

1986-12

# A software system implementation guide and system prototyping facility for the MCORTEX executive on the real time cluster.

Garrett, Douglas R.

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS

A SOFTWARE SYSTEM IMPLEMENTATION GUIDE
AND SYSTEM PROTOTYPING
FACILITY FOR THE MCORTEX EXECUTIVE ON
THE REAL TIME CLUSTER

by

Douglas R. Garrett

December 1986

Thesis Advisor Uno R. Kodres

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 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) | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): 52 | 7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School |
| 6c ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000 | 7b ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000 |
| 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 SOFTWARE SYSTEM IMPLEMENTATION GUIDE AND SYSTEM PROTOTYPING FACILITY FOR THE MCORTEX EXECUTIVE ON THE REAL TIME CLUSTER

12 PERSONAL AUTHOR(S): Garrett, Douglas R.

| 13a TYPE OF REPORT | 13b TIME COVERED | 14 DATE OF REPORT (Year, Month, Day) | 15 PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM _____ TO _____ | 1986 December | 80 |

16 SUPPLEMENTARY NOTATION

| 17 COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Concurrent microprocessing; Embedded, real-time software system modeling |
| | | | |
| | | | |

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

This thesis develops a prototyping facility to support accurate exploratory modeling of the temporal structure of real time, concurrent software systems on a parallel processor architecture. The hierarchical bus parallel processor architecture, called the Real Time Cluster Star (RTC*), is the hardware on which an executive operating system, the Extended Multi-COmputer Real Time EXecutive (E-MCORTEX), provides the capability for concurrent real time processing. The prototyping facility is a tool to aid the system designer to assess the tasking structure and the resulting temporal behavior of concurrent multiprocess systems. This facility allows an early modeling of a proposed real time system so that system's design flaws can be discovered and corrected before the system is fully developed.

| 20 DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21 ABSTRACT SECURITY CLASSIFICATION | |
|---|---|---|
| ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS | unclassified | |
| 22a NAME OF RESPONSIBLE INDIVIDUAL: Prof. Uno R. Kodres | 22b TELEPHONE (Include Area Code): (408) 646-2174 | 22c OFFICE SYMBOL: Code 52Kr |

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

Approved for public release; distribution is unlimited.

# A Software System Implementation Guide And System Prototyping Facility For the Mcortex Executive on the Real Time Cluster

by

Douglas R. Garrett
Lieutenant, United States Navy
B.S., The College of William and Mary in Virginia, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1986

## ABSTRACT

This thesis develops a prototyping facility to support accurate exploratory modeling of the temporal structure of real time, concurrent software systems on a parallel processor architecture. The hierarchical bus parallel processor architecture, called the Real Time Cluster Star (RTC*), is the hardware on which an executive operating system, the Extended Multi-COmputer Real Time EXecutive (E-MCORTEX), provides the capability for concurrent real time processing. The prototyping facility is a tool to aid the system designer to assess the tasking structure and the resulting temporal behavior of concurrent multiprocess systems. This facility allows an early modeling of a proposed real time system so that system's design flaws can be discovered and corrected before the system is fully developed.

## THESIS DISCLAIMER

Some terms used in this thesis are registered trademarks of commercial products. Rather than citing each occurrence of a trademark, all trademarks appearing in this thesis will be listed below following the firm holding the trademark:

1. INTEL Corporation, Santa Clara, California
   - INTEL
   - MULTIBUS
   - iSBC 86/12A
   - iSBC 86/30
   - 8086

2. Digital Research, Pacific Grove, California
   - CP/M-86
   - PL/I-86
   - LINK86
   - DDT86

3. XEROX Corporation, Stamford, Connecticut
   - Ethernet Local Area Network

4. InterLAN Corporation, Westford, Massachusetts
   - NI3010 Ethernet Communication Controller Board
   - NI3210 Ethernet Communication Controller Board

5. United States Department of Defense
   - ADA

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# I. INTRODUCTION

## A. PROJECT OVERVIEW

The specific goal of this thesis is to develop and implement a software facility for explicit use in application software system development on the Real Time Cluster Star under the Extended Multi-COmputer Real Time EXecutive (E-MCORTEX) at the Naval Postgraduate School, Monterey, California. The general purpose is to provide aid to the system designer and lead programmer tasked with organizing and executing the development of embedded, real time, concurrent multiprocess software systems. It is the intent that this facility support accurate, dynamic, prototyping of the software system's modular partitioning, process multiplexing and data transfer requirements, and to allow continued development into the final system implementation.

A facility allowing efficient, flexible exploration of these fundamental design features prior to the implementation of the detailed design specifications of each process will produce a clean, tested and better understood structure on which to build a detailed implementation. Although systems modeled under this development tool will reflect the temporal character of the current hardware, the consequences of the logical organization and multiplexing of processes targeted for other hardware suites can likewise be productively simulated and analyzed.

## B. DISCUSSION

The push for greater computational rates has given rise to a rapid advancement in parallel architectural designs. Compounding the complexities inherent in parallel capability with exponential growth in computational capacities and expanding application domains, the emergence of new software design strategies to keep pace has generally lagged far behind. Serious design issues center around the partitioning of complex problems into individual, identifiable tasks and the organizational multiplexing of these tasks on processor and data path resources. Resolution of issues must not only promote the attainment of real time application requirements but also accommodate system evolution and maintenance.

Current design methodology generally lacks abstraction and evaluation of the temporal structural features of concurrent software systems in the early design phases of system development. Early design decisions provide the system's skeletal structure

upon which the detailed implementation of the various cooperating tasks of the integrated system must be built, orchestrated and refined. Weaknesses in strategies to dynamically execute, reconfigure and compare variants of this foundational organization prior to detailed design implementation creates a grossly inefficient development environment generally incapable of supporting practical exploratory design efforts without extreme expense in time and dollars.

The lack of a formal approach plagues the development of multiprocessor software systems. Typically projects severely overrun delivery dates and costs due to integration tests revealing that real time requirements can not be met and design overhaul is necessary. Systems are commonly accepted only partially implemented or with hastily conceived, poorly engineered fixes to meet specifications. System integration testing is far too late in the development cycle to discover design discrepancies in the organizational foundation of the system. An avoidance of this common scenario must certainly be thwarted if future systems are to be employed in reasonably timely fashion and the ensuing maintenance costs are kept from absorbing all available development resources.

Early prototyping is targeted at one important goal, to extract and model the system's temporal organizational features. This necessitates a partitioning of the problem into logical task components and the identification of the data communications between these tasks. The unit tasks and their associated data interdependencies are the building blocks from which the system structure is assembled, analyzed, restructured and eventually finalized. It is important to realize that this modeling is done without any detailed implementation of the internal activities of the unit tasks beyond estimations of required execution time of each task and the identification of data dependencies.

## C. THE AEGIS MODELING GROUP

The research interests of the AEGIS Modeling Group today embraces a broad spectrum of topical areas within the Computer Science discipline. Initially formed in the late 1970s as a general body to investigate alternative implementations of the U.S. Navy's mid 1960 design vintage AEGIS COMBAT SYSTEM, the central focus was on the AN/SPY-1A phased array processing unit.

The founding objectives were twofold. First, a feasibility look at replacing an ageing mainframe system with a complex of more updated, efficient and commercially

available LSI and VLSI micro-components as an integrated modular modeling of the AEGIS system and secondly, to likewise update the application software complement of the system through modern Software Engineering principles in support of maintainability and extendability.

From these objectives three branches of interrelated research have evolved under the AEGIS Modeling Group. Research into hardware models and implementations was initiated by W.D. Dilmore [Ref. 1] and R.A. Gayler [Ref. 2]. This was shortly followed by studies of the modification of the embedded application software complex by R.S. Riche and C.E. Williams [Ref. 3] that initiated the modernization of the algorithms. Lastly, the most prolific area of research was begun by W.J. Wasson [Ref. 4: p. 11] whose efforts initiated an entire sequence of projects that have evolved the necessary systems level software in support of the AEGIS modeling effort and given rise to the Real Time Cluster Star architecture. This system software has evolved in scope beyond that of specific AEGIS modeling to support a broader, more general class of embedded, real time, concurrent multiprocessing applications.

## D. THESIS STRUCTURE

Chapter Two reviews the major hardware components and current configuration of the Real Time Cluster followed by an introductory overview of the organization and distribution of the system level software. Chapter Three presents the implementation specifics of application complexes as required by the system level software. The mechanics required in application software implementation are presented as a user oriented guide. Particular interest lies in the MCORTEX mechanisms for supporting user process synchronization and data sharing as the application's temporal and structural foundations, features required for accurate system prototyping. Chapter Four defines the features and implementation of a facility for system design exploration and prototyping of real time, concurrent application software systems. The concluding chapter offers general remarks in review of the application software development facility.

# II. RTC* SYSTEM ARCHITECTURE

## A. HARDWARE SUITE

### 1. Overview

The Real Time Cluster Star (RTC*) is a multiple microprocessor, hierarchical bus structured hardware suite resembling the Carnegie-Mellon Cm* architecture [Ref. 5]. RTC* is specifically configured for the development and implementation of real time, concurrent multiple sensor data gathering and integration systems. These applications are typical of those targeted by the AEGIS Modeling Group in which real time, integrated sensor data is utilized to effectively manage the employment of multiple system effectors.

Currently RTC* is composed of two clusters each containing up to four single board microprocessor computer systems, cluster level random access memory, secondary storage facilities and an ETHERNET intercluster interface. The major system hardware components are presented and the current hardware configuration is illustrated in Figure 2.1.

### 2. Cluster Configuration

Generally clusters are tightly coupled groups of microprocessors that in addition to maintaining independent, asynchronous control of some local resources, cooperate in varying degrees in synchronous sharing of cluster level resources and intercluster data.

#### *a. Computing Nodes*

Unit level computing capability is provided by the INTEL iSBC 86/12A and iSBC 86/30 Single Board Computers each comprising a complete computer system built into a single printed circuit assembly. These single board systems are composed of an INTEL 8086 16-Bit Microprocessor, 64K bytes dynamic RAM extendable to 128K on the 86/30 systems, 16K bytes ROM again extendable to 64K on the 86/30, one serial communications port, three programmable parallel I/O ports and Multibus interface control logic. [Ref. 6]

The single board system includes an internal bus for board I/O operations and local memory accesses. This represents the first level in the RTC* bus hierarchy scheme. Additionally, each single board system in the cluster maintains a dedicated

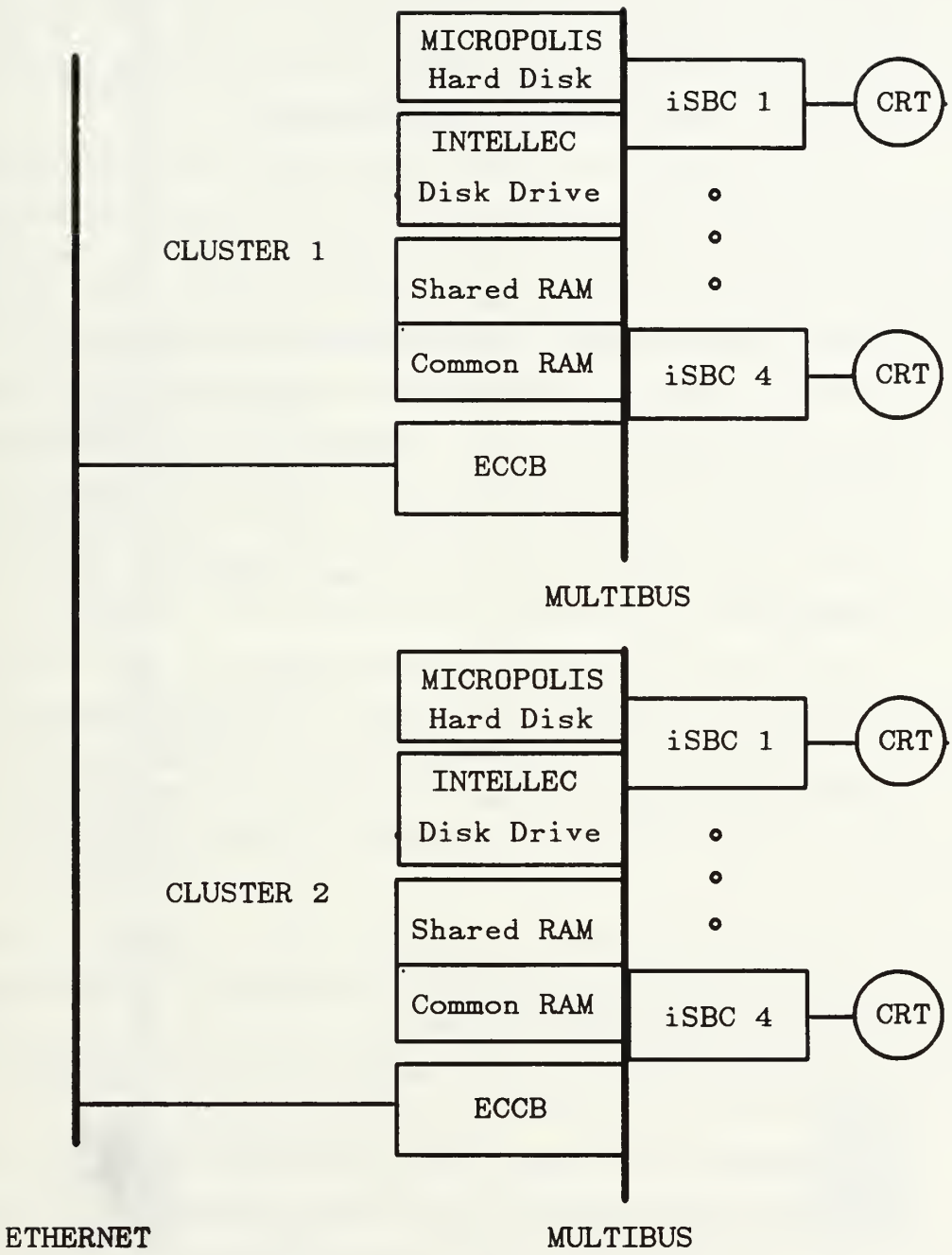



Figure 2.1 RTC* Hardware Configuration.

