TECHNICAL REPORT RD-GC-93-35

A REUSABLE ADA REAL-TIME MULTIPROCESSING EXECUTIVE FOR MISSILE SYSTEMS

Wanda M. Hughes
Guidance and Control Directorate
Research, Development, and Engineering Center

and

On-line Applications Research, Inc.
2227 Drake Avenue, Suite 10-F
Huntsville, Alabama 35805

September 1993

U.S. ARMY MISSILE COMMAND

Redstone Arsenal, Alabama 35898-5000

Approved for public release; Distribution is unlimited.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188 Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TR-RD-GC-93-35	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Guidance & Control Directorate RD&E Center; 6b. OFFICE SYMBOL (If applicable) AMSMI-RD-GC-S	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Commander, U.S. Army Missile Command ATTN: AMSMI-RD-GC-S Bldg. 4381 (Wanda Hughes) Redstone Arsenal, AL 35898-5254	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
A Reusable Ada Real-Time Multiprocessing Executive for Missile Systems

12. PERSONAL AUTHOR(S)
Wanda M. Hughes

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 6/89 TO 1/93	September 1993	18

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	RTEMS, C Language, Intertask Communication, Dynamic Memory Allocation, Semaphore, Executive, Multiprocessing, Ada, Reuse

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Real–Time Executive for Military Systems (RTEMS), is a state–of–the–art, object–oriented, real–time executive which provides a high performance environment for distributed embedded applications. This environment supports any mixture of homogeneous and heterogeneous, tightly–and–loosely–coupled processors. The RTEMS environment is based upon proposed standards for a real–time distributed tasking model. Although Ada was designed to provide a common language for large scale and real–time systems, it is generally acknowledged that Ada does not provide the complete set of capabilities required by many real–time military applications. The design philosophy driving the development of RTEMS is to provide a reusable component which reduces the size and complexity of software development efforts. This in turn promotes the development of cost effective, easily maintainable real–time weapon systems.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Wanda M. Hughes	22b. TELEPHONE (Include Area Code) (205) 876-4484	22c. OFFICE SYMBOL AMSMI-RD-GC-S

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

TABLE OF CONTENTS

I. INTRODUCTION

The weapon systems of today are the most complex in history. They typically contain a large, complicated real–time embedded software system integrated into a sophisticated hardware platform. As a result, today's software technology is pushed to its current limits. Although Ada was developed to provide efficient solutions to these difficult problems, several shortcomings have become evident since its adoption. The most notable of these concerns the Ada real–time executive contained in the runtime support package provided with existing Ada compilers.

Given the complexity of today's distributed embedded real–time applications, it is necessary to provide application developers with all the features of a modern commercial off the shelf real–time embedded executive. This executive must provide a solid reusable foundation upon which high–performance military and commercial distributed embedded systems may be developed.

In order to understand the problems associated with providing this real–time executive, one must first understand the domain of real–time embedded applications. In addition, one must understand the problems which must be addressed during development of real–time embedded applications and current solution approaches.

II. REAL–TIME EMBEDDED SYSTEMS

An embedded system is any computer system which is built into a larger system consisting of multiple technologies such as digital and analog electronics, mechanical devices, and sensors. Real–time embedded systems have a complex set of characteristics which distinguish them from other software problems. Real–time embedded systems are driven by and must respond to real world events while adhering to rigorous requirements imposed by the environment with which they interact. The correctness of the system depends not only on the results of computations, but also on the time at which the results are produced. The most important and complex characteristic of real–time application systems is that they must receive and respond to a set of external stimuli with rigid and critical time constraints.

Real–time systems are classified as hard or soft real–time applications. The distinction between a hard and a soft real–time system is based upon the correctness of data relative to its timeliness. In a hard real–time system, "correct" data received or processed at the incorrect time is of no use or, even worse, results in undesired, possibly tragic, consequences. A soft real–time application has less stringent timing requirements; data is still viable even if it is does not arrive at the opportune moment. A single real–time application can be composed of both soft and hard real–time components.

