In fully automatic systems, it includes controlling actuators to position the weapon aimpoint.

System requirements: Real-Time Response, Numeric Accuracy.

Runtime Environment Features Required (Prioritized): Interrupt Management, I/O Management, Commonly Called Code Sequences, Time Management.

5.4 Simulation Systems

Brief description: Simulation systems (or more correctly, emulators) are used to provide the actions of the systems they emulate without having the expense or schedule delay of the real system. They are typically used to test equipment that would normally connect to the simulated device, or to provide early feedback to designers on how a system will behave under controlled circumstances.

System requirements: Real-Time Response, H/W Interface, Distributed Processing, Concurrency, Periodic Processing, Numeric Accuracy, Message Processing, High Throughput.

Runtime Environment Features Required (Prioritized): Time Management, Interrupt Management, Rendezvous Management, I/O Management, Dynamic Memory Management, Processor Management.

5.5 C3I Systems

Brief description: Command, Control, Communications, and Intelligence systems typically are used to assist in battle management at various levels, from front-line troops to top-level officers.

System requirements: Fault Tolerance, Distributed Processing, Multi-Level Security, Concurrency, Continuous Operation, Message Processing. Space based C3I systems frequently have Program/Data Size Limitations as well.

Runtime Environment Features Required (Prioritized): I/O Management, Rendezvous Management, Dynamic Memory Management, Time Management, Exception Management, Task Activation, Task Termination.

5.6 Operating Systems

Brief description: Operating Systems control the execution of programs running on one or more processors.

System requirements: H/W Interface, Fault Tolerance, Distributed Processing, Multi-level Security, Concurrency, Message Processing.

Runtime Environment Features Required (Prioritized): Processor Management, I/O Management, Dynamic Memory Management, Task Activation, Task Termination, Target

Guideline to Select, Configure, and Use an Ada Runtime Environment

Housekeeping Functions, Exception Management, Interrupt Management, Time Management.

5.7 Navigation Systems

Brief description: Navigation systems determine the current position, direction, velocity, and acceleration of the vehicle in which they are contained.

System requirements: Real-Time Response, Periodic Processing, Numeric Accuracy, Message Processing.

Runtime Environment Features Required (Prioritized): Interrupt Management, Rendezvous Management, I/O Management, Commonly Called Code Sequences.

5.8 Artificial Intelligence

Brief description: AI systems mimic human behavior to a greater degree than typical computer based systems.

System requirements: High Throughput, Distributed Processing.

Runtime Environment Features Required (Prioritized): Dynamic Storage Management, Commonly Called Code Sequences, I/O Management for Data Base driven systems, and Processor Management for parallel systems.

5.9 Robotics/Process Control

Brief description: Robotics and Process Control systems are characterized by having the ability to physically manipulate the environment in which they are located.

System requirements: Real-Time Processing, H/W Interface, Distributed Processing, Concurrency, Numeric Accuracy, Continuous Operation, Message Processing.

Runtime Environment Features Required (Prioritized): Interrupt Management, Rendezvous Management, I/O Management, Processor Management, Time Management, Exception Management.

Guideline to Select, Configure, and Use an Ada Runtime Environment

6. Guidelines

This guideline contains the following three sections: *To Select* a runtime environment, *To Configure* a runtime environment, and *To Use* a runtime environment.

6.1 To Select a Runtime Environment

This section contains a checklist of questions an application developer should consider *before* actually making the commitment to use a particular runtime environment for a given application. Once the checklist has been refined to address the system requirements, the compiler vendor should be approached for the specifics. The "*Ada-Europe Guidelines for Ada Compiler Specification and Selection*" [7] was used as a starting point for this information. It was augmented to include additional considerations specific to real-time applications.

6.1.1 Documentation

Complete and accurate documentation is essential for real-time systems development. Often predictability is just as important as performance. A software managers' greatest fear is discovering an undocumented "feature" of some vendor supplied software which prevents progress. Projects can be delayed weeks or even months while trying to isolate subtle interactions of an executive and finding a solution once the anomaly is understood. Typical documentation may consist of a selection of the following:

- a.) Installation Guide
- b.) Appendix F of the reference manual
- c.) User Reference Manual
- d.) Target Reference Manual
- e.) and/or Runtime Configuration
- f.) Runtime Libraries
- g.) Program Libraries
- h.) Implementation Details
- i.) Ada Compiler Validation Summary Reports (usually not supplied by the compiler vendor, available through NTIS)

Which documents listed above are available to the user?

Is there a separate charge for specific documentation?

6.1.2 Degree of Configurability

Before actually selecting the compilation system to be used for a particular application it is important to know its degree of configurability.

Questions to ask:

What features are configurable (ie. interrupts, runtime initialization, runtime libraries, tasking algorithm, timer resolution, etc.)? A more extensive list of configurable components is provided under the section "To Configure a Runtime Environment" below.

Can the runtime system be configured by the user through a linker?

Guideline to Select, Configure, and Use an Ada Runtime Environment

Are there any tools or components available which assist in customizing the runtime library?

Is the configurable data used by the linker provided to the user for review, modification, and configuration management?

Is the implementation of the runtime features organized so that some modules of the runtime can be excluded from the application's execution image if the application does not require features implemented by those modules?

Is the documentation adequate?

6.1.3 Chapter 13

It is important to find out to what extent features of Chapter 13 are implemented by the compiler on the target machine.

Questions to ask:

Are there any restrictions on the representation clauses and implementation dependent features such as:

- a.) Length Clauses
- b.) Enumeration Representation Clauses
- c.) Record Representation Clauses
- d.) Address Clauses
- e.) Interrupts (entry address clauses)
- f.) Size Representation Attributes
- g.) Fixed Point SMALL Attributes
- h.) Machine Code Insertions
- i.) Interface to Other Languages
- j.) Unchecked Storage Deallocation
- k.) Unchecked Type Conversions

Does the compiler support fast interrupts (i.e., restricted interrupt entries)?

6.1.4 Appendix F

The reference manual of each Ada implementation must include an appendix (called Appendix F) that describes all implementation-dependent characteristics. [5]

Questions to Ask:

What is the form, allowed places, and effect of every implementation-dependent pragma?