ADM-3A Dumb Terminal interactive I/O device [Ref. 7]. The terminal provides a dedicated user I/O link to each single board system in support of independent multiuser capability under the CP/M-86 Operating System and application loading and process output display when operating under the MCORTEX executive. The full duplex terminal interface is established through an RS-232C serial port resident onboard each single board system. I/O port addresses are given in Figure 2.2.

The 8086 microprocessor operates on a 5 MHz clock in the case of the iSBC 86/12A system and includes a 16 bit ALU, four 16 bit general purpose registers (addressable as eight 8 bit registers), two 16 bit pointer registers, two 16 bit index registers, four 16 bit segment registers and a nine bit flag register. The iSBC 86/30 board system can operate at 8 MHz. The 8086 instruction set supports 8 and 16 bit arithmetic operations, logical and string operations and multiple addressing and data transfer modes. The microprocessor provides a segmented one Mbyte logical address space by combining the left shifted four bits of each 16 bit segment register with the 16 bits of an offset register resulting in a 20 bit physical address. This allows a 64K byte address space via a pointer register for any given segment register value. [Ref. 6]

*b. Cluster Bus*

The INTEL MULTIBUS provides the backplane on which a variety of modular components are assembled to form a cluster and establishes the second level in the RTC* bus hierarchy. The parallel signal lines are partitioned into 20 address lines, 16 bidirectional data lines, 8 multilevel interrupt lines and a small collection of power, timing and supply lines [Ref. 8]. Practical data transfer rates are on the order of 12 Mbits/sec.

The MULTIBUS provides 12 slots for module attachment. The RTC* configuration establishes the six odd numbered slots as *masters*. A board occupying a master slot, upon its request, can be granted control of the bus for data transfer. Modules occupying the remaining six even numbered slots can not establish bus control and thus function in an access only, *slave* capacity within the cluster.

The internal bus of each single board system interfaces with the MULTIBUS for all external (off board) I/O and memory transactions. This requires that single board computers reside in master slots to enable access of cluster level memory. Typically access to local board RAM by other than the resident microprocessor is allowable via dual ported RAM control logic and the MULTIBUS interface, however, this capability is permanently disabled in the RTC* configuration.

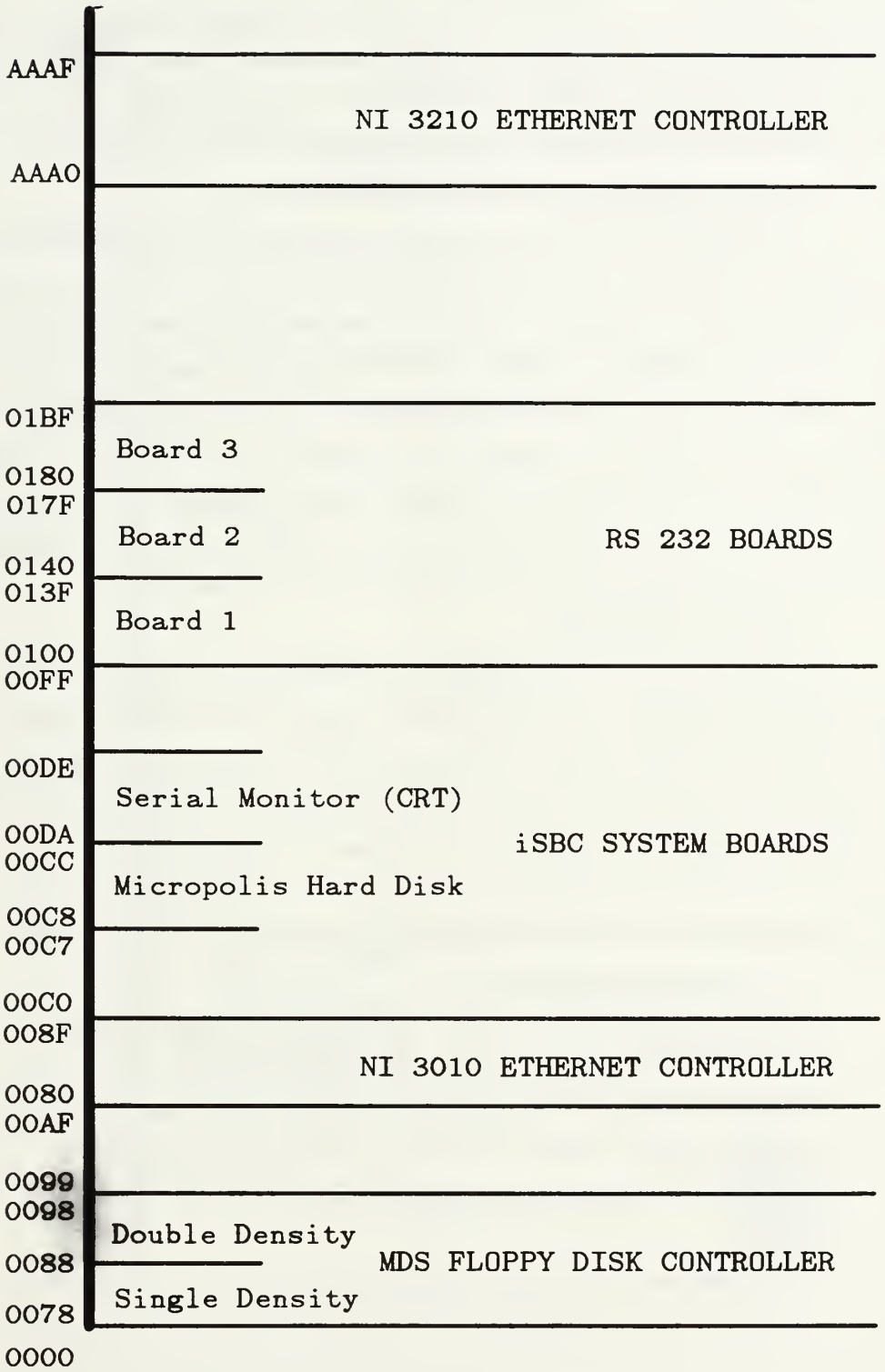



Figure 2.2 RTC* 8086 Microprocessor I/O Port Addresses.

This creates an environment in which RAM onboard each single board computer is exclusively private to the local CPU and the sharing of common data via the MULTIBUS link must be managed through cluster level memory outside the exclusive domain of any single board system in the cluster. The local onboard RAM lies in the lowest 64K of the physical address space (00000H-0FFFFH), the cluster RAM in the remaining address space above 0FFFF hexadecimal.

In typical MULTIBUS systems resolution of bus contention is based on the physical location of contending modules. In RTC* however resolution of MULTIBUS request contentions among modules occupying master slots is based on faster, binary tree arbiter logic located on a unique, special purpose board within each cluster. This board does not reside in a slot on the bus itself but has a hardwired interface to each of the odd numbered MULTIBUS slots. The logic is designed to ensure that no master module will disproportionally dominate bus control, however, operational observation has detected that under certain overloaded combinations of simultaneous bus requests, control is traded between two modules.

*c. Cluster Level Memory*

The cluster can be configured to provide 64, 96 or 128K of 8 bit word dynamic RAM resident on the MULTIBUS above the 0FFFFH physical address. Figure 2.3 illustrates the physical location of RAM within the logical address space. As the primary, cluster level, rapid access data resource, this RAM is located on boards requiring only non-master slots. Note however that the extended 64K of RAM onboard the iSBC 86/30 system can be allocated to the cluster level address space. Reguardless, cluster level memory is directly addressable via the MULTIBUS by all 8086 microprocessors within the cluster.

Secondary storage is available only at the cluster level. The INTELLEC MDS Diskette Operating System provides two eight inch floppy disk drives (A and B) and an intelligent Diskette Controller for channel management. The drives and associated power supplies reside enclosed as a peripheral to the cluster. Each drive provides approximately 2.05 Mbits of user data available at a transfer rate of .25 Mbits per second.

The Diskette Controller has two components on the MULTIBUS, the Channel Board and the Interface Board. The Channel Board is the primary control center of this disk system. It occupies a non-master MULTIBUS slot providing control of both Controller components via an 8 bit microprogrammed processor

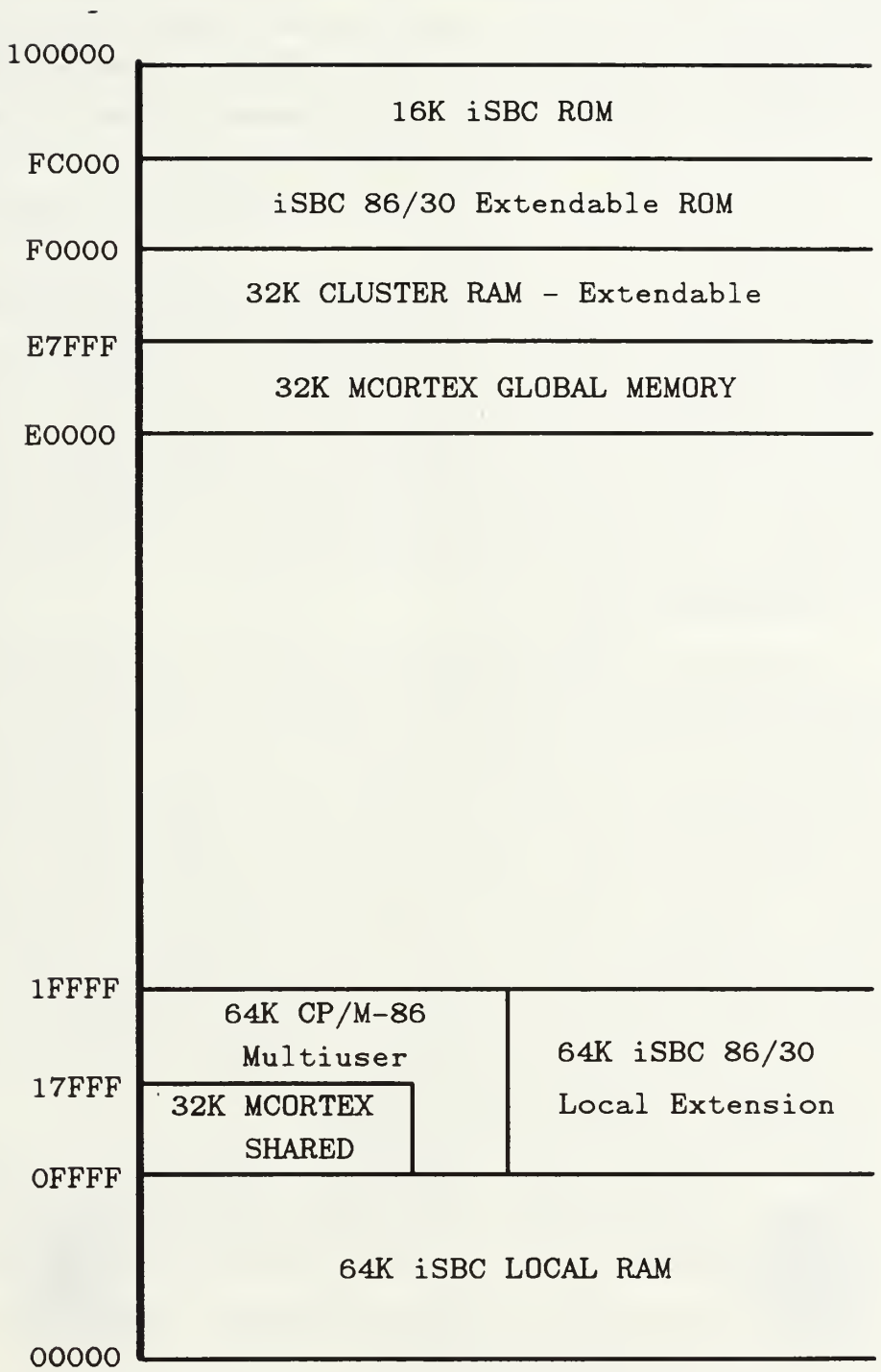



Figure 2.3 RTC* Physical Memory Distribution.

executing from 512 x 32 bits of onboard PROM. The Channel Board functions to coordinate the actions of the DOS with the I/O requests made by the microprocessors within the computing nodes of the cluster. The Interface Board requires a master slot on the MULTIBUS. It facilitates the transfer of system control signals and user data between the Diskette Controller, the drives and the MULTIBUS. As the working link, the Interface Board operates the disk hardware and maintains data integrity through an encoding/decoding CRC scheme. [Ref. 9]

The Micropolis model 1223-1 hard disk system provides additional on line storage at the cluster level. The hard disk system presents five logical disks (C to G). Each surface (disk) is partitioned into 580 tracks of 24 x 512 byte sectors or approximately 7 Mbytes per surface. Disk operation, data integrity and general housekeeping duties are performed by an onboard processor in the unit. Micropolis interface with the cluster is via an arbitrarily selected parallel I/O port on one of the single board systems in the cluster structure. [Ref. 10]

*d. ETHERNET Interface*

The interface between each cluster and the system unifying ETHERNET link is provided through an interLAN NI3010 ETHERNET Communication Controller Board (ECCB) [Ref. 11]. One per cluster, this board resides in a master slot on the MULTIBUS requiring not only access to cluster shared memory via the MULTIBUS but also a dedicated single board computer to integrate its function into the software structure of the cluster. ETHERNET data packets both transmitted from the cluster and received by the cluster are ported via the MULTIBUS between the ECCB and a specified area in cluster level memory (0800H:8078H-0C57H). This area is a staging site for the preparation of user declared application data shared between processes that execute on single board computers located in different clusters.

3. **ETHERNET - Intercluster System Assembly**

RTC* is a loosely coupled assemblage of tightly coupled single board computer clusters. The association of the clusters to form the system is actively managed and maintained at the cluster level but provided for by the passive character of the ETHERNET coaxial cable. A detailed specification of ETHERNET currently incorporated in RTC* is contained in [Ref. 12].