A typical example of a hard real–time system is a nuclear reactor control system which must not only detect failures, but must also respond quickly enough to prevent a meltdown. This application also has soft real–time requirements because it may involve a man–machine interface. Providing an interactive input to the control system is not as critical as setting off an alarm to indicate a failure condition. However, the interactive system component must respond within an acceptable time limit to allow the operator to interact efficiently with the control system.

III. REAL–TIME EXECUTIVES

A crucial characteristic of real–time applications is the importance of coordinating and managing a large number of concurrent activities. Unfortunately, most software is designed to execute with a single thread of control. The real–time executive provides a cornerstone on which to build a multitasking application. The application can be divided into a set of logical, autonomous tasks which execute independently, resulting in an asynchronous processing stream. Tasks may communicate and synchronize both with one another and the external environment using facilities provided by the real–time executive.

Complex real–time embedded applications can require more computational power than is available from a single microprocessor. These applications can benefit by distributing the computational activity over a number of microprocessors. A system with microprocessors all of the same type is referred to as homogeneous, while a system composed of mixed microprocessor types is referred to as heterogeneous. A multiprocessor real–time executive provides services which allow transparent access to global resources as well as communication and synchronization between tasks on different microprocessors.

The real–time executive also provides a foundation upon which a rich library of standard application components can be built. These components can be used repeatedly in other real–time projects. Thus, development time and cost are reduced.

IV. REAL–TIME APPLICATION AREAS

There are many areas, both military and commercial, which utilize real–time embedded systems. Major application areas include:

- Industrial Control
- Telecommunications
- Automobiles
- Aerospace
- Instrumentation
- Consumer Appliances
- Office Electronics
- Military

Although the environments vary widely among these categories, the embedded applications share the critical need for the services provided by a real–time executive. This common requirement makes technology transfer a desirable and effective possibility.

The success of current embedded applications combined with the rapidly increasing power of microprocessors has lead to a corresponding increase in application complexity, often demanding the use of multiple processors in a distributed application. This increase combined with the continuously expanding number of application areas, has lead to the need for a common set of capabilities to be provided by a real–time executive. A common capability set encourages the reuse of software and avoids the retraining of personnel from project to project regardless of the application area. In addition, real–time software reliability is maximized, development time is reduced, and software testing and debugging is simplified. As a result, both time and money are saved during software development.

V. STANDARDIZATION EFFORTS

The importance of defining this common functionality in real–time executives has lead to a number of standardization efforts. These efforts can be divided into two categories: executive standards and embedded real–time language standards.

VI. EXECUTIVE STANDARDIZATION EFFORTS

Executive standardization efforts have lead to a variety of interface standards including the Microprocessor Operating Systems Interfaces (MOSI), the Real–Time Executive Interface Definition (RTEID), and the Open real–time Kernel Interface Definition (ORKID). Unfortunately, because these specifications do not attempt to fully define the runtime behavioral characteristics of a compliant executive, one may have "portable" application software which may or may not execute in the same manner across multiple compliant executives. For example, neither RTEID nor ORKID specify whether a numerically low priority indicates a logically low or high priority task. Thus, a standard which defines only the interface requirements but does not fully define runtime behavior is insufficient for real–time embedded applications.

VII. LANGUAGE STANDARDIZATION EFFORTS

Ada was developed to provide a common language for large scale and real–time systems. Ada is largely successful in meeting the needs of this diverse user group. However, the original and current Ada language definition (ANSI/MIL–STD–1815A–1983) does not adequately satisfy the needs of the real–time embedded systems community. One of the most commonly cited limitations is that it provides only the rendezvous for task communication and synchronization.

The Ada 9X committee has been formed to address problems in the original Ada standard and develop a second generation Ada standard. Many of the problems addressed by the Ada 9X committee reflect the development community's dissatisfaction with the real–time features of Ada. A number of these specifically reflect the desire for communication, synchronization, and tasking control features commonly found in real–time executives. Many of these requests conflict and no amount of effort will satisfy all users.

In response to the requests for more robust real–time executive features, Ada 9X adds support for resetting task priorities, critical regions, and asynchronous communication. Ada 9X support for systems with multiple processors, although improved over that of Ada 83, still leaves a great deal of implementation detail to the compiler vendor and the end user.