What are the names and types of every implementation-dependent attribute?

What is the specification of the package SYSTEM?

What conventions are used for any implementation-generated names denoting implementation-dependent components?

Guideline to Select, Configure, and Use an Ada Runtime Environment

What is the interpretation of expressions that appear in address clauses, including those for interrupts?

Are there any implementation-dependent characteristics of the input-output packages?

6.1.5 Target Dependent Information

Target dependent information is needed by the semantic analysis part of an Ada compiler. Examples include the size of INTEGER type, and the memory model. It can be supplied by:

- a.) recompiling packages STANDARD and/or SYSTEM
- b.) a parameter file, or command line options
- c.) incorporating a package which is linked into the compiler,
- d.) being built into the compiler.

Questions to ask:

How is target dependent information supplied to the compiler?

How easy is it to change for a new target?

What is the specification for package STANDARD?

6.1.6 Target Initialization

Target dependant actions might be required to initialize the target following power-up or a reset-sequence. This would be modified for targets that have different initialization requirements. Some requirements might pertain to:

- a.) Processor Mode
- b.) Interrupt Enable
- c.) Memory Management Setup
- d.) Co-processor Setup

Questions to ask:

What configuration parameters are supplied with the compiler?

What assumptions (if any) are made about the target's state before initialization?

What default declaration, and purpose, does each parameter have?

What steps are followed when initializing the target machine?

Does the Ada runtime depend upon static initialization (parameters fixed at link time), and if not how does this effect ROM and RAM?

Are there configuration routines for which user implemented routines may be substituted?

6.1.7 Target I/O

These types of problem usually do not manifest themselves in obvious ways, but rather result in working but unreliable systems. They may pass the acceptance testing and operate properly for months only to fail in a catastrophic fashion during a critical moment.

It is hoped that this guide will assist software developers through some of the problems in adopting Ada for real-time embedded projects. By providing information on how Ada implementations operate, there will be a reduction in the uncertainty associated with switching from assembly language executives, where every aspect is provided in minute detail, to Ada where the executive functions appear as a black box (or magic).

Guideline to Select, Configure, and Use an Ada Runtime Environment

Target I/O might be modified when targets use a different I/O device than originally supplied by the compiler. Devices which may vary for different targets are:

- a.) Serial ports
- b.) Parallel ports
- c.) Monitors
- d.) Disk drivers

Questions to ask:

What I/O devices does the compiler support? (i.e. Serial I/O, Parallel I/O)

Are the standard I/O devices and file systems functional on embedded systems without change? If so, what packages are affected by this and what are their limitations?

Is there a facility available which allows the I/O packages to give a program running on the target system access to the host file system?

6.1.8 Target Timer

The target timer might be modified when targets use a different timer device or the timer device is located elsewhere than the compiler originally supplied. Timer devices might vary in the following way:

- a.) Configuration
- b.) Timer interrupts
- c.) Tick Interval

Questions to ask:

Is the timer configuration necessary on the bare machine?

If no references to time are made within an application, can all time-related runtime code be eliminated?

How is the timer interrupt routine declared and is it capable of modification?

6.1.9 Data Representation

The compiler vendor should provide details as to how the various Ada types are represented. This is especially important if other languages are to be used with Ada or if special I/O routines are to be written. Data representation can be categorized under the following headings:

- a.) Addressing Structure (segmented, linear, paged)
- b.) Alignment Restrictions (word; byte)
- c.) Type Implementation (i.e. Character, Integer, Boolean, Floating Point, Enumeration, Access, and Record)

Guideline to Select, Configure, and Use an Ada Runtime Environment

Questions to ask:

What is the mapping of scalar types and subtypes?

What is the mapping of arrays (column major order)?

What is the mapping of records without discriminants?

What is the mapping of records with discriminants?

What is the effect of having arrays depending on discriminants? (Is the heap used for such objects under certain circumstances?)

What is the mapping of access types?

What is the effect of pragma PACK?

What predefined types are supported (i.e. LONG_INTEGER, SHORT_INTEGER)?

What are the rules governing the conversion to and from floating point types (i.e., rounding)?

What are the calling conventions of subprograms and how are non-scalar parameters passed?

6.1.10 Implementation of Tasking

A task logically operates in parallel with other parts of a program. It is written as a *task* specification (which specifies the name and formal parameters of its entries), and a *task* body which defines its execution. A *task unit* is one of the kinds of *program unit*. A *task type* is a *type* that permits the subsequent *declaration* of any number of similar tasks of the type. A value of a task type is said to *designate* a task. [5] Some facilities which are provided for the implementation of tasking are:

- a.) Task Creation
- b.) Queuing
- c.) Timing
- d.) Scheduling
- e.) Task Dispatching
- f.) Rendezvous
- g.) Termination

Questions to ask:

What is the storage management technique used (e.g. acquisition of stack and heap space for a new task) in a multitasking program?

What is the method of implementing the Ada rendezvous mechanism? For example, an Ada runtime kernel or monitor may be defined, or the implementation may rely on target operating system facilities.

Guideline to Select, Configure, and Use an Ada Runtime Environment

In the absence of latencies due to application software, what is the guaranteed accuracy of the delay statement, and what application characteristics can alter this basic accuracy?

Is tasking supported on multiprocessor architectures?

What is the method used for associating external interrupts with task entries?

What tasking optimizations are possible and what are the circumstances under which they can be achieved?

6.1.11 Interrupt Handler/Interrupt Vectors

External events and incoming data are typically handled by interrupt handlers. The interrupt vector directs the transfer of control to the appropriate handler.

Questions to ask:

What mechanisms are supported for application software to handle an interrupt?

How do the interrupt vectors used by the runtime get initialized?

Are software interrupts (traps) supported in the same way as hardware interrupts?

What restrictions are imposed on interrupt handlers with regard to accessing data outside the local scope of the handler?

6.1.12 Storage Management

The following are four classes of storage allocation classifications for a typical Ada program.

- a. Access - an access value is either null or refers to an object created by an allocation (storage is typically allocated from the heap).
- b.) Local - used for objects that are declared in subprograms or in packages nested within subprograms. Local storage is typically allocated from the runtime stack.
- c.) Static Allocation - Used for objects only when they exist throughout the ENTIRE execution of the program which contains the objects.
- d.) Temporary - Used for non-scalar values resulting from expressions or functions. Such storage is allocated from the runtime stack or heap when needed and released once the value is discarded or assigned.