ETHERNET is the third and highest communication level in the hierarchical bus structure. This passive, serial communications link is commonly found at the base of many local area networks providing the bottom two layers in the International

Standards Organization's Open System Interconnection (ISO OSI) model, the Physical and Data Link layers. Communication between the MULTIBUS resident ETHERNET Communication Controller Board (ECCB) and the ETHERNET cable is accomplished over a transceiver cable running between the board and a dedicated transceiver and clamp on the coaxial cable itself.

## B. SYSTEM SOFTWARE

### 1. Historical Review

The system software configuration and distribution currently implemented on the RTC* hardware suite represents six years of iterative engineering, refinement and extension. Successive thesis projects under the Aegis Modeling Group have evolved the current implementation of E-MCORTEX as a real time executive for distributed concurrent execution of asynchronous but cooperative data sharing embedded application modules. Although E-MCORTEX operates under the design premises of the initial concept, enhancements and extensions have improved the facility for supporting application development and implementation.

The series of successive projects that has led to the current implementation of MCORTEX was initiated in 1980 by W.J. Wasson [Ref. 4: p. 11], who successfully structured its foundation from the Multics notion of a virtual processor and the eventcount model of process synchronization [Ref. 13: p. 116]. His description includes not only a cleanly partitioned software system but also a hardware configuration integrating low cost, reliable and available commercial components. Three following projects refined the design of the kernel and produced initial implementations. In early 1981 D.K. Rapantizikos succeeded in the initial implementation [Ref. 14] followed later that same year by E.R. Cox's [Ref. 15] further refinement. Mid 1982 S.G. Klinefelter [Ref. 16] published his work adapting the MCORTEX executive for dynamic interaction with the operating system utilities. MCORTEX began to take the form of the current implementation in 1984 when W.R. Rowe integrated the real time executive as a system software layer over the multiuser CP/M-86 operating system [Ref. 17: p. 10].

The current Extended-MCORTEX represents the most recent adaptive iteration, an implementation refined through two successive projects. First D.J. Brewer [Ref. 18: p. 11] extended MCORTEX to the cluster organization loosely integrating the clusters with ETHERNET local area networking. The final enhancement by R. Haeger

[Ref. 19: p. 11] has produced the present operational version of E-MCORTEX in which the executive not only synchronizes processes via eventcounts throughout the entire system across cluster boundaries but likewise associates user defined application data structures for transmission with the distributed eventcounts.

Presently two system modifications of noteworthy importance are underway. First is a further extension of E-MCORTEX to accommodate application software systems developed in ADA and secondly a reconfiguration of the ETHERNET software interface to update the system to faster interLAN NI3210 ETHERNET Communication Controller Board. The current system software distribution and memory maps are illustrated in Figures 2.4 and 2.5.

2. **Computing Node Software Organization**

The local RAM of each single board computer maintains a complete copy of the CP/M-86 Operating System and MCORTEX kernel. The MCORTEX executive is a component of the system software designed to create and actively manage an application environment for multiprocess synchronization and interprocess data sharing through system shared memory resources. MCORTEX provides each application process with all the available CP/M-86 Operating System functions.

Recall that instruction code and data resident in RAM at the single board system level is accessible only to the microprocessor resident on the board in the RTC* configuration. This distributed configuration of complete local copies of these system modules is one design feature supporting real time applications. Allowing each microprocessor a private copy alleviates MULTIBUS contention for access by the computing nodes to a cluster shared copy. The MULTIBUS is left open for MCORTEX synchronization activities and shared data transfers. In general, multiple copies of operating systems modules supports a more robust system reconfiguration capability eliminating the catastrophic single point failure potential of a single copy. Additionally each local board RAM provides a copy of the MCORTEX loader and DDT 86 in support of software development. Memory space occupied by the loader can be utilized in application runtime structures since the loader functions are complete once execution begins. For a detailed description of the loader operation see [Ref. 17: pp. 32-35].

Of the 64K of RAM resident on the single board system, approximately 28K is designated for application processes and their runtime structures under segment 0439H contiguous from offset 0000H to 6E6FH (04390H-0B1FFH physical). The

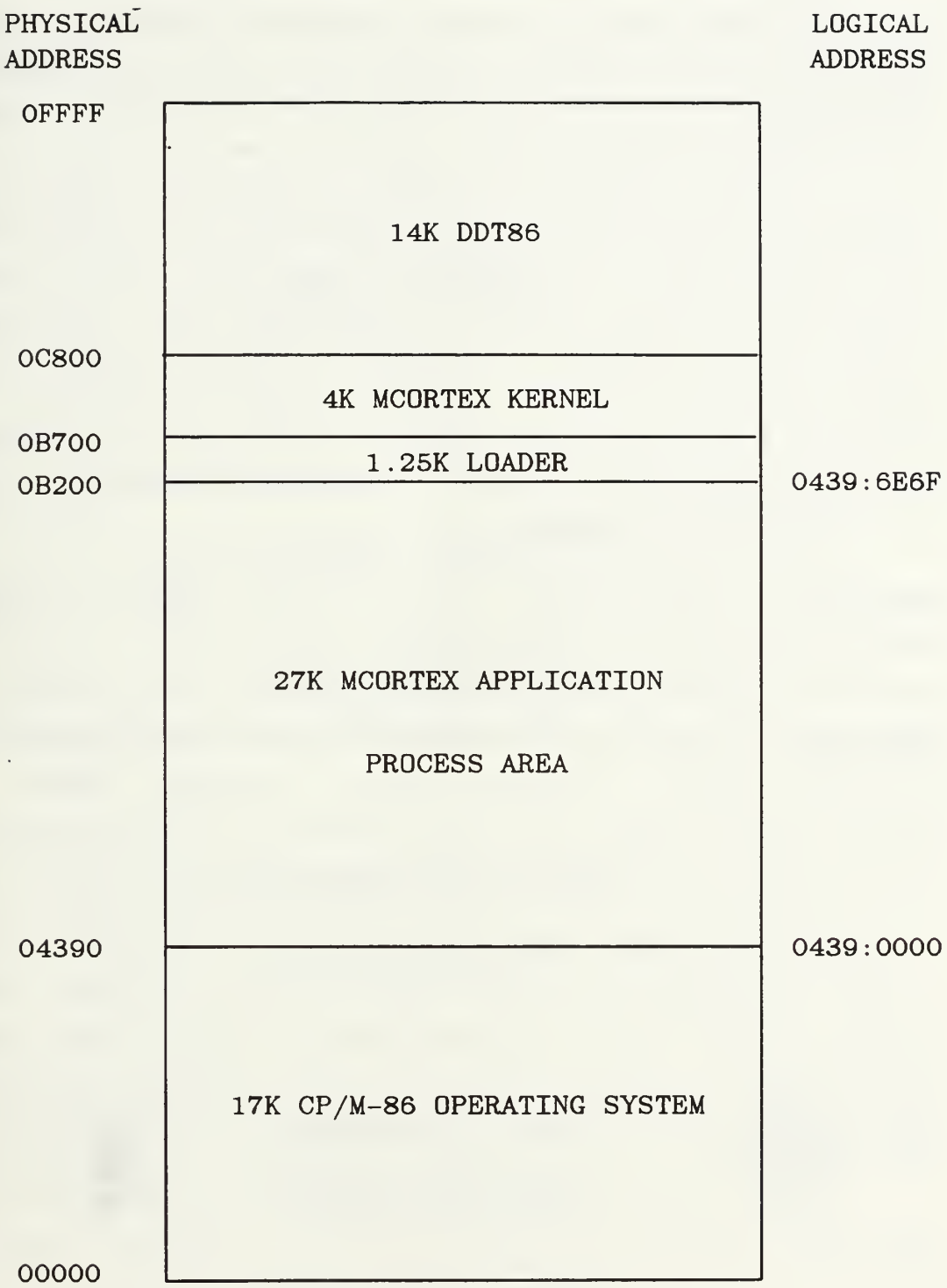



Figure 2.4 RTC* Local RAM Software Distribution.

utilization of this address space is left up to the application designer except in the case of one single board computer in each cluster that must be dedicated to the system Driver. The current implementation of the Driver modified by R. Haeger [Ref. 19: pp. 70-83] from previous implementations, although a system process, is created, scheduled and executed as a MCORTEX process. The Driver is the first application process loaded and executed functioning to establish the relationships between created eventcounts and associated application declared data structures and secondly to act as the ECCB device handler. The dedication of a single board system and resident microprocessor to intercluster ETHERNET communications is again a design decision in support of real time efficiency. If the driver were required to contend with co-resident, user created MCORTEX application processes for CPU cycles, the accumulative demand from other intracluster, user processes for ETHERNET communication services could become a seriously degrading system bottleneck.

3. **Cluster Software Organization**

From the available cluster RAM, MCORTEX designates a lower 32K addressable as segment 0800H contiguous between 8000H and FFFFH (10000H-17FFFH physical), as Shared Memory. Shared Memory is designated to accommodate all application declared shared data structures and serve specifically as a staging area for the transmission and reception of extracluster shared applications data and system distributed eventcounts over the ETHERNET channel. The implementation of the current operation and management of this activity represent the contribution made by R. Haeger under his modification to Brewer's Driver process.

An additional 32K of cluster RAM is designated as Common Memory which lies much higher in the logical address space in segment E000H contiguous from 0000H to 7FFFH (E0000H-E7FFFH physical). Under MCORTEX, Common Memory provides a MCORTEX Global Data area accessible only to the single board resident MCORTEX kernels. This Global Data area is used to maintain data structures necessary for the proper scheduling and synchronization of MCORTEX created application processes.

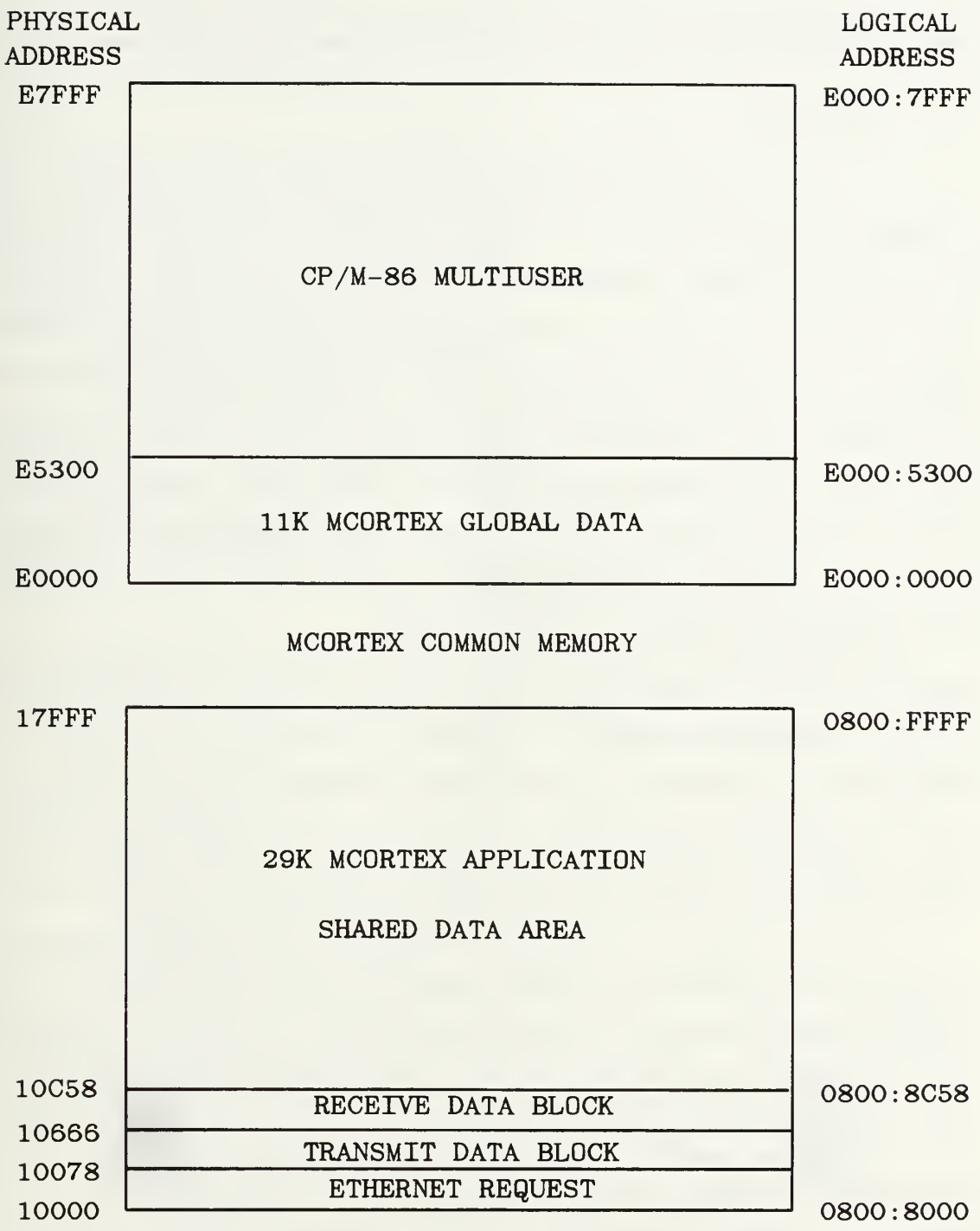



Figure 2.5 RTC* Cluster RAM Software Distribution.

# III. IMPLEMENTING APPLICATION SYSTEMS UNDER E - MCORTEX

## A. OVERVIEW

E-MCORTEX has evolved under AEGIS Modeling Group research as an executive system for organizing the structure and managing the execution of real time, embedded, concurrent multiprocess software systems. The facilities of this executive for supporting this class of applications have evolved directly from characteristics of the applications themselves. A system's response in meeting real time requirements is greatly enhanced through dedication of the hardware resources to single applications. System and application process code and data structures are statically assigned to shared RAM resources distributed across the system to effect concurrent execution, rapid access and minimal context switching. This static, embedded feature is detrimental only in that it disallows dynamic process redistribution or degraded operation in response to hardware failures. A single critical hardware component failure can prove catastrophic to the entire system.