Unfortunately, until Ada 9X is approved, these new features will not be available to the Ada community. Even after Ada 9X is approved, it will be years before mature products with the new features will be available. Even then, Ada will only provide a subset of the features common to most real–time executives, and many of these features are optional. Moreover, no matter how well the characteristics of the real–time capabilities are specified, their performance will always be vendor dependent.

VIII. BENEFITS OF STANDARDIZATION

The current solution to providing Ada developers with real–time embedded executive functionality is for the Government to purchase commercially off the shelf executives. This presents problems because the Government does not own, nor has the right to modify, code contained within the purchased executive. V&V techniques in this situation are more difficult than if the complete source code were available. Responsibility for system failures due to faulty software is yet another area to be resolved under this environment. Although some executive vendors will provide source code at an outrageous price, further problems arise when one considers licensing and maintenance costs. Typically a license must be purchased for each testbed and fielded unit. In addition, each processor in a multiprocessor system has to be licensed to use the executive. The cost of licensing (royalties) combined with the added trouble of insuring compliance with the licensing agreement is unattractive.

By standardizing on a single executive which is completely government owned, the government obtains a maintainable, high–performance executive with uniform behavior across multiple environments. With an executive based on existing standards, there will already be a body of commercial and military application developers familiar with the capabilities of such an executive. This increases the reusability of software, reduces personnel training, and ultimately leads to significant cost reductions.

IX. ADA'S REAL–TIME EXECUTIVE

Ada provides only the rendezvous construct for task communication and synchronization and does not address the problems associated with distributed real–time applications. It does not provide such standard constructs as semaphores, message queues, events, or asynchronous signals. All other forms of intertask communication and synchronization must be fabricated at the task level by the user. This results in a high level of overhead for operations which could have been provided much more efficiently by a real–time executive. For example, to implement a binary semaphore using the Ada rendezvous requires two context switches even when the semaphore is available. Acquiring an available semaphore is performed by most executives in a fraction of the time it takes to perform a single context switch. As a result, a large number of military embedded applications either do not allow or severely restrict the use of the Ada tasking facilities.

X. RTEMS

To address this problem, the U.S. Army Missile Command (MICOM) has developed RTEMS, a state–of–the–art, object–oriented, real–time executive which provides a high performance environment for distributed multiprocessor embedded applications. RTEMS provides a full featured, commercially compatible real–time distributed tasking model. RTEMS includes the following features:

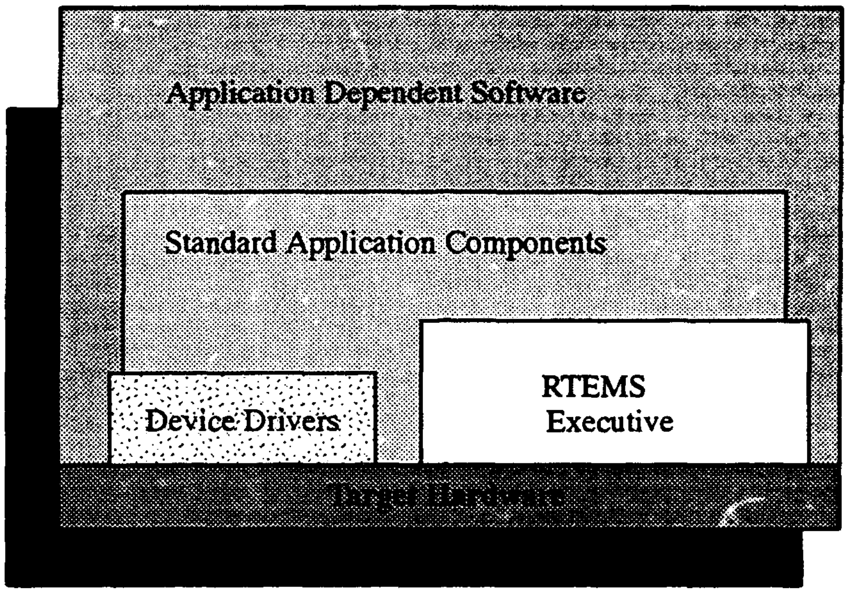
Application Dependent Software
Standard Application Components
Device Drivers
RTEMS Executive
Target Hardware