Questions to ask:

What is the primary stack management technique being used? (Main program stack.)

Is there a secondary stack management being used? (Dependent task stacks.)

Guideline to Select, Configure, and Use an Ada Runtime Environment

What is the method of allocating and deallocating storage for tasks?

How are access type collections managed?

How is the heap managed?

What is the storage reclamation (garbage collection) technique used?

What is the runtime system storage size?

6.1.13 Subroutine Call and Parameter Passing Conventions

The method used for passing parameters (especially for calling non-Ada subprograms) must be known. Some calling conventions and parameter handlings are:

- a.) Call Site - An example is when the responsibility lies on the user to strip parameters from the stack upon a return. (calling conventions)
- b.) Stack Frame & Prologue/Epilog Conventions - This pertains to the direction that the stack grows, the position of the frame pointer, and the position of the stack pointer. (calling conventions)
- c.) Descriptor Block - parameters are accessed indirectly through a descriptor table (parameter passing convention).
- d.) The representation of Boolean, Fixed Point, and Floating Point arguments. (parameter passing conventions)

Questions to ask:

What is the parameter passing method used?

What is the mechanism used for returning results from a function (especially where the result is a record or unconstrained array type)?

6.1.14 Saving Machine State During a Context Switch

Depending on the application, registers and floating point coprocessor context may or may not be saved and must be changed as required.

Questions to ask:

Under what circumstances do the registers get saved in a context switch, especially floating point registers?

6.1.15 Exception Handling

To raise an exception is to abandon normal program execution so as to draw attention to the fact that the corresponding situation has arisen. Executing some action, in response to the arising of an exception, is called handling the exception. The compiler vendor should

Guideline to Select, Configure, and Use an Ada Runtime Environment

describe the mechanism for finding the appropriate handler when an exception is raised. [5]
Exception handling is comprised of the following:

- a.) Raise Action - Result of a raise statement or failed runtime check (i.e. overflow)
- b.) Trap Action - A hardware initiated exception
- c.) Exception Propagation

Questions to ask:

What is the exception identification scheme used? (How are exception numbers allocated?)

What is the mechanism used for exception handling?

What is the overhead associated with using exception handling?

What is the relationship with hardware and host operating system exceptions?

6.1.16 Unhandled Exceptions

A procedure which is a "last resort" handler should be provided for exceptions that are propagated out of an Ada program. This provides a useful debugging tool with compilation systems that provide a sparse-level trace back.

Questions to ask:

Does the compiler provide debugging and diagnostic messages when an exception causes program termination?

6.1.17 Generics

A generic unit is a template either for a set of *subprograms* or for a set of *packages*. A subprogram or package created using the template is called an *instance* of the generic unit. A *generic instantiation* is the kind of *declaration* that creates an instance. A generic unit is written as a subprogram or package but with the specification prefixed by a *generic formal* part which may declare *generic formal parameters*. A generic formal parameter is either a *type*, a *subprogram*, or an *object*. A generic unit is one of the kinds of *program unit*. [5]

The compiler vendor should describe the runtime implications of generics.

Questions to ask:

What are the circumstances under which it is possible to share code between two different generic instantiations? Is any user control available?

Is there any additional object code or data requirements imposed by the use of generics (especially when code sharing is in use)?

Are there any limitations to the compilation of generic units?

Guideline to Select, Configure, and Use an Ada Runtime Environment

6.1.18 I/O Interfaces

The compiler vendor should describe the I/O which the runtime supports. The facilities that would be supported would be found in:

- a.) Package Direct_IO
- b.) Package Sequential_IO
- c.) Package Text_IO
- d.) Package Low_Level_IO

Questions to ask:

What is the functionality of the above listed I/O packages with nonstandard I/O devices?

What are the limitations of the above listed I/O packages with nonstandard I/O devices?

Is an interface to the target provided for Low_Level_IO?

Is formatted I/O available?

Is binary I/O available?

Are there any restrictions on types that can be instantiated for input-output?

6.1.19 Compiler Capacity and Tool Availability

When selecting a compilation system it is important to investigate the other tools available, such as: design tools, library management, configuration management, and debugging. An integrated toolset is best.

Questions to Ask:

What are the capacities of the compiler (i.e. number of nested loops, number of nesting levels in procedures which are separately compiled, number of variables allowed)?

What are the known compiler deficiencies?

Are design tools provided?

Are configuration management tools provided?

Are library management tools provided?

Are debugging tools provided?

Are test support tools provided?

What process must be followed for the inclusion of the runtime system into an application?

Are there any tools or components available which assist in customizing the runtime library?

6.2 To Configure a Runtime Environment

Although the instruction set of a particular architecture provides a level of standardization, features such as main memory size, method of I/O, memory management, floating point coprocessors, and power up sequences are necessarily machine dependent. For this reason, the user will probably need to make certain alterations to the runtime support provided to fit a particular environment.

This section will present the features that may be configured into a runtime. However, not all vendors provide the same degree of configurability, or provide the same methods of configuring the components. For example, there are essentially three configuration mechanisms:

1. User modifies vendor supplied runtime routines (and recompiles, reassembles, as necessary). Usually there are two parts to the runtime, a part the user can configure, which is referred to as user-configurable, and a permanent part, or non-configurable part of the runtime. The user-configurable code is typically a set of assembly language routines, called from the permanent part of the RTS and generated code. Calls to the user-configurable code can be made directly from an Ada program via pragma INTERFACE. [10] A set of rules or conventions must be followed to ensure compatibility between the user-configurable and non-configurable parts of the runtime.
2. Linker options. These are switches on the link phase and can span the gamut from very simple to complex. They will be discussed in more detail below.
3. Vendor supplied pragmas.

Configuration of the RTE takes place *after* the application developer has selected a compilation system. Hopefully, the compiler selection was performed with the system requirements in mind. Section five of this report details typical system requirements. The list is representative rather than exhaustive, therefore, any given system will most likely have some combination of the requirements detailed there.