The current RTC* architecture allots 28K of the 64K RAM available on each single board system to accommodate application process code and runtime structures (segment 0439H:0000H-6E6FH). Application processes are developed and implemented as parameterless PL/I-86 procedures constructed under the utilities of the CP/M-86 Operating System. Each user process is compiled independently and then linked with other task processes and system object modules to form a single, multiplexed executable module for static loading to a single board computer.

Multiplexed application processes share single board resources and the functions of runtime routines and CP/M-86 utilities, however, each user process in the executable module is uniquely identified and initialized from within the module as a MCORTEX process. Entry to these processes is made strictly through established MCORTEX function calls. The specification of application processes as MCORTEX processes establishes each as a unique virtual processor. Information in the specification is used by MCORTEX to prepare the state of the actual processor for execution of application code of the designated virtual processor. Mapping virtual processors to a real processor from among multiplexed processes is enacted by the executive based on process state and relative priority. The highest priority ready process is assigned the real processor.

Upon loading an executable module, all of the created MCORTEX processes are made ready. From this point, achieving process synchronization through the actions of the process scheduling mechanism is determined by the application processes employment of the MCORTEX synchronization functions and the declared process priority.

## B. APPLICATION IMPLEMENTATION MECHANICS

This information is presented as a general purpose users guide to implementing application systems under the E-MCORTEX executive and as a basis for productive utilization of the prototyping facility.

### 1. Application Interface to MCORTEX

The System Definition module (*Sysdef.pli*) produced by D.J. Brewer [Ref. 18: p. 103] creates the eleven available MCORTEX interface procedures. Its general utility is to define the executive's interface and provide a gateway that resolves the parameter anomalies between the PL/I-86 application modules and the PL/M-86 MCORTEX kernel. The Sysdef file holds the procedure declarations which are made visible to each application module through the PL/I-86 **%INCLUDE** construct [Ref. 20: p. 25]. The actual executable code for each function declaration resides in the Gatemod / Gatetrc assembly code module (*Gatem/t.a86*) [Ref. 18: pp. 105-110]. Assembly of this module produces either the *Gatemod.obj* or the *Gatetrc.obj* module one of which must be linked with the application process modules in forming the final multiplexed executable module. Both object modules perform the execution of all MCORTEX functions but in addition Gatetrc includes a trace capability that displays the activities of the executive for system diagnostics. A copy of the Sysdef module is provided in Appendix A for local reference.

The eleven MCORTEX procedures can be partitioned into two subsets for exclusive use in either the *initialization* phase of application system establishment or application ***process synchronization*** during system execution. Table 1 provides the proper syntactic form for embedding the MCORTEX procedure calls into the PL/I-86 source code of the application task and initialization modules.

As an add feature the Sysdef module provides a **%REPLACE** block allowing the user to substitute necessary values for alphanumeric labels used throughout application process source code. The use of labels over more cryptic identifiers and pointers enhances readability supporting followup maintenance.

TABLE 1

MCORTEX FUNCTION SYNTAX

***** INITIALIZATION FUNCTIONS *****

call **CREATE_EVC** (' < eventcount id > 'b4)

call **CREATE_SEQ** (' < sequencer id > 'b4)

call **DEFINE_CLUSTER** (' < local cluster address > 'b4)

call **DISTRIBUTION_MAP** (' < distributed type > 'b4,' < id > 'b4,
' < ETHERNET cluster addressing > ,b4);

call **CREATE_PROC** (' < virtual proc.id > 'b4,' < priority > 'b4,
' < stack ptr.max. > 'b4,' < stack seg. > 'b4,
' < instruction ptr. > 'b4,' < code seg. > 'b4,
' < data seg. > 'b4.,' < extra seg. > 'b4);

***** SYNCHRONIZATION FUNCTIONS *****

call **ADVANCE** (' < eventcount id > 'b4)

call **AWAIT** (' < eventcount id > 'b4,' < threshold value > 'b4)

call **PREEMPT** (' < virtual processor id > 'b4)

< variable > = **TICKET** (' < sequencer id > 'b4)

< variable > = **READ** (' < eventcount id > 'b4)

< variable > = **ADD2BIT16** (' < variable > 'b4,' < variable > 'b4)

2. **Initialization of Software Components**

Initialization modules are unique, user generated PL/I-86 procedures developed and compiled as individual source files [Ref. 21: pp. 55-70]. One must be provided as the first executable process within *each* multiplexed application module by linking its object module with object modules from application process source files. The initialization is executed only once when the executable application module is loaded on a single board system. Two categories of modules are required, one to perform the *cluster* initialization and the other to provide *application node* initialization.

***a. Cluster Initialization***

A unique cluster initialization module is constructed for *each* hardware cluster to be utilized in the application. See Figure 3.1. This module is the absolute first application code to be executed in the cluster. Its first function is to establish the

cluster under the MCORTEX executive. This is followed by the creation of all eventcounts and sequencers used by MCORTEX processes throughout the entire cluster. Eventcounts that must be visible to processes that execute on processors *outside* the cluster must be specifically *distributed*, all others are generally referred to as *local*. After all internal initializations are executed an AWAIT call is made on the common initialization process reserve eventcount 'fe'. An ADVANCE call on this eventcount from within any MCORTEX process will return all processors to CP/M-86 control. [Ref. 18: p. 73]

The Driver is a PL/I-86 system process designed to provide a communications interface between clusters via the ETHERNET interface. Cluster intialization creates the Driver, a single MCORTEX process for execution on a dedicated single board computer. This is consistent with the function of the Driver as the ECCB controller requiring a dedicated cluster microprocessor. As a result, the Driver never competes for CPU cycles, therefore its priority and identity are permanently fixed. The remaining Driver initialization parameters are established as are those of any other application processes. The code segment is established as 0439H for location in the user area of local memory, and the data, stack and extra segments are fixed at segment 0800H so that data can be stored in local memory (offset less that 8000H) or in the defined Shared Memory area (offset greater or equal to 8000H). Establishing the two remaining parameter values for the maximum stack and instruction pointers will be discussed under process multiplexing.

*b. Application Node Initialization*

The unique initialization module for local board application processes requires fewer declarations than at the cluster level. See Figure 3.2. This module is strictly tailored to uniquely create its co-linked application process object modules as MCORTEX processes. Although eventcounts and sequencers *can* be created at this level, accepted convention reserves this activity for cluster initialization. Cluster identification and eventcount distribution are *strictly* cluster initialization activities.

Of the eight parameters required to establish a MCORTEX process, four are constants under the current E-MCORTEX implementation. As in the case of the Driver module, the code segment is permanently set as 0439H to properly address application instruction code loaded in the local RAM. Likewise the data, stack and extra segments are set as 0800H for Shared Memory addressing. Of the remaining four

```
<procedure name> : procedure options (main);
  %include 'sysdef.pli';
  %replace /* optional-local user labels */
  /* main */
  call DEFINE_CLUSTER ('<local cluster address>'b4);
  /* '0001'b4 or '0002'b4 in two cluster system */


  /* user area as required by application */
  call CREATE_EVC ('<eventcount 1 id>'b4);
                                .
  call CREATE_EVC ('<eventcount n id>'b4);

  call CREATE_SEQ ('<sequencer  1 id>'b4);
                                .
  call CREATE_SEQ ('<sequencer  n id>'b4);


  /* REQUIRED-system labels replaced in Sysdef.pli */
  call CREATE_EVC (ERB_READ);
  call CREATE_EVC (ERB_WRITE);
  call CREATE_SEQ (ERB_WRITE_REQUEST);


  /* distribute eventcounts to other clusters
     as required by the application */
  call DISTRIBUTION_MAP ('00'b4,'<id>'b4,'0003'b4);
  /* allows eventcount dist. only = '00'b4 */
  /* cluster ETHERNET addressing = '0003'b4 */
                                .
  call DISTRIBUTION_MAP ('00'b4,'<id>'b4,'0003'b4);


  /* create ECCB DRIVER as a MCORTEX process */
  call CREATE_PROC ('fc'b4, '80'b4,
                    '<stack ptr. max from map>'b4,
                    '0800'b4,'<ip from map>'b4,
                    '0439'b4,'0800'b4,'0800'b4);

  call AWAIT ('fe'b4,'01'b4);
end <procedure name> ;
```

Figure 3.1 Cluster Initialization Module Structure.

individual process parameters, the MCORTEX process (virtual processor) id must be uniquely established by the designer and the execution priority of the process set. Again the method of determination of the maximum stack and instruction pointer values is deferred until process multiplexing is introduced.

```
<procedure name> : procedure options (main);
  %include 'sysdef.pli';

  /* main */

  /* create local application process 1 */
  call CREATE_PROC('<virtual proc.id>'b4,'<priority>'b4,
                   '<stack ptr. max from map>'b4,
                   '0800'b4,'<ip from map>'b4,
                   '0439'b4,'0800'b4,'0800'b4);
                                        .
                                        .
                                        .
  /* create local application process n */
  call CREATE_PROC('<virtual proc.id>'b4,'<priority>'b4,
                   '<stack ptr. max from map>'b4,
                   '0800'b4,'<ip from map>'b4,
                   '0439'b4,'0800'b4,'0800'b4);

  call AWAIT ('fe'b4,'01'b4);
end <procedure name> ;
```

Figure 3.2 Application Initialization Module Structure.

3. **Establishing Process Synchronization**

*a. Discussion*

Chapter II of D.J. Brewer's thesis [Ref. 18: pp. 18-27] presents the fundamental notions of an event oriented synchronization model as the foundation of the MCORTEX executive. As opposed to models constructed on the principle of mutual exclusion and synchronization around shared data, the event oriented model coordinates the execution of unit events (processes) through an established signalling convention. The MCORTEX synchronization mechanism is based on observations of events in the course of asynchronous execution made through primitive system objects. Objects of type *eventcount* are created to keep a count of the number of events that have occured in a particular class. Objects of type *sequencer* establish a sequential

ordering of the execution of events in the system. Although designed for physically shared memory, this synchronization mechanism is equally suited for distributed systems which characteristically lack globally shared memory resources and exhibit unpredictable timing delays. [Ref. 13: p. 116]

*b. Mechanics*

Manipulation of eventcount and sequencer values is carried out at the unit process level through calls to appropriate MCORTEX procedures. These procedures are available to all application processes through the *Sysdef.pli* declaration file establishing the MCORTEX interface through which the synchronizational behavior of the system is driven.

(1) *Eventcount Primitives.* The *Advance*(*E*) primitive is used to signal completion of the next iterative execution of the event with which the eventcount *E* is associated. This procedure increments by one the integer value associated with *E* from the initial value of zero. The eventcount value can be used to count the number of completions of an application process associated with eventcount *E*. [Ref. 13: p. 116]

The *Await* and *Read* primitives both retrieve current values held by specified eventcounts. The *Read*(*E*) function returns the current value of eventcount *E*. The returned eventcount value is assigned to a variable that may be usefully employed as a conditional control value. The *Await*(*E*,*V*) procedure is used to suspend the application process until the eventcount *E* has reached the specified threshold value *V*. If the eventcount value has already reached or exceeds the threshold value, then the application process is not suspended, although the physical processor may be switched to a higher priority ready process. [Ref. 13: p. 116]

(2) *Sequencer Primitives.* The sequencer is used to control exclusive sequential access to a shared resource. The function *Ticket*(*S*) returns the current non-negative integer value of sequencer *S* and increments the value of *S* by one. A sequencer is used in an analogous fashion to a retail customer service queue. A customer requiring service takes the next available sequentially numbered ticket. The next available server increments a centrally visible counter (an eventcount). Waiting customers receive service when the counter becomes equal to their ticket number. Under MCORTEX each declared sequencer establishes an independent roll of tickets. The ticketing function dispenses sequentially numbered tickets for the specified sequencer and the await primitive causes processes to wait until the eventcount value equals the ticketed (threshold) value. [Ref. 18: p. 24]

4. **Establishing Process Data Sharing**

***a. Discussion***

The current E-MCORTEX interprocess data sharing mechanism is the product of the Haeger thesis [Ref. 19: p. 11] built on the intercluster ETHERNET communication interface established by Brewer. The mechanism generally provides dynamic runtime exercise of memory and data path resources but within strictly prescribed implementation guidelines under static, application designed structures.

Haeger identifies three classes of shared data as the communications interface between cooperating processes [Ref. 19: pp. 22-24]. Class delineation is founded upon the relative locations of the sharing processes. First are processes that share a common single board system. Secondly are those that reside on different boards within a common cluster and finally those that reside in different clusters.

As assumed by PL/I-86 routines, the stack, data and extra segment registers must have identical contents. In compliance with this requirement and as a design simplification of the sharing mechanism, the MCORTEX executive physically stores all three classes of shared data in cluster Shared Memory. By establishing the registers' contents as segment 0800H the remaining sixteen bit offset address can range between 8000H and FFFFH to address the cluster shared physical memory residing in the physical address space between 10000H and 17FFFH.

This may initially appear an over-simplification that creates unnecessary MULTIBUS activity in the case of common board resident processes and opens multiple options for implementing intercluster sharing. In actuality the MULTIBUS at a 10 Mbit/sec. transfer rate easily manages the data transfer requirements of processes implemented in high level languages. For cases of intercluster sharing, Haeger presents and contrasts three models [Ref. 19: pp. 18-21]. His final design implements the most efficient model which minimizes ETHERNET transmissions and maximizes RAM utilization through minimal data duplication. Specifically he employs a scheme that uniquely manages at the cluster level a duplication of the intercluster shared structures required only by local cluster resident processes.