Figure 1. RTEMS Application Architecture

- reusable software component based on proposed standards
- multitasking capabilities
- homogeneous and heterogeneous distributed multiprocessing supported on both tightly and loosely coupled platforms
- event–driven, priority–based, preemptive scheduling
- optional Rate Monotonic Scheduling
- numerous communication and synchronization mechanisms
- responsive interrupt management
- deterministic execution times
- dynamic fixed and variable memory allocation
- high level of user configurability

RTEMS is a mature product with over ten man–years of development. The portability of RTEMS is demonstrated by the current availability for the Intel i80x86 and Motorola MC680x0 CISC processor families, as well the Intel i80960 RISC processor family. RTEMS provides the capabilities required by software developers designing and implementing time–critical military applications.

XI. RTEMS APPLICATION ARCHITECTURE

One important design goal of RTEMS was to provide a bridge between two critical layers of typical embedded real–time systems. Figure 1 illustrates how RTEMS serves as a buffer between the project dependent application code and the target hardware. Most hardware dependencies for real–time embedded applications can be localized to the low level device drivers. RTEMS provides efficient tools for incorporating these hardware dependencies into the system while simultaneously providing a general mechanism to the distributed application code that accesses them. A well designed real–time embedded system project can benefit from this architecture by building a rich library of standard application components which can be used repeatedly in other real–time embedded projects.

XII. RTEMS INTERNAL ARCHITECTURE

As illustrated in Figure 2, RTEMS can be viewed as a set of components that work in harmony to provide a set of services to a real–time embedded application system. The executive interface presented to the application is formed by grouping services into logical sets called **managers**.

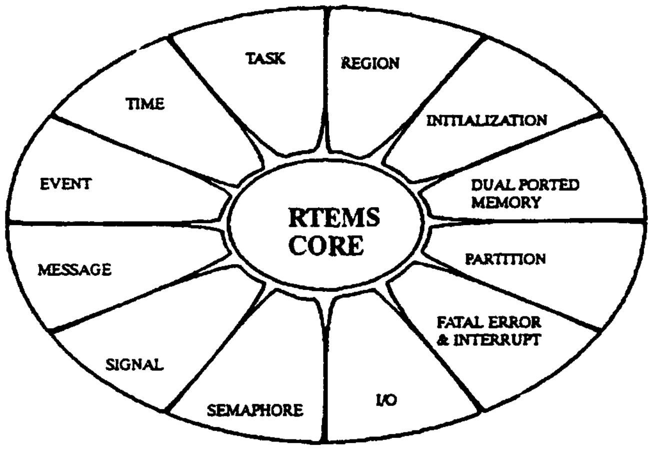

Figure 2. RTEMS Internal Architecture

Functions utilized by multiple managers such as scheduling, dispatching, and object management are provided by **handlers** in the **executive core.** Together these components provide a powerful runtime environment that promotes the development of efficient real–time embedded application systems.

XIII. COMMUNICATION AND SYNCHRONIZATION

In real–time multitasking applications, the ability for cooperating execution threads to communicate and synchronize with each other is imperative. Distributed embedded applications place the additional requirement to perform this type of processing across physical processor boundaries. A distributed real–time executive should provide an application with the following capabilities:

- Data transfer between cooperating tasks
- Data transfer between tasks and ISRs
- Synchronization of cooperating tasks
- Synchronization of tasks and ISRs

Most RTEMS managers can be used to provide some form of communication and/or synchronization, even across processors. However, managers dedicated specifically to communication and synchronization provide well established mechanisms which directly map to the application's varying needs. This level of flexibility allows the application designer to match the features of a particular manager with the complexity of communication and synchronization required. The following managers were specifically designed for communication and synchronization:

- Semaphore
- Message
- Event
- Signal

The semaphore manager supports mutual exclusion involving the synchronization of access to one or more shared user resources. The message manager supports both communication and synchronization, while the event manager primarily provides a high performance synchronization mechanism. The signal manager supports only asynchronous communication and is typically used for exception handling.