It is important to obtain all of the documentation for the selected compilation system. Vendors provide a separate manual titled "Target Handbook", or "Runtime System Configuration Guide". A technical person to contact at the vendor site is often very helpful. This may require a maintenance contract with the vendor, which is generally recommended.

Features that typically need configuring are: bootstrap (power up) sequences, timers, interrupt vectoring, main memory configuration parameters, and method of doing I/O. These are typically machine-dependent and the user will probably need to make certain alterations to the runtime support to fit a particular environment.

6.2.1 Bootstrapping

The start up code (or bootstrap code) is the module (or modules) invoked when the system is reset. Its primary function is to initialize the stack, transfer control to the RTS execution code, and possibly terminate the program upon completion. Other initializations performed at this time may include:

Guideline to Select, Configure, and Use an Ada Runtime Environment

- Initialization of the main program stack.
- Initialization of primitive storage management data structures.
- Hardware initialization.
- Initialization of I/O routines.

6.2.2 Interrupt Vector

The interrupt vector table must be established before the execution of Ada code begins. The table can be initialized by:

- Placing the preset values of the interrupt vector with user values (via a table).
- Explicitly initializing the values with user written code. This code must be called or executed by the start up code.
- Specifying address clauses for task entries in the Ada program.
- Provide default handlers for all vectors.

The first method is typically used in a bare system, the second method used with an underlying operating system.

6.2.3 User-Configurable Module Dependencies

For each user-configurable module, the following must be known:

- Its name as known to the non-configurable runtime modules.
- Any input parameters required.
- Any output parameters returned.
- Any side effects resulting from its usage.
- Any user-configurable runtime module dependencies.
- Any non-configurable runtime module dependencies.

Special care should be made to preserve registers, especially the interrupt status.

6.2.4 Timer Interrupt

The timer interrupt has many uses. Depending on how it is implemented, it can be used:

- to allow user direct access to the timer interrupt
- by package CALENDAR
- by tasking/time-slice scheduling
- by the `delay` statement in a program that uses tasking.

Resolution of timer interrupts is usually programmable. Be advised that if timer interrupts are set to occur frequently that the overhead for this function could be substantial. For example, if interrupts are programmed for every millisecond, and the clock interrupt routine takes 300 microseconds, 30% of the CPU time will be spend servicing the timer interrupts. Since timer interrupts are often a high priority, care must be taken to insure that lower level interrupts will not be deferred beyond their deadline by a timer interrupt.

6.2.5 Linker Options

Linker switches/parameters could be used to specify:

Guideline to Select, Configure, and Use an Ada Runtime Environment

- whether or not a floating point coprocessor is to be used in the system.
- the total amount of memory available.
- the amount of storage to be allocated to the system heap.
- the amount of storage allocated to the collection area.
- the amount of storage allocated to the stack area.
- the minimum size of an element on the heap.
- the maximum number of active tasks. This would be used to determine the amount of storage required by the task control blocks.
- a pointer which designates the beginning of the task control block area.
- whether time-slicing is to be enabled.
- the length of time for the time-slice interval.
- the default size of a library task stack. A library task is a task where the task body is declared in a package at the outermost level.
- the main program stack size.
- the default priority for all tasks, that can be overridden by the pragma PRIORITY.
- the length of time between timer interrupts and the resolution of the delay timer.
- the lowest interrupt permitted in the Ada code for standard interrupt tasks, i.e., an interrupt entry defined in an address clause.
- the highest interrupt accessible in the interrupt vector table for Ada code using standard interrupt tasks. i.e., an interrupt entry defined in an address clause.
- the range of words reserved for the interrupt vector table used for Ada standard interrupt tasks, i.e., and interrupt entry defined as an address clause.

Special care must be taken to insure that the switches used during linking are maintained by configuration management. Typically this is achieved by performing the link by using a command file which is placed under configuration management control.

6.3 To Use a Runtime Environment

There are two essential aspects to using an Ada RTE. These are communicating the characteristics of a configured RTE to the software designers and maintaining the configuration for future releases. Since the behavior of an Ada program can vary tremendously based on the composition of the RTE, it is crucial that the dependencies between the application program and the RTE be well documented.

The documentation of the configured RTE should be in the form of a user's guide which would provide details on how RTE features are supported, and Ada source comments which indicate the dependence on those features within the application program. The user's guide should be required reading for all software designers, and should assist them in their analysis and program architecture. For example, if the RTE implements a time-sliced based scheduler, this design decision is likely to have a major impact on the software architecture of a real-time system.

Once the software is designed, the dependencies on the particular underlying RTE should be well documented in the form of regular comments that can be automatically extracted from the source code. Typical methods suggest the use of specific keywords in the embedded PDL (program design language) such as:

`--/Requires: Automatic Storage Reclamation", or`

`--/Assertion: Priority-based preemption guaranteed within 500 microseconds".`

Guideline to Select, Configure, and Use an Ada Runtime Environment

These dependencies should obviously be summarized in the "Software Detailed Design Document (SDDD)".

The second aspect: RTE configuration management, imposes very serious considerations on an Ada software development. Issues such as validation, reliability, maintainability, and liability, complicate real-time applications that require modifications to the RTE. The main overriding concern is guaranteeing that subsequent "builds" of a system have the proper RTE composition. At a minimum, this implies that the RTE generation/configuration process is somewhat automated and the RTE is generated using this process prior to critical system builds.

Ideally, the RTE build process should: a.) document all of the configuration parameters (including versions of all included components), b.) provide a unique serial number for the documentation file, c.) and incorporate the serial number in the binary image of the RTE. This would permit backtracking of binary to the source level composition.

The configuration issue is further complicated by design changes that require different RTE composition and/or new maintenance releases of the vendor supplied RTE. The scope of all RTE changes must be well understood, and reviewed with respect to the application program RTE dependency summary discussed above.

The interaction between application programs and the Ada RTE can introduce anomalies that manifest themselves in insidious ways. This dictates extreme care when making any RTE modifications. Experience has provided numerous examples of programs that crash after several days of seemingly flawless execution; or programs that deadlock at apparently random intervals because of small changes made to the RTE. Clearly, RTE management should be performed only by the most competent personnel on the development team.