***b. Mechanics***

Structuring data sharing in application systems can be reduced to a three phased specification which does not require detailed knowledge of the underlying system mechanisms on the part of the application system designer. The first phase requires an abstraction and global declaration of *all* interprocess shared data structures

required in the application. These are established as applications shared PL/I-86 structures via the *shared.dcl* file included in the PL/I-86 source files of *all* application modules in the system. This provides visibility of all global data structures at the level of each process and ensures that all processes throughout the system have consistent structures through which to share data. Each structure is declared as a pointer based circular queue indexed from zero to n-1 where n is the queue length. The required maximum lengths of the circular queues are based on estimates of data producing and data consuming process interactions.

The second phase of this data sharing specification requires the application system designer to statically specify for *each cluster* a subset consisting *only* of those globally visible structures required for use by processes that execute at the local cluster. This subset must include structures intended for *both intercluster and intracluster* data sharing. The designation of this subset is accomplished by establishing a unique, cluster specific *pointer.ass* file for each cluster. This file establishes the absolute offset addresses of the required data structures in the cluster Shared Memory (0800H:8C58H-FFFFH) by setting the pointers of the variable based queues through use of the PL/I-86 **UNSPEC** construct [Ref. 20: p. 72]. This file is likewise included in the source file of all processes *common to a particular cluster* allowing the alloted cluster RAM to be efficiently managed and tailored to the specific requirements of the cluster processes.

Detailed implementation of the functions of the unit processes comprising the application system are developed *independently* from the establishment of shared data structures. The results of phases one and two bring into the local environment of each process the pre-established global data structures. These structures are the communications interfaces between the individual process environments throughout the application system.

The third and final phase in establishing shared data resources resolves the issues of maintaining data consistency among the multiplicity of copies of intercluster shared structures and the segregation of intracluster from intercluster shared structures within each Shared Memory. The central problem focuses on delineating the subset of global data structures established in more than one cluster Shared Memory. This is critical for maintaining data consistency among all instantiations of each structure and thus effecting intercluster sharing. This is precisely the contribution made by Haeger.

Haeger's underlying assumption holds that every shared data item is related to some eventcount [Ref. 19: p. 39]. His extension of system wide data sharing is reached first through association of *each* shared data structure with the specific eventcount that when advanced signals not only an event completion but also a new iteration of data in the associated structure. Data structures that exist in only one cluster for the purposes of strict intracluster sharing are naturally associated with eventcounts created for intracluster synchronization. Those eventcounts specifically distributed to other clusters naturally associate with data structures whose contents are of consequence at the distributed cluster site and are locally identified at that cluster. To make new iterations of data available at each cluster where a structure is instantiated, the Driver process transmits not only the updated eventcount value but also the new data iteration which is queued appropriately. Whenever an eventcount is advanced, if it is distributed and has an associated shared data structure, the update data is likewise distributed over the same clusters. [Ref. 19: pp. 33-40]

The correct application specific execution of this distribution is established through two data files each read once by the Driver at the initialization of the cluster. First is the *address.dat* file that establishes the number of distributed clusters and specifics of the cluster's ETHERNET addressing. These files, one per cluster, remain static in a two cluster system. See Figure 3.3.

```
            *** Cluster 1 Address.dat File ***

1
'00000000'b,'00000001'b,
'00000000'b,'00000001'b

            *** Cluster 2 Address.dat File ***

1
'00000000'b,'00000010'b,
'00000000'b,'00000010'b
```

Figure 3.3 Cluster Address.dat Files.

The application system designer determines the association of data structures with eventcounts established in the second data file that must be created as part of the software system implementation, the *relation.dat* file. Cluster specific, this

formatted input data file uniquely establishes the logical relationship between the necessary eventcounts and *all* data structures specifically used by the processes that execute within the cluster. Eventcounts are **not** required to have associated data structures, however, the reverse is absolutely required. The relation.dat file allows each cluster's shared data memory to be independently configured. As a result data shared via multiple independent copies of common structures can reside at different offset addresses in the Shared Memories of different clusters. See Figure 3.4.

```
 <evc id><#data items><item point><queue len><bytes>
(hex)    (decimal)     (hex)     (decimal) (decimal)

column
  5          15          25          35          45
-------------------------------------------------------
  01         1           8c58        50          6
  2a         3           8d84        10          3
                         8da2        20          5
                         8e06        2           10
  33         2           8e1a        30          5
                         8eb0        50          9
  dd         1           9072        15          5
  00         0           0000        00          00
```

Figure 3.4 Relation.dat File Format.

The contents of this file is read once by the Driver process and maintained in Global Memory as the cluster's Relation Table. The file and table are isomorphic providing space for up to 100 eventcount IDs as allowed under MCORTEX, up to 10 data items (structures) as required per eventcount, absolute pointer address offset of the first byte in each queue, length of the queue in number of items and number of bytes per data item. [Ref. 19: pp. 41-42]. The Driver process as the ETHERNET controller manages the organization of data packets with the supplied specifications establishing and maintaining the data queues beginning each with slot zero. The system designer must only be aware of memory management within the Shared Memory space (0800H:8C58H-FFFFH) of each cluster and most critically must maintain consistency between the proper address offsets in the *relation.dat* and *pointer.ass* files common to each cluster.

The result from this shared data specification allows processes that synchronize on common eventcounts to likewise share data. It is absolutely

unnecessary for the application processes to know the physical location of other cooperating processes. However, if the system designer reconfigures the distribution of application processes, he must ensure that dependent processes remain reachable through appropriate eventcount distributions and that required data structures are provided in the appropriate cluster's Shared Memory. This requires a recompilation of the process source code inclusive of new data pointer assignment values, additions and deletions to the cluster's relation.dat file, changes to the initialization source files effecting process creation and eventcount distributions and a relinking of the initialization and process object modules to produce the new executable load module for each single board computer.

5. **Structuring Application Modules**

Each user designed, PL/I-86 implemented application task is established as a parameterless procedure residing in a unique file *<filename> .pli.* Process source files are independently compiled [Ref. 21: pp. 55-70] producing object modules which are selectively linked to form executable application modules of multiplexed processes. It is at the individual process level that the synchronization activity afforded by MCORTEX through eventcount creation and dynamic data sharing is fully exercised. Figure 3.5 provides the generic configuration of an application task module illustrating the basic organization and designating the required components.

6. **Multiplexing Application Processes**

*a. Discussion*

The final stage in application system implementation is partitioning the unit process modules into discrete subsets for execution at the single board processor level. The success of this partitioning in configuring a fully functional software complex depends on a number of critical areas. First is ensuring that all synchronization variables used in each partition are created in the appropriate cluster initialization module and those necessary for intercluster process synchronization are properly distributed. Second is ensuring that global data structures required by processes in each partition are correctly established in the appropriate cluster Shared Memory and are correctly associated with their proper eventcounts. Third is ensuring that the pointer assignments to cluster shared data structures established in the relation.dat file are correctly reflected in the pointer.ass file for correct process memory addressing. The fourth and final area is ensuring that the initialization module for each partition is correctly tailored to establish the application processes within its executable

```
<procedure name> :   procedure ;
  %replace /* optional-local user labels */
  %include 'sysdef.pli';/* declare MCORTEX Functions */
  %include 'share.dcl'; /* declare system wide vars. */
                        /* and data structs.(PL/I-86)*/
  DECLARE          /* local vars. and data structures */
  /* main */
  %include 'pointer.ass'; /* cluster specific ptr. */
                          /* addrs. to shared data*/
                          /* used in this cluster */
  DO LOOP TO INFINITY
  call AWAIT ('<eventcount 1>'b4,'<threshold value>'b4);
                          .
                          .
                          .
  call AWAIT ('<eventcount n>'b4,'<threshold value>'b4);
                                          .

          user design required PL/I-86 process code


  call ADVANCE ('<eventcount m>'b4);
  END LOOP INFINITY
end <procedure name>;
```

Figure 3.5 Application Module Structure.

module as MCORTEX virtual processors. Once completed and compiled, the partitions of application tasks, initialization and system interface object modules can be linked to produce the application system's executable load modules.

***b. General Link-86 Mechanics***

LINK-86 is a linkage editor provided as a utility under CP/M-86 for combining relocatable object files into single executable command (*.cmd*) files [Ref. 22: pp. 63-80]. This utility is used to combine the object modules resulting from the individual compilation of the application processes PL/I-86 source files with appropriate initialization and system modules. Effectively, the output of the linking process is a set of application tasks which operate under the MCORTEX executive on a single board computer.

For purposes of ease and organization the option of placing linker commands in input files (*<filename>.inp*) for direct linker input is chosen. This allows lengthy lists of object module names designated for one single board computer and the special linker instructions to be maintained in a single, storeable file. Invoking the linker with the contents of the input file requires only specifying the input file name followed directly by the letter *I* in square brackets after entering **LINK86** at the operating system prompt.

```
C>LINK86 <input filename>[I]
```

The required format of a Link-86 input file is provided in Figure 3.6. The linker assumes that input modules are of extension *.obj*, therefore explicit specification is not required after file names.

```
<filename module produced> =
<filename object mod 1><optional linker parameters>,
<filename object mod 2>,
                        .
                        .
<filename object mod n>
```

Figure 3.6 Link-86 Input File Structure.

*c. MCORTEX Multiplexing Specifics*

In the case of linking MCORTEX application modules special linker instructions are required to properly fix code and data segments in compliance with the executive's memory management scheme. The code segment must be absolutely fixed as 0439H and the data segment as 0800H. Additionally, default parameters concerning memory allocation of the data segment must be overridden through the Additional (ad) and Maximum (m) parameters [Ref. 22: pp. 75-76]. The last special requirement of the linker is the production of a *map file* that contains absolute address information needed to complete each process's MCORTEX declaration in the appropriate initialization module. These requirements are fulfilled by including the following parameter specification in each input file.

```
[code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
```

(1) *Using the Map file.* The map file produced by the linker is placed on the same disk from which the linker reads the input file. The map file is created under the filename designated in the input file for the output command module except that its extension is *.map*. Note the file in Figure 3.7.

The first section of the file provides a summary of all code and data segments generated from the linking to produce the associated executable command file. The upper bound of the offset address range of the last entry is the last memory location occupied in the data segment. From this offset address, the Stack Pointer (maximum) address parameter required for the MCORTEX declaration (CREATE_PROC) of each application module in the corresponding initialization module is determined. The calculation requires the addition of the size of the stack required by the first application module to this offset value. For subsequent application modules stack size is simply added to the pointer offset calculated for the previous module. [Ref. 18: p. 75] As a general rule-of-thumb 120H can comfortably be used for all application module stack sizes including the Driver module.

The second section of the file lists in order of occurance individual maps for all modules linked to form the command module. These maps provide offset addresses specific to the code and data segments of each module. The Instruction Pointer parameter required in the CREATE_PROCESS function is taken directly as the *lower* (beginning) offset address of the CODE entry for each individual application module [Ref. 18: p. 76].

It is apparent that these specific parameter values are not available at the time of the linking to produce the Map file. The designer is responsible for assigning *arbitrary values* to the SP and IP parameters in all CREATE_PROC function calls for the initial linking. The extraction of the actual parameter values is then made from the Map file and placed in the appropriate function calls in the initialization module. The initialization module is then recompiled and relinked with the appropriate application object modules to form the final, executable command module.

(2) *Cluster Driver Input File.* Figure 3.8 provides the exact elements of the input file required to produce the cluster's single board dedicated (ECCB) Driver module. The object modules from the compilation of the initialization module (previously described) and the *Sysdev.pli* object module containing the Driver are linked with the object modules from the assembly of the system's *Asmrout.A86* and *Gatemod/trc.A86* source files. The Driver **cannot** be multiplexed with other application

```
Map for file:  SBC1-1.CMD

Segments
--------

Length   Start    Stop   Align Comb  Name             Class

2700  (0000:0005-2704)  BYTE  PUB   CODE             CODE
0519  (0000:0100-0618)  WORD  PUB   DATA             DATA
0021  (0000:061A-063A)  WORD  COM   ?CONSP           DATA
0013  (0000:063C-064E)  WORD  COM   ?FPBSTK          DATA
002E  (0000:0650-067D)  WORD  COM   ?FPB             DATA
0002  (0000:067E-067F)  WORD  COM   ?CNCOL           DATA
0009  (0000:0680-0688)  WORD  COM   ?FILAT           DATA
0008  (0000:068A-0691)  WORD  COM   ?FMTS            DATA
001B  (0000:0692-06AC)  WORD  COM   ?EBUFF           DATA
0003  (0000:06AE-06B0)  WORD  COM   ?ONCOD           DATA
0025  (0000:06B2-06D6)  WORD  COM   SYSIN            DATA
0028  (0000:06D8-06FF)  WORD  COM   SYSPRINT         DATA

Groups           Segments
------           --------

CGROUP  CODE
DGROUP  DATA  ?CONSP  ?FPBSTK  ?FPB
  ?CNCOL  ?FILAT  ?FMTS  ?EBUFF
  ?ONCOD  SYSIN  SYSPRINT

map for module:  SBC1-1i

002A  (0000:0005-002E)  CODE
004A  (0000:0100-0149)  DATA

map for module:  APPL_PROCESS_1

009C  (0000:002F-00CA)  CODE
001E  (0000:014A-0167)  DATA

map for module:  APPL_PROCESS_2

00A2  (0000:00CB-016C)  CODE
0023  (0000:0168-018A)  DATA

map for module:  APPL_PROCESS_3

0038  (0000:016D-01A4)  CODE
000B  (0000:018C-0196)  DATA

map for module:  GATEM/T

0103  (0000:01A5-02A7)  CODE
0004  (0000:0198-019B)  DATA

map for module:  PL1SVBLK

005C  (0000:02A8-0303)  CODE

map for module:  PL1RECOV
```

Figure 3.7 Example Map File.

tasks residing as the sole process in the resulting command module. The assembly routines of the Asmrout module provide Driver required access to hardware resources [Ref. 18: p. 187].

```
<filename module produced> =
<cluster initial. module><required para. specification>,
sysdev,
asmrout,
gatemod
```

Figure 3.8 Cluster Initialization Link-86 Input.