XIV. TIME

The development of responsive real–time applications requires an understanding of how RTEMS maintains and supports time–related operations. The basic unit of time in RTEMS is known as a **tick**. The frequency of clock ticks is completely application dependent and determines the granularity and accuracy of all interval and calendar time operations. The clock tick is typically provided by some external mechanism such as a real–time clock or counter/timer device.

By tracking time in units of ticks, RTEMS is capable of supporting interval timing functions such as task delays, timeouts, timeslicing, and the delayed posting of events. An interval is defined as a number of ticks relative to the current time. For example, when a task delays for an interval of ten ticks, it is implied that the task will not execute until ten clock ticks have occurred.

Interval timing alone is not sufficient for the many applications which require that time be kept in wall time or true calendar form. Consequently, RTEMS maintains the current date and time. This allows selected time operations to be scheduled at an actual calendar date and time. For example, a task could request to delay until midnight on New Year's Eve before lowering the ball at Times Square.

In order to integrate seamlessly with the Ada environment, an RTEMS specific version of the Ada CALENDAR package must be provided. The CALENDAR package provides only a subset of the functionality in the RTEMS time manager. Thus, an RTEMS specific version of CALENDAR can be implemented using the RTEMS time manager.

XV. MEMORY MANAGEMENT

RTEMS memory management facilities can be grouped into two classes: dynamic memory allocation and address translation. Dynamic memory allocation is required by applications whose memory requirements vary through the application's course of execution. Address translation is needed by distributed applications which share memory with another CPU or an intelligent Input/Output processor. The following RTEMS managers provide services which manipulate memory:

- Region
- Dual Ported Memory
- Partition

RTEMS memory management features allow an application to create simple memory pools of fixed size buffers and/or more complex memory pools of variable size segments. The partition manager provides services to manage and maintain pools of fixed size entities such as resource control blocks. Alternatively, the region manager provides a more general purpose memory allocation scheme that supports variable size blocks of memory which are dynamically obtained and freed by the application. The dual-ported memory manager provides executive support for address translation between internal and external dual-ported RAM address space.

XVI. RTEMS LANGUAGE INTERFACES

RTEMS was designed to provide a multitasking, real-time envirc ıment for applications written in any mixture of high-level languages and assembly. The initial technical approach was to develop an interface for each language without modifying any underlying RTEMS source code. Interfaces were developed which provided the full functionality of RTEMS to applications written in Ada, C, and assembly language. The Ada interfaces to the current RTEMS product are dependent on compiler specific implementation characteristics. This dependence does not provide the high reusability found in other RTEMS components.

In order to provide a standard reusable component for the support of Ada-based distributed real-time military applications, an Ada implementation based on the design of the existing, mature RTEMS product is currently under development. This will be accomplished not with a line-by-line translation of C to Ada, but by a thorough implementation of the RTEMS design in Ada. This approach allows algorithmic enhancements utilizing alternative Ada constructs and paradigms which should result in an efficient, powerful, and portable executive for Ada applications. To enhance portability, use of pragmas and machine code constructs will be minimized.

XVII. CONCLUSION

An analysis of the software reuse paradigm strongly suggests a tremendous potential for gain by concentrating on two key attributes of software systems: size and complexity. Boehm accurately states, "One of the primary controllable factors we have for improving software productivity is the number of instructions we choose to develop, either by deferring the development of marginally useful features or by using already available software." Fortunately, this may be the key factor which impacts software development productivity in the future. Standish observes, "the cost of software is an exponential function of software size, halving the size of the software which must be built much more than halves the cost of building it."

Furthermore, today's software is the most complex software in history. The software systems of the future are unlikely to be less complex. In fact, many experts believe that software complexity is also increasing exponentially. Once again, by reducing the complexity of the software which must be developed, an exponential reduction in the cost of its development, and even more importantly, its maintenance, is achieved.