Guideline to Select, Configure, and Use an Ada Runtime Environment

7. Effects of Runtime Issues on the Development of Reusable Software

The flexibility allowed in Ada runtime implementations makes it possible to solve a wide variety of problems with the Ada language. However, this flexibility usually introduces serious compatibility problems when attempting to reuse software, especially in real-time embedded applications. These applications have stringent timing requirements which make their correct execution much more sensitive to the implementation approach. For example, a queue manager may depend on access types or the raising of an exception on an access check. The overhead associated with allocating and deallocating storage, and in raising exceptions can vary substantially among different runtime configurations. The impact is that a "reusable" software component that works well on one runtime configuration may have very poor performance on another configuration.

In cases where storage for allocators is not reclaimed (for performance reasons), some software components may cause the entire system to fail unexpectedly due to a storage error. The error may even occur in another subprogram that ran out of storage because the "reused" component allocated too much storage. This makes isolating the problem very difficult because many subprograms share a common resource (heap memory).

The clear implication is that "reusable" components may not use any implementation dependent features of the language. This is, of course, impossible to achieve if efficiency is a concern. What is required instead is "configurable reusable components". This can be achieved by providing a large collection of components, where the appropriate version can be selected for each particular application. Given the number of variants in the runtime, it is impractical to supply all possible permutations, but rather some mechanism should be supplied to combine characteristics of components. To a large extent, this may be achievable using generics. It may also be useful to have a tool which would "build" the correct component based on supplied specifications of requirements. That is, by supplying the tool with the characteristics of the configured runtime, it could select the combinations of reusable components that match the runtime capability. In the above example where no storage reclamation is done, the component "build" tool would not select a reusable component that is likely to allocate and deallocate storage continuously throughout program execution.

Reusability for real-time software is generally improved as the available processing capacity (including memory size) greatly exceeds what is required for an optimal implementation. This tends to eliminate the fine tuning that is typically required in real-time applications. To the extent that the cost effectiveness of reusing software outweighs the additional cost of hardware (processors/memory), future systems may find reuse more attractive for embedded systems. Clearly, design approaches which allow additional processing capacity to be effectively utilized (the addition of more processors) lend themselves to being able to accept some performance penalty for using non-custom software. In return for the additional hardware cost, the development and maintenance costs are likely to be far less. The tradeoff must be made on a case by case basis and will depend largely on the number of systems to be produced.

Guideline to Select, Configure, and Use an Ada Runtime Environment

8. Summary

The period of performance for this contract spanned nine months. The information contained in this report represents the state of the technology at the time it was issued. Reports of this nature can become outdated if not maintained, but it is felt that even though a particular version of the compiler presented may become obsolete, important information can be learned from the contents of this report for at least the next few years. For example, the guidelines section will not change dramatically until the technology changes dramatically. It is believed this section will remain valid for about five years.

A report of this nature was necessary because of the void of information supplied by vendors regarding the runtime specifics. Users were having difficulty getting detailed information on Ada implementations. As a result, they frequently selected compilation systems that did not match their application requirements. The difficulties they had, due to the poor match, produced a bad image for Ada, in general.

Providing information on compilers will help alleviate the problems and promote the wide use of Ada, and that was the intent of this report. For the most part, it was difficult to obtain the runtime information from the vendors. Although it is recognized that their main function is to produce a "production quality" compiler, this information must be readily accessible for the user to choose a compiler implementation which suits a given application. Often it took repeated phone calls and contact with a technical person to get the proper answer. In fairness to the vendors, some were very helpful and informative.

Compilation systems are maturing, and this can be seen from the issues the compiler vendors are now addressing. A few years ago, the push was towards validation. Now that validation has been achieved for most vendor products, optimizations and Chapter 13 features are being addressed, as well as tasking problems and runtime system variants. A runtime system variant may contain such desired features as semaphores, mailboxes, different tasking schedulers, etc., which are not part of the Ada language standard, but are often required in real-time applications.

In general, configurability of the runtime system is being addressed by the vendors, runtime sizes are decreasing, generated code quality (due to optimizations) is improving, and Chapter 13 features are being implemented. All of these are features that users have waited years for with great anticipation.

Other difficulties included obtaining the AVO reports for a given compilation system. The current mechanism to obtain validation reports is unworkable. Because of multiple validation sites and long delays between validation and the availability of the report from the National Technical Information Service (NTIS), it is extremely difficult to obtain the reports on compilers of interest. One way to avoid this might be to have all information contained in the validation report maintained in machine readable form in a very regular format. This information should be forwarded to the Ada Information Clearinghouse, which would then be responsible for placing it in a directory on the AJPO host.

Finally, a serious potential customer should contact the AdaIC directly to obtain the most recent validation information, rather than relying solely on the published AdaIC listing of validated compilation systems. The AdaIC listing changes monthly and the printed listing

Guideline to Select, Configure, and Use an Ada Runtime Environment

lags behind the actual validated compiler status. The AdaIC listing does provide a number for the user to call to check on the most recent status.