(3) *Application Node Input File.* Figure 3.9 provides the format required for multiplexing application task object modules. The system's *gatemod* object module along with the specific initialization module are the only additional required components necessary to produce a functional executable command module. If required by the application's process distribution, a single task module can be linked.

```
<filename module produced> =
<board initial. module><required para. specification>,
<filename appl. object mod 1>,
                        .
                        .
                        .
<filename appl. object mod n>,
gatemod
```

Figure 3.9 Application Node Link-86 Input.

# IV. E-MCORTEX APPLICATION PROTOTYPING FACILITY

## A. DESIGN CONSIDERATIONS

The fundamental purpose of this prototyping facility is to provide the application system designer with a means to model and analyze the temporal structural features of proposed application systems.

The provisions of this facility support dynamic prototype execution of a structural model of the application system. Allowing prototype execution of the temporal structure early in the design and implementation phases of system evolution reveals design flaws and bottlenecks that would otherwise go undetected until detailed design implementation and integration testing. Discovery of hardware deficiencies or discrepancies in the foundational structure of the software system in these later phases typically results in high development cost overruns and deferred completion schedules. Early prototyping and exploratory modeling of system designs therefore not only optimizes the system foundation before detailed implementation but also maximizes the resources available in the development environment.

## B. IMPLEMENTATION AND UTILIZATION

To provide an accurate prototype model, this facility requires the designer to specify system process parameters that directly shape the temporal character of the application system. This includes an application task synchronization scheme, required interprocess data sharing requirements, estimated task execution times and a system multiprocessing design. See Appendix E. The prototype model executes as the real time structural shell of a proposed design in the MCORTEX multiprocessing environment exercising the available resources of the RTC* hardware suite. Additionally, the facility provides on line diagnostic information to the designer as a direct indicator of the dynamic character of the prototype model without distorting the execution of the model with diagnostic overhead.

### 1. Cluster Initialization

The initialization load module required for each cluster under MCORTEX utilization is fully implemented and linked in the prototyping facility. Cluster one initialization resides under file *simc1.cmd*, cluster two under *simc2.cmd* on each appropriate cluster hard disk. This resource serves as the foundational support for the

prototyping activity but is also totally capable of supporting fully developed and implemented application complexes through proper data structure declaration via the *share.dcl* file, cluster Shared Memory configuration via the *pointer.ass* file and shared data eventcount association via the appropriate cluster *relation.dat* file. The source code of the individual component modules for each cluster initialization is provided in Appendices C and D.

*a. Established Eventcounts*

Each cluster initialization module establishes a full 100 eventcount complement for each of the two clusters currently available. Five categories of eventcounts are pre-established for the designer to draw from in prototyping system designs and constructing final implementations. See Table 2.

TABLE 2
ESTABLISHED EVENTCOUNTS

| CLUSTER ONE EVENTCOUNTS | | CLUSTER TWO EVENTCOUNTS |
|---|---|---|
| 11 to 1F | local no data | 21 to 2F |
| 30 to 3F | local with data | 40 to 4F |
| 50 to 5F | distributed no data | 60 to 6F |
| 70 to 7F | distributed with data | 80 to 8F |
| 01,03,05,07 | initialized to one | 02,04,06,08 |

The first four categories provide eventcounts that fit the two orthogonal properties of eventcount distribution and shared data association. The first eventcount form is for local cluster employment (not distributed) with no associated shared data. The second is for local cluster use but with associated data sharing. Third are eventcounts distributed among clusters with no data sharing and finally the fourth category - distributed with shared data. The designer must ensure that the eventcounts

selected for a specific task meet the functional requirements of the process within the design of the application system. If these requirements change as task modules are relocated or data dependencies are altered, proper eventcount changes reflecting these requirements must likewise be effected.

The fifth category of supplied eventcounts provides a unique property to the prototyping environment. Four per cluster, these eventcounts are initialized to a value of one vice the zero initial value established through standard eventcount creation. This is accomplished simply through a single Advance made on each of these eventcounts by embedding the MCORTEX function calls in the Driver module linked into each cluster initialization load module. The utility of these eventcounts lies in providing a pre-advanced synchronization flag to allow the initial execution of system tasks that rely on external sensors to signal data availability. Following the initial execution subsequent advances can be made throughout the model to allow continued lead task execution.

*b. Simulated Data Sharing*

Interprocess data sharing is a general necessity managed through unique application structures. To provide accurate simulation of data transactions between application processes without requiring declaration of specific data structures, all exchanges are made as byte count transfers between processes and Shared Memory. Global single byte variables are declared and established in Shared Memory via the provided *share.dcl* and *pointer.ass* files. Local task single byte variable declarations are made generically for each modeled application process. All eventcounts created in each cluster designated for associated data sharing are established in the appropriate *relation.dat* file. These eventcounts are associated with a constant, single byte, single queue structure at the first available offset address in each Shared Memory. As a result, all shared data transfers in each cluster are made between local RAM and this single byte Shared Memory location. This is of some consequence within the simulation since single byte data transfer is less efficient than direct assignment statements made between more complex isomorphic data structures, however the differences are moderate when the sharing data structures are small. The prototype requires only the specific number of data bytes consumed from and supplied to Shared Memory for each simulated task.

2. **Task Modeling**

The Generic Process module (*generic.pli*) is designed as a task template to model the variable, temporal parameters of application processes. *Generic.pli* source code is provided in Appendix B. The system designer is responsible for creating a copy of this template under a unique filename for each task to be modeled in the prototype system and providing the appropriate parameter values in each copy to emulate the intended process features. The extracted parameter block of the Generic Process is provided in Table 3.

TABLE 3
GENERIC PROCESS PARAMETERS

```
GENERIC_PROCESS: procedure;
   %replace
      TASK_ID     by     ' ',  /* single character */
      EXEC_TIME_FACTOR  by  00; /* 01 = 0.003 sec*/
      EVCwait_1   by '   'b4,  /* first await evt, */
      EVCwait_2   by '   'b4,  /* comment out as   */
/*    EVCwait_3   by '   'b4      task required,   */
/*    EVCwait_4   by '   'b4,                      */
/*    EVCwait_5   by '   'b4,     last await evt   */
      EVCadvance by '   'b4,  /* signal task done */
      BYTES_IN    by     00,  /*consume share data*/
      BYTES_OUT   by     00,  /*produce share data*/
```

The TASK_ID parameter requires a single character as a unique process identifier. This identifier is written to the CRT of the single board system on which the process is multiplexed to indicate that it is currently under execution. The identifier is repetitively produced to consume the estimated execution time required by the task established through the EXEC_TIME_FACTOR parameter. For each iterative 'put' of the identifier, corresponding to the integer value of the time factor, 0.003 seconds of execution time is consumed. This time factor is exclusive of shared data transfer time which varies independently as a function of the number data bytes transferred.

The generic template can accommodate six eventcounts, five specifying awaited events and one signaling task completion. Two digit hexadecimal eventcount identifiers properly selected from Table 2 are entered as necessary to construct the functional synchronizational model of the process. Additional synchronization function calls and parameters can be included as required and unnecessary provided parameters deleted or commented out.

Specification of shared data transfer requirements of each task is made through the BYTES_IN and BYTES_OUT parameters. The assigned positive integer values represent the number of data bytes consumed by the process from Shared Memory and produced by the process for Shared Memory storage. Data transfers are made via the MULTIBUS between local and cluster RAM resources exercising the actual data paths required in the final implemented application. Once all required processes are modeled for prototype execution, each must be independently compiled.

3. **The Idle Process**

The Idle Process module (*idle.pli*) is provided to indicate when a microprocessor employed in application execution is not actively executing an application process. This is accomplished by establishing the idle process as a MCORTEX process within each multiplexed subset of application process models. The idle process requires no awaited event completions and is therefore always in a ready state. However, through its creation as the lowest possible priority process (FFH) it executes only when *no* other processes at the computing node are ready for execution.

Each complete iterative execution of this process consumes 0.01 seconds of idle processor time. This idleing is an important characteristic of the application system identified by a single character (-) written to the appropriate CRT. This is followed by a MCORTEX advance function call on the reserve eventcount 'FA' hexadecimal which forces a rescheduling of the local MCORTEX processes. If any other local MCORTEX process (application task) is ready, it will execute by virtue of higher priority, however, if no other MCORTEX process has become ready, the idle process will again execute. This repetitive idle cycle will continue until some local MCORTEX process is made ready or 32676 is reached in the idle process loop. Appendix B provides a copy of the idle process source code.

4. **Task Multiplexing**

The multiplexing of task modules on the assigned microprocessor with appropriate initialization modules and the idle process proceeds exactly as presented under Multiplexing Application Processes in Chapter 3. This is necessary in order to execute the prototype under MCORTEX in an environment that accurately reflects the temporal behavior of the design organization. Reconfigurations of this organization in response to diagnostic indicators produced at the appropriate CRTs require the restructuring of initialization modules and linker input files. In addition, reconfigurations may require reassignments of eventcounts to process models in cases of process intercluster relocation or changing shared data requirements.

This may initially appear as a large workload in file editing and recompiling. However the ability to observe and modify the dynamic behavior of the application system's organizational and temporal character prior to detailed modular implementation and integrated real time testing allows the designer to make necessary software and possibly hardware changes in response to system requirements early in the development cycle.

# V. CONCLUDING REMARKS

The goal of this thesis in developing an effective, multiprocess prototyping facility for real time application systems executed under the E-MCORTEX executive is met. The strength of this tool lies in its implementation for execution of the prototype model in the environment in which final detailed application system implementation will be made. This feature ensures the system designer that the observed, dynamic temporal behavior of the prototype is a true and accurate system foundation within the assigned design parameters.

The limitations of this tool revolve around the static designation of parameters in the modeling of system processes. Fixing each process's estimated execution time does not produce a prototype model which reflects a distribution of execution times expected of each unit process. This estimation coupled with a reduction of complex shared data structures to single byte locations renders the model as a mechanism for observing average system performance. Thus the prototype more directly reflects the consequences of the process multiplexing scheme and synchronizational specifications than dynamic variations in the system's executional behavior.

The prototyping facility supports exploratory modeling in the design space of multiprocess systems. Requiring the system designer to provide only those system features that shape the temporal character of the design segregates detailed task implementation until later development phases. This promotes early prototyping and assessment of design structure allowing early detection of hardware deficiencies or software system flaws. Earlier discovery of system design problems in the development cycle promotes efficiency and utilization of development resources.

A byproduct of this facility development is a complete and thorough formalization of the specific requirements for application system implementation under the E-MCORTEX operating executive. Previous documentation focused on the design and implementation issues surrounding the executive itself. The application designer was generally left with only model demonstration programs from which to infer strict application interface requirements and implementation techniques.

Utilization of this prototyping facility can provide early execution of the structural design of application systems. Once this design is tested and optimized, the

model components themselves can easily be modified to integrate the detailed functions of the application processes. This natural transition to the final system implementation suggests that early prototype execution and testing be a standard practice in multiprocess system development.

# APPENDIX A

## SYSDEF.PLI SOURCE FILE

```
/*************************************************************/
/** SYSTEM DEFINITION    SYSDEF.PLI    D.BREWER 1 SEP 84 **/
/**=========================================================**/
/** Source code establishing MCORTEX function interface  **/
/** to be %INCLUDE'd with all application source modules **/
/*************************************************************/

   DECLARE

     advance ENTRY (BIT (8)),
       /* advance (event_count_id) */

     await ENTRY (BIT (8), BIT (16)),
       /* await (event_count_id, awaited_value) */

     create_evc ENTRY (BIT (8)),
       /* create_evc (event_count_id) */

     create_proc ENTRY (BIT (8), BIT (8),
                        BIT (16), BIT (16), BIT (16),
                        BIT (16), BIT (16), BIT (16)),
    /* create_proc (processor_id, processor_priority,    */
    /*              stack_pointer_highest, stack_seg, ip */
    /*              code_seg, data_seg, extra_seg)       */

     create_seq ENTRY (BIT (8)),
       /* create_seq (sequence_id) */

     preempt ENTRY (BIT (8)),
       /* preempt (processor_id) */

     read ENTRY (BIT (8)) RETURNS (BIT (16)),
       /* read (event_count_id) */
       /* RETURNS current_event_count */

     ticket ENTRY (BIT (8)) RETURNS (BIT (16)),
       /* ticket (sequence_id) */
       /* RETURNS unique_ticket_value */

     define_cluster ENTRY  (bit (16)),

       /* define_cluster (local_cluster_address) */

     distribution_map ENTRY (bit (8), bit (8), bit (16)),

    /* distribution_map(distribution_type,id,cluster_addr)*/

     add2bit16 ENTRY (BIT(16), BIT(16)) RETURNS (BIT (16));
       /* add2bit16 ( a_16bit_#, another_16bit_#) */
       /* RETURNS a_16bit_# + another_16bit_#     */


  %replace

    /*-------------------------------------------------
***  EVC$ID's ***

     (1) USER                */

  /* (2) SYSTEM       */
```

```
ERB_READ   by 'fc'b4,
ERB_WRITE  by 'fd'b4,

  /*-----------------------------------------------------
       ***   SEQUENCER NAMES   ***

     (1) USER                */

  /* (2) SYSTEM  */
ERB_WRITE_REQUEST by 'ff'b4,

  /*-----------------------------------------------------
    ***  SHARED VARIABLE POINTERS  ***

     (1) USER               */

/*   (2) SYSTEM  */
block_ptr_value  by '8000'b4,
xmit_ptr_value   by '8078'b4,
rcv_ptr_value    by '8666'b4,

END_RESERVE  by 'FFFF'b4;
```