Therefore, generic software is not just a sound engineering goal, but rather an essential target of opportunity in the battle to bring the software crisis to a suitable end. By identifying key reusable software components which contain highly complex software functions, developers achieve the benefits from both a size and complexity viewpoint. With deference to the size and complexity attributes, RTEMS was designed from ground zero to assimilate them into the following design philosophy:

> **Build a reusable software component that provides the functionality necessary to reduce the amount of software in a real–time military application by a significant amount, and reduce the burden of real–time tactical software developers by abstracting, isolating, and uncoupling a highly complex, but key component of most classes of real–time systems.**

The developers of RTEMS are confident that these software reuse goals can be realized through the use of the current release of the RTEMS executive. Furthermore, the completion of the RTEMS Ada implementation will provide the military systems development community with a reusable cornerstone and foundation upon which to build cost effective, maintainable software for the real–time military systems of today as well as those of the 21st century. RTEMS is a mature reusable real–time executive with robust support for both single and multiprocessor embedded systems. The RTEMS multiprocessing design supports all combinations of tightly– and loosely–coupled processors in both homogeneous and heterogeneous processor configurations.

INITIAL DISTRIBUTION LIST

	Copies
U.S. Army Materiel System Analysis Activity ATTN: AMXSY–MP (Herbert Cohen) Aberdeen Proving Ground, MD 21005	1
IIT Research Institute ATTN: GACIAC 10 W. 35th Street Chicago, IL 60616	1
Naval Weapons Center Missile Software Technology Office Code 3901C, ATTN: Mr. Carl W. Hall China Lake, CA 93555–6001	1
On–Line Applications Research 2227 Drake Avenue SW, Suite 10–F Huntsville, AL 35805	3
Louis H. Coglianese Advanced Technology IBM Corporation Federal Systems Company Route 17C, Mail Drop 0210 Owego, NY 13827	1
Kenneth Gregson MIT Lincoln Laboratory 244 Wood Street Lesington, MA 02173	1
John R. James Washington Engineering Division Intermetrics, Inc. 7918 Jones Branch Drive Suite 710 McLean, VA 22012	1
Keng Low SSC Lab M.S. 4002 2550 Beckleymeade Avenue Dallas, Texas 75237	1
Jeff Stewart Software Engineering Institute RM 5505 Carnegie Mellon University Pittsburgh, PA 15213	1

INITIAL DISTRIBUTION LIST (CON'T)

	Copies
James M. Short ODDRE&E (R&AT) RM. 3D1089, The Pentagon Washington, D.C. 20301–3080	1
Connie Palmer McDonnel Douglas Computer & Software Technology Ctr MS: 3064285 St. Louis, MO 63133–0516	1
Chris Anderson Ada–9X Project Office Pl/VTET Kirkland AFB, NM 87117–6008	1
Virginia Caster ODDRE&E RM. 3E118, The Pentagon Washington, D.C. 20301–3080	1
Sholom Cohen Software Engineering Institute Carnegie–Mellon University Pittsburgh, PA 15213	1
CEA Incorporated Blue Hills Office Park 150 Royall Street Suite 260, ATTN: Mr. John Shockro Canton, MA 01021	1
VITA 10229 N. Scottsdale Rd. Suite B, ATTN: Mr. Ray Alderman Scottsdale, AZ 85253	1
Westinghouse Electric Corp. P.O. Box 746 – MS432 ATTN: Mr. Eli Solomon Baltimore, MD 21203	1
Dept. of Computer Science B–173 Florida State University ATTN: Dr. Ted Baker Tallahassee, FL 32306–4019	1

INITIAL DISTRIBUTION LIST (CON'D)

		Copies
DSD Laboratories 75 Union Avenue ATTN: Mr. Roger Whitehead Studbury, MA 01776		1
AMSMI-RD		1
AMSMI-RD-BA		1
AMSMI-RD-BA-AD,	Bruce Lewis	1
AMSMI-RD-BA-C3,	Bob Christian	1
AMSMI-RD-CS-R		15
AMSMI-RD-CS-T		1
AMSMI-RD-GC,	Dr. Paul Jacobs	1
AMSMI-RD-GC-S,	Gerald E. Scheiman	1
	Wanda M. Hughes	10
	Phillip R. Acuff	4
AMSMI-RD-SS		1
AMSMI-GC-IP,	Mr. Fred H. Bush	1
CSSD-CR-S,	Mr. Frank Poslajko	1
SFAE-AD-ATA-SE,	Mr. Julian Cothran	1
	Mr. John Carter	1
SFAE-FS-ML-TM,	Mr. Frank Gregory	1