Guideline to Select, Configure, and Use an Ada Runtime Environment

9. References

- [1] "Software Engineering Issues on Ada Technology Insertion for Real-time Embedded Systems", final report delivered to Center for Software Engineering, CECOM, by LabTek Corporation, September 30, 1987.
- [2] Ada Runtime Environment Working Group of ACM SIGAda, "A Framework for Describing Ada Runtime Environments", October 15, 1987.
- [3] Ada Runtime Environment Working Group of ACM SIGAda, "First Annual Survey of Mission Critical Application Requirements for Runtime Environments", December 1, 1987.
- [4] Ada Letters, "Ada Compiler Validation Procedures and Guidelines", ACM SIGAda, Volume VII, Number 2, March, April 1987.
- [5] ANSI/MIL-STD-1815A-1983. "Reference Manual for the Ada Programming Language", American National Standards Institute, Inc., 1983.
- [6] The Info-Ada Newsletter, Volume 2, Issue 2, February 1988, Volume 2, Issue 3, March 1988, Vol. 2, Issue 8, August 1988.
- [7] Nissen, PJC, Wichmann, BA, and others, "Ada-Europe Guidelines for Ada Compiler Specification and Selection", Ada Letters, Volume III, Number 5, March, April 1984.
- [8] "An Approach to Tailoring the Ada Runtime Environment", interim report delivered to Center for Software Engineering, CECOM, by IIT Research Institute, 1988.
- [9] "Real Time Performance Benchmarks for Ada", interim report delivered to Center for Software Engineering, CECOM, by Technical Management and Service Corp., 1988.
- [10] DDC-I Ada Compiler System, Run-Time System Configuration Guide for DACS-80x86, Document No: DDC-I 5801/RPT/66 Issue 3, DDC-I Ada Compiler System User's Guide for DACS-80x86, Document No.: DDC-I 5801/RPT/62, Issue 9, DDC-I, Inc., Phoenix, AZ, 1988.
- [11] User Manual, Tartan Ada VMS/1750A, Version Number V1.0, Tartan Laboratories Inc., Pittsburgh, PA, 1987.
- [12] System Designers' Ada-Plus VAX/VMS, MC68020 Vol 1, 2, and 3, Reference: D.A.REF.AF[BC-MH], Issue 3.0, System Designers, Cambridge, MA, 1988.
- [13] Verdex Ada Development System VADS, Version 5.41 for SUN-3/UNIX => Motorola 68000 Family Processors, Document No. VAda-040-13125, Verdex Corp., VA, 1987.

Guideline to Select, Configure, and Use an Ada Runtime Environment

- [14] Performance Issues Working Group of ACM SIGAda, PIWG Test Suite dated January 1988.
- [15] Alsys PC AT Ada Cross-Compiler for the Intel iAPX86 Family, "Cross Development Guide, Appendix F, Ada Probe User's Guide", Version 3.2, August 1987, Alsys.
- [16] Technical Summary for Alsys Cross Compilation System for Intel 80x86, Version 3.21, July 11, 1988, Alsys.
- [17] Technical Summary for Alsys Cross Compilation Systems for Motorola M680x0 (Version 3.5), July 11, 1988, Alsys.
- [18] AdaIC Newsletter, Vol. VI, No. 2, July 1988. AdaIC Newsletter, March 1988, AdaIC Newsletter, December 1, 1987.
- [19] Technical Specification, Rational R1000 to M68000 Family Cross-Development Facility, Document Control Number: 6001, Rev. 0, November 1986, Rational.
- [20] Technical Specification, Rational R1000 to MIL-STD-1750A Cross-Development Facility, Document Control Number: 6000, Rev. 0, November 1986, Rational.
- [21] Technical Summary, Ada-86, Document: 6027-1, SofTech Inc., Waltham, MA, 1986.
- [22] "4th Annual Directory of Validated Ada Compilers", Defense Science & Electronics, February 1988.
- [23] Appendix F, Implementation-Dependent Language Features, Rational MIL-STD-1750A Cross Development Facility, 6/15/88.
- [24] Appendix F, Implementation-Dependent Language Features, Rational M68000 Cross Development Facility, 4/15/88.
- [25] TeleGen2 User Guide for VAX/VMS to 1750A Targets, UG-1030N-V1.7 (VAX.1750A), TeleSoft, January 1988.
- [26] TeleGen2 User Guide for VAX/VMS to Embedded MC680X0 Targets, UG-1002N-V (VAX.E68), TeleSoft, November 1987.
- [27] User Manual, Tartan Ada Runtime Client Package, Draft 880229.1559, Version 1.0, Tartan Laboratories Inc., 1988
- [28] User Manual, Tartan Expanded Memory Package, Draft 880325.0905, Version 1.0, Tartan Laboratories Inc., 1988

Guideline to Select, Configure, and Use an Ada Runtime Environment

[29] Ada Compiler Validation Summary Report: SofTech, Inc., VAX 11/780 and 11/785 host for Intel iAPX 8086, Intel iAPX 80186, Intel iAPX 80286 real mode, and Intel iAPX 8086 protected mode targets. Ada joint program office 1987.

[30] Ada Compiler Validation Summary Report: System Designers, SD VAX x Motorola M68000/10 Ada-Plus, 2A.00 VAX 8600 host for MC68010 target, June 1986.

[31] Ada Compiler Validation Summary Report: CAP Industry Ltd., CAPTACS-E286, V1.0 DEC VAX 8800 host for Intel 80286 target, December 1986.

[32] Ada Compiler Validation Summary Report: TeleSoft, TeleGen2 E68, Version 3.11 MicroVAX II host for Motorola 68020, 68010, and Tektronix 8540 (M68010 CPU) target, September 1986.

Guideline to Select, Configure, and Use an Ada Runtime Environment

10 Appendix A

The following pages contain the survey submitted to the compiler vendors listed in section 4 of this report.

Guideline to Select, Configure, and Use an Ada Runtime Environment

SURVEY OF RUNTIME ENVIRONMENT COMPONENTS

Instructions: Attached is a list of eleven components of a runtime environment as defined by the Ada RunTime Environment Working Group (ARTEWG) of SIGAda. Please indicate the storage overhead associated with each feature, in K bytes. Sizes should include both code and data, *not* just code storage. We would prefer it if you could adhere to our breakdown of runtime environment components, but if your breakdown is significantly different, use a separate sheet of paper and list each category with its storage requirement. We are interested in the bare machine targets only. If your company provides more than one bare machine target compiler, we would appreciate a response for each target. Just duplicate the questionnaire as needed. Following the runtime environment components section are seven additional questions.

Your responses will be used in a runtime environment study. The purpose of the study is to provide information to the U.S. Army on how to configure a runtime environment. A copy of the report will be placed in the public domain and will be provided to you upon request. All compiler vendors of bare machine targets are being asked for their input. Your response is appreciated.

HOST

TARGET

COMPILER
VERSION

DATE

Guideline to Select, Configure, and Use an Ada Runtime Environment

For each of the following components of your runtime, indicate the storage requirements (both code and data). If your runtime does not support one of the features below, enter 0 for the size.

DYNAMIC MEMORY MANAGEMENT _____Kbytes

Responsible for allocation and deallocation of storage at runtime. Also detects when a request for storage cannot be fulfilled, and for raising the exception `STORAGE_ERROR` as appropriate.