# APPENDIX B

## GLOBAL PROTOTYPING COMPONENTS

```
/*************************************************************/
/** Generic Task        GENERIC.PLI     D.GARRETT  NOV 86 **/
/**=========================================================**/
/** Model template for simulation of application task     **/
/** processes, independent copy required in unique file   **/
/** for each simulated task;                              **/
/*************************************************************/

GENERIC_PROCESS: procedure;

   %replace
            TASK_ID    by    ' ',  /* single character */

            EVCwait_1  by '  'b4,  /* first await evt, */
            EVCwait_2  by '  'b4,  /* comment out as   */
         /* EVCwait_3  by '  'b4      task required,   */
         /* EVCwait_4  by '  'b4,                      */
         /* EVCwait_5  by '  'b4,     last await evt   */

            EVCadvance by '  'b4,  /* signal task done */

            BYTES_IN   by       ,  /*consume share data*/
            BYTES_OUT  by       ,  /*produce share data*/

            EXEC_TIME_FACTOR  by  15; /*task exec. time*/

   %include  'sysdef.pli';
   %include  'share.dcl';

   DECLARE   /* local RAM  vars. and data structures */

      LOCAL_DATA_IN  fixed binary (7),
      LOCAL_DATA_OUT fixed binary (7),

   k bit (16) static init ('0000'b4), /* threshold var*/
   (i,j) fixed binary (15);

   /* main */

   %include 'pointer.ass';

   do i = 1 to TASK_REPS; /* REPS set in share.dcl file  */

      k = add2bit16 (k,'0001'b4); /* increment threshold */

        call AWAIT (EVCwait_1,k);/*comment out as require*/
        call AWAIT (EVCwait_2,k);
    /*  call AWAIT (EVCwait_3,k);    */
    /*  call AWAIT (EVCwait_4,k);    */
    /*  call AWAIT (EVCwait_5,k);    */

      /* transfer data in from Shared Memory */

      do j = 1 to BYTES_IN;

         LOCAL_DATA_IN = SHARED_DATA_IN;
      end;

      /* expend estimated execution time */

      do j = 1 to EXEC_TIME_FACTOR;
```

```
      put edit (TASK_ID) (a);   /* task id to CRT */
   end;
   /* transfer data out to Shared Memory */
   do j = 1 to BYTES_OUT;
      SHARED_DATA_OUT = LOCAL_DATA_OUT;
   end;
   call  ADVANCE (EVCadvance);
 end;   /* do TASK_REPS */
 call ADVANCE ('fe'b4)'; /* return all CPUs to CP/M-86 */
END GENERIC_PROCESS;
```

```
/*************************************************************/
/** IDLE PROCESS          IDLE.PLI     D.GARRETT  NOV 86 **/
/**=========================================================**/
/** To be linked in all multiplexings of appl. tasks    **/
/** to display idle CPU activity of the local processor, **/
/** must be the lowest priority process in each group.    **/
/*************************************************************/

IDLE_PROCESS: procedure;
  %replace    infinity   by  32676;
  %include    'sysdef.pli';
  DECLARE
    i fixed bin (15);
   /* begin */
   do i = 1 to infinity;
      put edit ('-') (a);
      call ADVANCE ('fa'b4); /* reserve eventcount */
   end;
end IDLE_PROCESS;
```

```
/*****************************************************/
/** APPLICATION INITIALIZATION MOD.   D.GARRETT  NOV 86 **/
/**===============================================**/
/** Tailored for each multiplexed group of appl. tasks  **/
/** to properly initialize each and the required IDLE.pli**/
/** process, linked with task .obj mods. for multiplexing**/
/*****************************************************/

<initialization proc. name>:  proc options (main);

   %include 'sysdef.pli';

   /* create task 1 */
   CALL CREATE_PROC ('<id>'b4,   '<priority>'b4,
                     '<sp>'b4, '0800'b4, '<ip>'b4,
                     '0439'b4, '0800'b4, '0800'b4 );
                  .
                  .
                  .
   /* create task n */
   CALL CREATE_PROC ('<id>'b4,   '<priority>'b4,
                     '<sp>'b4, '0800'b4, '<ip>'b4,
                     '0439'b4, '0800'b4, '0800'b4 );

   /* create IDLE Process */
   CALL CREATE_PROC ('ff'b4,   'ff'b4,
                     '<sp>'b4, '0800'b4, '<ip>'b4,
                     '0439'b4, '0800'b4, '0800'b4 );

   CALL AWAIT ('fe'b4, '01'b4);

end CLUSTER1_INITIALIZATION;
```

```
/*****************************************************/
/** APPLICATION LINK-86 INPUT FILE     D.GARRETT  NOV 86 **/
/**===============================================**/
/** Tailored to link specific appl. task modules created **/
/** with Generic.pli, the required IDLE.obj and GATEMOD  **/
/** to form executable module of multiplexed tasks.      **/
/*****************************************************/

<name .cmd module produced> =
<int.>[code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
<task 1 filename>,
      .
      .
<task n filename>,
idle,
gatemod
```

```
/*****************************************************/
/** GLOBAL DATA          SHARE.DCL      D.GARRETT  NOV 86 **/
/**=================================================**/
/** Global declaration of system wide shared structures  **/
/** for actual exercise of transfer data paths under     **/
/** system prototype execution; num. bytes set by task   **/
/*****************************************************/

%replace  TASK_REPS   by  100; /* sets number of execution*/
                               /* iterations for all tasks*/

DECLARE       /* SHARED DATA */
  (shared_data_pointer) pointer,
  SHARED_DATA_IN  fixed binary (7) based (shared_data_pointer),
  SHARED_DATA_OUT fixed binary (7) based (shared_data_pointer);
```

# APPENDIX C

## CLUSTER 1 PROTOTYPING COMPONENTS

```
/*************************************************************/
/** SIMC1i.PLI                              D.GARRETT  NOV 86 **/
/**=========================================================**/
/** Cluster 1 initialization source code: creates and       **/
/** distributes all necessary eventcounts, establishes      **/
/** cluster DRIVER ECCB module as a MCORTEX process         **/
/*************************************************************/

CLUSTER1_INITIALIZATION:  proc options (main);

   %include 'sysdef.pli';

   CALL DEFINE_CLUSTER ('0001'b4);

   /* value initialized to ONE */

   CALL CREATE_EVC ('01'b4);
   CALL CREATE_EVC ('03'b4);
   CALL CREATE_EVC ('05'b4);
   CALL CREATE_EVC ('07'b4);

   /* 'local' only with NO DATA associated */

   CALL CREATE_EVC ('11'b4);
   CALL CREATE_EVC ('12'b4);
   CALL CREATE_EVC ('13'b4);
   CALL CREATE_EVC ('14'b4);
   CALL CREATE_EVC ('15'b4);
   CALL CREATE_EVC ('16'b4);
   CALL CREATE_EVC ('17'b4);
   CALL CREATE_EVC ('18'b4);
   CALL CREATE_EVC ('19'b4);
   CALL CREATE_EVC ('1a'b4);
   CALL CREATE_EVC ('1b'b4);
   CALL CREATE_EVC ('1c'b4);
   CALL CREATE_EVC ('1d'b4);
   CALL CREATE_EVC ('1e'b4);
   CALL CREATE_EVC ('1f'b4);

   /* 'local' only with  DATA associated */

   CALL CREATE_EVC ('30'b4);
   CALL CREATE_EVC ('31'b4);
   CALL CREATE_EVC ('32'b4);
   CALL CREATE_EVC ('33'b4);
   CALL CREATE_EVC ('34'b4);
   CALL CREATE_EVC ('35'b4);
   CALL CREATE_EVC ('36'b4);
   CALL CREATE_EVC ('37'b4);
   CALL CREATE_EVC ('38'b4);
   CALL CREATE_EVC ('39'b4);
   CALL CREATE_EVC ('3a'b4);
   CALL CREATE_EVC ('3b'b4);
   CALL CREATE_EVC ('3c'b4);
   CALL CREATE_EVC ('3d'b4);
```

```
CALL CREATE_EVC ('3e'b4);
CALL CREATE_EVC ('3f'b4);

/* 'distributed' to cluster 2 with NO DATA associated */

CALL CREATE_EVC ('50'b4);
CALL CREATE_EVC ('51'b4);
CALL CREATE_EVC ('52'b4);
CALL CREATE_EVC ('53'b4);
CALL CREATE_EVC ('54'b4);
CALL CREATE_EVC ('55'b4);
CALL CREATE_EVC ('56'b4);
CALL CREATE_EVC ('57'b4);
CALL CREATE_EVC ('58'b4);
CALL CREATE_EVC ('59'b4);
CALL CREATE_EVC ('5a'b4);
CALL CREATE_EVC ('5b'b4);
CALL CREATE_EVC ('5c'b4);
CALL CREATE_EVC ('5d'b4);
CALL CREATE_EVC ('5e'b4);
CALL CREATE_EVC ('5f'b4);

/* 'distributed' to cluster 2 with  DATA associated */

CALL CREATE_EVC ('70'b4);
CALL CREATE_EVC ('71'b4);
CALL CREATE_EVC ('72'b4);
CALL CREATE_EVC ('73'b4);
CALL CREATE_EVC ('74'b4);
CALL CREATE_EVC ('75'b4);
CALL CREATE_EVC ('76'b4);
CALL CREATE_EVC ('77'b4);
CALL CREATE_EVC ('78'b4);
CALL CREATE_EVC ('79'b4);
CALL CREATE_EVC ('7a'b4);
CALL CREATE_EVC ('7b'b4);
CALL CREATE_EVC ('7c'b4);
CALL CREATE_EVC ('7d'b4);
CALL CREATE_EVC ('7e'b4);
CALL CREATE_EVC ('7f'b4);

/* reserved for IDLE process */

CALL CREATE_EVC ('fa'b4);

/* system Ethernet */

CALL CREATE_EVC (ERB_READ);
CALL CREATE_EVC (ERB_WRITE);
CALL CREATE_SEQ (ERB_WRITE_REQUEST);

/* establish cluster 2 remote copies */

CALL DISTRIBUTION_MAP ('00'b4, '50'b4, '0003'b4);
CALL DISTRIBUTION_MAP ('00'b4, '51'b4, '0003'b4);
CALL DISTRIBUTION_MAP ('00'b4, '52'b4, '0003'b4);
CALL DISTRIBUTION_MAP ('00'b4, '53'b4, '0003'b4);
CALL DISTRIBUTION_MAP ('00'b4, '54'b4, '0003'b4);
CALL DISTRIBUTION_MAP ('00'b4, '55'b4, '0003'b4);
```

```
    CALL DISTRIBUTION_MAP ('00'b4, '56'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '57'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '58'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '59'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '5a'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '5b'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '5c'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '5d'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '5e'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '5f'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '70'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '71'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '72'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '73'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '74'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '75'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '76'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '77'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '78'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '79'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '7a'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '7b'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '7c'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '7d'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '7e'b4, '0003'b4);
    CALL DISTRIBUTION_MAP ('00'b4, '7f'b4, '0003'b4);

    /* create DRIVER */

    CALL CREATE_PROC ('fc'b4,   '80'b4,
                      '2753'b4, '0800'b4, '009b'b4,
                      '0439'b4, '0800'b4, '0800'b4 );

    CALL AWAIT ('fe'b4, '01'b4);

end CLUSTER1_INITIALIZATION;
```

```
{**********************************************************}
{** RELATION.DAT                          D.GARRETT  NOV 86 **}
{**========================================================**}
{** Cluster 1 specific: relates local and distributed    **}
{** eventcounts with shared data to Shared RAM offset;   **}
{** for prototyping all access common memory location    **}
{**********************************************************}

    30          1          8c58       1          1
    31          1          8c58       1          1
    32          1          8c58       1          1
    33          1          8c58       1          1
    34          1          8c58       1          1
    35          1          8c58       1          1
    36          1          8c58       1          1
    37          1          8c58       1          1
    38          1          8c58       1          1
    39          1          8c58       1          1
    3a          1          8c58       1          1
    3b          1          8c58       1          1
    3c          1          8c58       1          1
    3d          1          8c58       1          1
    3e          1          8c58       1          1
    3f          1          8c58       1          1
    70          1          8c58       1          1
    71          1          8c58       1          1
    72          1          8c58       1          1
    73          1          8c58       1          1
    74          1          8c58       1          1
    75          1          8c58       1          1
    76          1          8c58       1          1
    77          1          8c58       1          1
    78          1          8c58       1          1
    79          1          8c58       1          1
    7a          1          8c58       1          1
    7b          1          8c58       1          1
    7c          1          8c58       1          1
    7d          1          8c58       1          1
    7e          1          8c58       1          1
    7f          1          8c58       1          1
    80          1          8c58       1          1
    81          1          8c58       1          1
    82          1          8c58       1          1
    83          1          8c58       1          1
    84          1          8c58       1          1
    85          1          8c58       1          1
    86          1          8c58       1          1
    87          1          8c58       1          1
    88          1          8c58       1          1
    89          1          8c58       1          1
    8a          1          8c58       1          1
    8b          1          8c58       1          1
    8c          1          8c58       1          1
    8d          1          8c58       1          1
    8e          1          8c58       1          1
    8f          1          8c58       1          1
    00          0          0000       0          0
```

```
/*************************************************************
** SIMC1.INP                                  D.GARRETT  NOV 86 **
**=========================================================**
** Cluster 1 specific: Initialization and Driver Link86 **
** input file, produces SIMC1.cmd load module for        **
** cluster 1 initialization.                             **
*************************************************************/

simc1 =
simc1i[code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
sysdev,
asmrout,
gatemod
```

```
/*************************************************************
** ADDRESS.DAT                                D.GARRETT  NOV 86 **
**=========================================================**
** Static file contents for Cluster 1 establishing       **
** Ethernet addressing info. and local cluster address   **
*************************************************************/

1,
'00000000'b,'00000001'b,
'00000000'b,'00000001'b
```

```
/*************************************************************
** POINTER.ASS                                D.GARRETT  NOV 86 **
**=========================================================**
** Offset address for shared data structures required    **
** by all tasks in cluster 1; for prototype all data     **
** transfers to and from common memory location          **
*************************************************************/

unspec(shared_data_pointer)  =  '8c58'b4;
```