PROCESSOR MANAGEMENT _____Kbytes

Implements the assignment of physical processors to tasks that are "logically executing". The processor management function is invoked by other components of the runtime environment, in order to block and unblock tasks. It keeps a list of those tasks which are "logically executing" and uses this list, in conjunction with the priorities of tasks, to determine which task or tasks should be assigned to processors.

INTERRUPT MANAGEMENT _____Kbytes

Responsible for initialization of the interrupt mechanism of the underlying computing resource, and it is also responsible for resetting that mechanism after an interrupt has occurred, if the architecture of the underlying computing resource requires such resetting.

TIME MANAGEMENT _____Kbytes

Consists of all those portions of the runtime environment that will support the predefined package `CALENDAR` and the implementation of delay statements. If the underlying computing resource offers enough functionality, the support of package `CALENDAR` is trivial.

EXCEPTION MANAGEMENT _____Kbytes

Function implements Ada semantics for exceptions: that is, it determines whether there is a matching handler for the exception at hand, and if there is one, it transfers control to the handler. If there is no matching handler, it invokes the Task Termination function to terminate the task at hand or the main program.

RENDEZVOUS MANAGEMENT _____Kbytes

Implements the semantics of the Ada rendezvous model. In order to do so, it utilizes variables that are internal to the runtime environments. These variables reflect, among other things, which tasks are blocked because they are waiting to rendezvous with other tasks, and what the exact circumstances of these wait states are. The rendezvous management function cooperates with the interrupt management function in the

Guideline to Select, Configure, and Use an Ada Runtime Environment

implementation of interrupt rendezvous, if the interrupt rendezvous is supported by the runtime environment.

TASK ACTIVATION

_____Kbytes

At some point after the task object has been created, the execution of the new task has to be started. This is effected by the task activation function. This function is invoked by the creator of a new task in order to start the new task's activation (which is defined as the execution of the declarative part of the task's body). It may also be invoked by the new task in order to signal the completion of that task's activation.

TASK TERMINATION

_____Kbytes

Implements the set of rules for the completion, termination, and abortion of tasks.

I/O MANAGEMENT

_____Kbytes

Consists of all those portions of the runtime environment that are provided for the support of input and output. This includes in particular all those functions that support predefined packages from Chapter 14 of the Ada Reference Manual.

COMMONLY CALLED CODE SEQUENCES

_____Kbytes

A "catchall" category. It includes runtime routines in the classical sense: commonly called sequences of code. Typical examples are operation for multi-word arithmetic, block moves and string operations. Ada attribute calculations also fall into this category.

HOUSEKEEPING FUNCTIONS

_____Kbytes

Associated with the start up and termination of the execution environment of an Ada program. Such actions include determination of the particular hardware and software execution environment, setting of variables identifying same, processor and interrupt initializations, and so on. Similarly, if a program terminates, control is typically returned to some surrounding software whose state must be reset upon program exit.

IS ANY COMPONENT MISSING?

_____Kbytes

Use this space to indicate what you feel is not covered by the above components. (Explain below.)

Guideline to Select, Configure, and Use an Ada Runtime Environment

ADDITIONAL QUESTIONS CONCERNING RUNTIME ENVIRONMENTS

1. What is the granularity of the linker in selecting objects to load? (Check only one).

- ☐ ALL objects are always loaded.
- ☐ Any part of a library unit being required loads the entire unit.
- ☐ Individual subprograms may be extracted from packages only.
- ☐ Data objects that are referenced are allocated memory.

2. Is any user customization of the runtime possible? (Check all that apply).

- ☐ yes, by pragmas
- ☐ yes, by compiler switches
- ☐ yes, by modifying/replacing the source to selective runtime routines provided by the compiler vendor with the purchase of the compiler (i.e. device drivers, etc.).
- ☐ yes, by modifying the source to the entire runtime (after purchasing it)
- ☐ no

3. What documentation is provided to help the user configure the runtime? (Title or titles of manuals).

4. Does the compiler vendor provide services to customize the runtime for a particular application?

☐ yes, ☐ no

If yes, what charges are associated with these services? (Explain as necessary).

5. What is the price for the source code for the runtime environment?

\$_____, or _____not for sale

Guideline to Select, Configure, and Use an Ada Runtime Environment

6. Can we contact you for any follow up questions or clarification of discrepancies?

___yes, ___no

If yes, Name:_____

Phone Number:_____

7. State any important questions which you feel should have been included in this survey.

Guideline to Select, Configure, and Use an Ada Runtime Environment

SURVEY OF RUNTIME ENVIRONMENTS (V2.0)

LabTek Corp. is collecting information on Ada compilation systems to assist users in selecting, configuring, and using Ada runtime environments. This work is sponsored by the government, and the resulting catalogue is expected to be available to the public at cost of duplication.

Thank you for responding to our request for information, which we were having some difficulty obtaining from the compiler vendors. Your reply to the following questions will be greatly appreciated. If you do not have information on a particular question, please skip it and go on to the next one. If you have more than one target, we are primarily interested in "bare" targets at this time.

Please return this form to:

LabTek Corp.
8 Lunar Drive
Woodbridge, CT 06525
Attn: Tom Griest

Thank you for your time and input.

Please indicate the "bare" machine compiler this survey pertains to, below:

HOST	TARGET	COMPILER VERSION	DATE
_____	_____	_____	_____
Your Name: _____	_____	Phone Number: _____	_____

Guideline to Select, Configure, and Use an Ada Runtime Environment

DEGREE OF CONFIGURABILITY OF THE RUNTIME ENVIRONMENT

1. What is the granularity of the linker in selecting objects to load? (Check only one).

- ☐ ALL objects are always loaded.
- ☐ Any part of a library unit being required loads the entire unit.
- ☐ Individual subprograms and/or data objects may be extracted from packages only.

2. Is any user customization of the runtime possible? (Check all that apply).

- ☐ by pragmas
- ☐ by compiler switches
- ☐ by linker switches
- ☐ by modifying/replacing the source to selective runtime routines provided by the compiler vendor with the purchase of the compiler (i.e. device drivers, etc.).
- ☐ by modifying the source to the entire runtime (after purchasing it)
- ☐ not configurable

3. What documentation is provided to help the user configure the runtime? (Title or titles of manuals).

4. Does the compiler vendor provide services to customize the runtime for a particular application?

☐ yes, ☐ no

If yes, what charges are associated with these services? (Explain as necessary).

5. What is the price for the source code for the runtime environment?

\$_____, or _____ not for sale

Guideline to Select, Configure, and Use an Ada Runtime Environment

6. The following questions are considered to be the "TOP 10" questions that must be asked before deciding to use a runtime for real-time software.

6.1 a.) What is the resolution of the clock used for delay statements?

6.1 b.) How long, and for what reasons are interrupts disabled?

6.2. What rendezvous optimizations are performed? For example, when can the called task operate in the same context as the calling task?

6.3. What are the restrictions for representation clauses? (Attach separate sheet if necessary).

6.4. What scheduling algorithms are supported? For example, time slicing, dynamic priorities, run-until-blocked, etc.

6.5 What are the restrictions on pragma INLINE?

6.6 Is code "ROM"able?

Guideline to Select, Configure, and Use an Ada Runtime Environment

6.7 What is the specification for **package STANDARD**? (Attach separate sheet).

6.8 What is the specification for **package SYSTEM**? (Attach separate sheet).

6.9 Are machine code inserts supported?

6.10 What object types are supported by **pragma SHARED**?

7. Check the items below that are configurable for your runtime system. Fill in additional items not found on the list in the spaces provided.

☐ Maximum No. of Tasks	☐ Dynamic Task Priority
☐ Task Time Slice Default	☐ Semaphore Operations
☐ Timer Resolution	☐ Exception Trace
☐ Default Stack Sizes	☐ Fast Interrupt Entry
☐ Default Task Priority	☐ Terminal I/O
☐ Optional Numeric Co-processor	☐ Runtime System Variations (i.e. priority inheritance)

Guideline to Select, Configure, and Use an Ada Runtime Environment

_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

8. For each of the following components of the runtime, indicate the storage requirements (both code and data). If the runtime does not support one of the features below, enter 0 for the size. (See supplement for definitions, if needed).

DYNAMIC MEMORY MANAGEMENT	_____Kbytes
PROCESSOR MANAGEMENT	_____Kbytes
INTERRUPT MANAGEMENT	_____Kbytes
TIME MANAGEMENT	_____Kbytes
EXCEPTION MANAGEMENT	_____Kbytes
RENDEZVOUS MANAGEMENT	_____Kbytes
TASK ACTIVATION	_____Kbytes
TASK TERMINATION	_____Kbytes
I/O MANAGEMENT	_____Kbytes
COMMONLY CALLED CODE SEQUENCES	_____Kbytes
HOUSEKEEPING FUNCTIONS	_____Kbytes
IS ANY COMPONENT MISSING? (please list)	_____Kbytes

9. The name of a good technical contact at the compiler vendor?

Name: _____
Phone: _____

Thank You!

Guideline to Select, Configure, and Use an Ada Runtime Environment

SUPPLEMENT TO SURVEY V2.0

DESCRIPTION OF RUNTIME ENVIRONMENT COMPONENTS

AS DETERMINED BY THE ARTEWG

DYNAMIC MEMORY MANAGEMENT - Responsible for allocation and deallocation of storage at runtime. Also detects when a request for storage cannot be fulfilled, and for raising the exception `STORAGE_ERROR` as appropriate.

PROCESSOR MANAGEMENT - Implements the assignment of physical processors to tasks that are "logically executing". The processor management function is invoked by other components of the runtime environment, in order to block and unblock tasks. It keeps a list of those tasks which are "logically executing" and uses this list, in conjunction with the priorities of tasks, to determine which task or tasks should be assigned to processors.

INTERRUPT MANAGEMENT - Responsible for initialization of the interrupt mechanism of the underlying computing resource, and it is also responsible for resetting that mechanism after an interrupt has occurred, if the architecture of the underlying computing resource requires such resetting.

TIME MANAGEMENT - Consists of all those portions of the runtime environment that will support the predefined package `CALENDAR` and the implementation of delay statements. If the underlying computing resource offers enough functionality, the support of package `CALENDAR` is trivial.

EXCEPTION MANAGEMENT - Function implements Ada semantics for exceptions: that is, it determines whether there is a matching handler for the exception at hand, and if there is one, it transfers control to the handler. If there is no matching handler, it invokes the Task Termination function to terminate the task at hand or the main program.

RENDEZVOUS MANAGEMENT - Implements the semantics of the Ada rendezvous model. In order to do so, it utilizes variables that are internal to the runtime environments. These variables reflect, among other things, which tasks are blocked because they are waiting to rendezvous with other tasks, and what the exact circumstances of these wait states are. The rendezvous management function cooperates with the interrupt management function in the implementation of interrupt rendezvous, if the interrupt rendezvous is supported by the runtime environment.

TASK ACTIVATION - At some point after the task object has been created, the execution of the new task has to be started. This is effected by the task activation function. This function is invoked by the creator of a new task in order to start the new task's activation (which is defined as the execution of the declarative part of the task's body). It may also be invoked by the new task in order to signal the completion of that task's activation.

TASK TERMINATION - Implements the set of rules for the completion, termination, and abortion of tasks.

Guideline to Select, Configure, and Use an Ada Runtime Environment

I/O MANAGEMENT - Consists of all those portions of the runtime environment that are provided for the support of input and output. This includes in particular all those functions that support predefined packages from Chapter 14 of the Ada Reference Manual.

COMMONLY CALLED CODE SEQUENCES - A "catchall" category. It includes runtime routines in the classical sense: commonly called sequences of code. Typical examples are operation for multi-word arithmetic, block moves and string operations. Ada attribute calculations also fall into this category.

HOUSEKEEPING FUNCTIONS - Associated with the start up and termination of the execution environment of an Ada program. Such actions include determination of the particular hardware and software execution environment, setting of variables identifying same, processor and interrupt initializations, and so on. Similarly, if a program terminates, control is typically returned to some surrounding software whose state must be reset upon program exit.

IS ANY COMPONENT MISSING? - Use this space to indicate what you feel is not covered by the above components. (Explain below.)