# APPENDIX D

## CLUSTER 2 PROTOTYPING COMPONENTS

```
/*************************************************************/
/** SIMC2i.pli                                  D.GARRETT  NOV 86 **/
/**=========================================================**/
/** Cluster 2 initialization source code: creates and       **/
/** distributes all necessary eventcounts, establishes      **/
/** cluster DRIVER ECCB module as a MCORTEX process         **/
/*************************************************************/

CLUSTER2_INITIALIZATION:  proc options (main);
   %include 'sysdef.pli';
   CALL DEFINE_CLUSTER ('0002'b4);

   /* value initialized to ONE */

   CALL CREATE_EVC ('02'b4);
   CALL CREATE_EVC ('04'b4);
   CALL CREATE_EVC ('06'b4);
   CALL CREATE_EVC ('08'b4);

   /* 'local' only with NO DATA associated */

   CALL CREATE_EVC ('20'b4);
   CALL CREATE_EVC ('21'b4);
   CALL CREATE_EVC ('22'b4);
   CALL CREATE_EVC ('23'b4);
   CALL CREATE_EVC ('24'b4);
   CALL CREATE_EVC ('25'b4);
   CALL CREATE_EVC ('26'b4);
   CALL CREATE_EVC ('27'b4);
   CALL CREATE_EVC ('28'b4);
   CALL CREATE_EVC ('29'b4);
   CALL CREATE_EVC ('2a'b4);
   CALL CREATE_EVC ('2b'b4);
   CALL CREATE_EVC ('2c'b4);
   CALL CREATE_EVC ('2d'b4);
   CALL CREATE_EVC ('2e'b4);
   CALL CREATE_EVC ('2f'b4);

   /* 'local' only with  DATA associated */

   CALL CREATE_EVC ('40'b4);
   CALL CREATE_EVC ('41'b4);
   CALL CREATE_EVC ('42'b4);
   CALL CREATE_EVC ('43'b4);
   CALL CREATE_EVC ('44'b4);
   CALL CREATE_EVC ('45'b4);
   CALL CREATE_EVC ('46'b4);
   CALL CREATE_EVC ('47'b4);
   CALL CREATE_EVC ('48'b4);
   CALL CREATE_EVC ('49'b4);
   CALL CREATE_EVC ('4a'b4);
   CALL CREATE_EVC ('4b'b4);
```

```
CALL CREATE_EVC ('4c'b4);
CALL CREATE_EVC ('4d'b4);
CALL CREATE_EVC ('4e'b4);
CALL CREATE_EVC ('4f'b4);

/* 'distributed' to cluster 1 with NO DATA associated */

CALL CREATE_EVC ('60'b4);
CALL CREATE_EVC ('61'b4);
CALL CREATE_EVC ('62'b4);
CALL CREATE_EVC ('63'b4);
CALL CREATE_EVC ('64'b4);
CALL CREATE_EVC ('65'b4);
CALL CREATE_EVC ('66'b4);
CALL CREATE_EVC ('67'b4);
CALL CREATE_EVC ('68'b4);
CALL CREATE_EVC ('69'b4);
CALL CREATE_EVC ('6a'b4);
CALL CREATE_EVC ('6b'b4);
CALL CREATE_EVC ('6c'b4);
CALL CREATE_EVC ('6d'b4);
CALL CREATE_EVC ('6e'b4);
CALL CREATE_EVC ('6f'b4);

/* 'distributed' to cluster 2 with  DATA associated */

CALL CREATE_EVC ('80'b4);
CALL CREATE_EVC ('81'b4);
CALL CREATE_EVC ('82'b4);
CALL CREATE_EVC ('83'b4);
CALL CREATE_EVC ('84'b4);
CALL CREATE_EVC ('85'b4);
CALL CREATE_EVC ('86'b4);
CALL CREATE_EVC ('87'b4);
CALL CREATE_EVC ('88'b4);
CALL CREATE_EVC ('89'b4);
CALL CREATE_EVC ('8a'b4);
CALL CREATE_EVC ('8b'b4);
CALL CREATE_EVC ('8c'b4);
CALL CREATE_EVC ('8d'b4);
CALL CREATE_EVC ('8e'b4);
CALL CREATE_EVC ('8f'b4);

/* reserved for IDLE process */

CALL CREATE_EVC ('fa'b4);

/* system Ethernet */

CALL CREATE_EVC (ERB_READ);
CALL CREATE_EVC (ERB_WRITE);
CALL CREATE_SEQ (ERB_WRITE_REQUEST);

/* establish cluster 2 remote copies */

CALL DISTRIBUTION_MAP ('00'b4, '60'b4, '0003'b4);
CALL DISTRIBUTION_MAP ('00'b4, '61'b4, '0003'b4);
CALL DISTRIBUTION_MAP ('00'b4, '62'b4, '0003'b4);
CALL DISTRIBUTION_MAP ('00'b4, '63'b4, '0003'b4);
```

```
   CALL DISTRIBUTION_MAP ('00'b4, '64'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '65'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '66'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '67'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '68'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '69'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '6a'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '6b'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '6c'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '6d'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '6e'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '6f'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '80'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '81'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '82'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '83'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '84'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '85'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '86'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '87'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '88'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '89'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '8a'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '8b'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '8c'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '8d'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '8e'b4, '0003'b4);
   CALL DISTRIBUTION_MAP ('00'b4, '8f'b4, '0003'b4);

   /* create DRIVER */

   CALL CREATE_PROC ('fc'b4,   '80'b4,
                     '27c1'b4, '0800'b4, '0293'b4,
                     '0439'b4, '0800'b4, '0800'b4 );

   CALL AWAIT ('fe'b4, '01'b4);

end CLUSTER2_INITIALIZATION;
```

```
*************************************************************
** RELATION.DAT                              D.GARRETT  NOV 86 **
**=========================================================**
** Cluster 2 specific: relates local and distributed       **
** eventcounts with shared data to shared RAM offset;      **
** for prototyping all access common memory location       **
*************************************************************

    40          1          8c58        1          1
    41          1          8c58        1          1
    42          1          8c58        1          1
    43          1          8c58        1          1
    44          1          8c58        1          1
    45          1          8c58        1          1
    46          1          8c58        1          1
    47          1          8c58        1          1
    48          1          8c58        1          1
    49          1          8c58        1          1
    4a          1          8c58        1          1
    4b          1          8c58        1          1
    4c          1          8c58        1          1
    4d          1          8c58        1          1
    4e          1          8c58        1          1
    4f          1          8c58        1          1
    70          1          8c58        1          1
    71          1          8c58        1          1
    72          1          8c58        1          1
    73          1          8c58        1          1
    74          1          8c58        1          1
    75          1          8c58        1          1
    76          1          8c58        1          1
    77          1          8c58        1          1
    78          1          8c58        1          1
    79          1          8c58        1          1
    7a          1          8c58        1          1
    7b          1          8c58        1          1
    7c          1          8c58        1          1
    7d          1          8c58        1          1
    7e          1          8c58        1          1
    7f          1          8c58        1          1
    80          1          8c58        1          1
    81          1          8c58        1          1
    82          1          8c58        1          1
    83          1          8c58        1          1
    84          1          8c58        1          1
    85          1          8c58        1          1
    86          1          8c58        1          1
    87          1          8c58        1          1
    88          1          8c58        1          1
    89          1          8c58        1          1
    8a          1          8c58        1          1
    8b          1          8c58        1          1
    8c          1          8c58        1          1
    8d          1          8c58        1          1
    8e          1          8c58        1          1
    8f          1          8c58        1          1
    00          0          0000        0          0
```

```
/*************************************************************
** SIMC2.INP                                  D.GARRETT  NOV 86 **
**=========================================================**
** Cluster 2 specific: Initialization and Driver Link86 **
** input file, produces SIMC2.cmd load module for       **
** cluster 2 initialization.                            **
*************************************************************/

simc2 =
simc2i[code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
sysdev,
asmrout,
gatemod
```

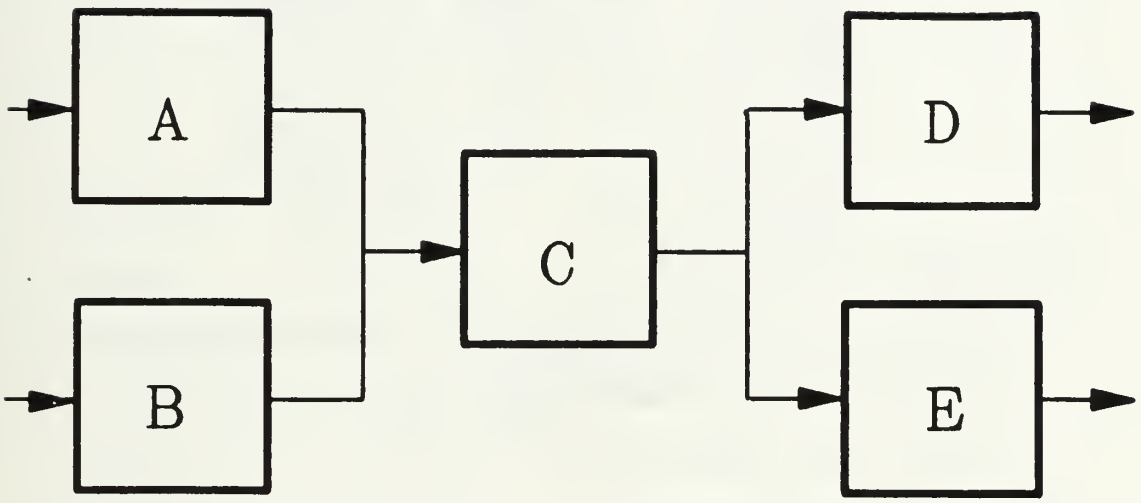
```
/*************************************************************
** ADDRESS.DAT                                D.GARRETT  NOV 86 **
**=========================================================**
** Static file contents for Cluster 2 establishing      **
** Ethernet addressing info. and local cluster address  **
*************************************************************/
]
'00000000'b,'00000010'b,
'00000000'b,'00000010'b
```

```
/*************************************************************
** POINTER.ASS                                D.GARRETT  NOV 86 **
**=========================================================**
** Offset address for shared data structures required   **
** by all tasks in cluster 2; for prototype all data    **
** transfers to and from common memory location         **
*************************************************************/

unspec(shared_data_pointer)  =  '8c58'b4;
```

# APPENDIX E
# PROTOTYPE DEMONSTRATION

SYSTEM UNIT PROCESS DIAGRAM




SYSTEM SPECIFICATIONS

| ID | Exec_Time | Node | Priority | Await | Advance | Bytes: in | out |
|---|---|---|---|---|---|---|---|
| A | 20 | 2-1 | 2 | 2,4 | 41 | 6 | 6 |
| B | 20 | 2-2 | 3 | 2,4 | 42 | 6 | 6 |
| C | 50 | 2-2 | 2 | 6,8,41,42 | 2,4,43 | 12 | 6 |
| D | 30 | 2-2 | 1 | 43 | 6 | 6 | 6 |
| E | 70 | 2-1 | 1 | 43 | 8 | 6 | 6 |

**** CLUSTER 2 - SBC 1 COMPONENTS ****

```
/*****************************************************/
/** pA.pli                              D.GARRETT  NOV 86 **/
/**=================================================**/
/** Task 'A' process model                          **/
/*****************************************************/
GENERIC_PROCESS: procedure;

   %replace
            TASK_ID      by   'A', /*  single character */

            EXEC_TIME_FACTOR  by   20;

            EVCwait_1    by '02'b4, /* eventcount id    */
            EVCwait_2    by '04'b4,
     /*     EVCwait_3    by  */
     /*     EVCwait_4    by  */
     /*     EVCwait_5    by  */

            EVCadvance   by  '41'b4,

            BYTES_IN     by         6,
            BYTES_OUT    by         6;

   %include  'sysdef.pli';
   %include  'share.dcl';

   DECLARE   /* process local vars. and data structures */

      LOCAL_DATA_IN   fixed binary (7),
      LOCAL_DATA_OUT  fixed binary (7),

      k bit (16) static init ('0000'b4),
      (i,j) fixed binary (15);

   %include 'pointer.ass';

   do i = 1 to TASK_REPS;

      k = add2bit16(k,'0001'b4);

      call AWAIT (EVCwait_1,k);
      call AWAIT (EVCwait_2,k);
/*    call AWAIT (EVCwait_3,k);    */
/*    call AWAIT (EVCwait_4,k);    */
/*    call AWAIT (EVCwait_5,k);    */

      do j = 1 to BYTES_IN;

         LOCAL_DATA_IN = SHARED_DATA_IN;
      end;

      do j = 1 to EXEC_TIME_FACTOR;

         put edit (TASK_ID) (a);
      end;

      do j = 1 to BYTES_OUT;

         SHARED_DATA_OUT  =  LOCAL_DATA_OUT;
      end;

      call  ADVANCE (EVCadvance);

   end;   /* do task reps */

END GENERIC_PROCESS;
```

```
/*****************************************************/
/** pE.pli                              D.GARRETT  NOV 86 **/
/**=================================================**/
/** Task 'E' process model                          **/
/*****************************************************/
GENERIC_PROCESS: procedure;

   %replace
            TASK_ID     by     'E', /*  single character */

            EXEC_TIME_FACTOR  by      70;

            EVCwait_1   by   '43'b4, /* eventcount id    */
     /*     EVCwait_2   by  */
     /*     EVCwait_3   by  */
     /*     EVCwait_4   by  */
     /*     EVCwait_5   by  */

            EVCadvance  by     '08'b4,

            BYTES_IN    by         6,
            BYTES_OUT   by         6,

   %include  'sysdef.pli';
   %include  'share.dcl';

   DECLARE   /* process local vars. and data structures */

      LOCAL_DATA_IN   fixed binary (7),
      LOCAL_DATA_OUT  fixed binary (7),

      k bit (16) static init ('0000'b4),
      (i,j) fixed binary (15);

   %include 'pointer.ass';

   do i = 1 to TASK_REPS;

      k = add2bit16(k,'0001'b4);

      call AWAIT (EVCwait_1,k);
/*    call AWAIT (EVCwait_2,k);   */
/*    call AWAIT (EVCwait_3,k);   */
/*    call AWAIT (EVCwait_4,k);   */
/*    call AWAIT (EVCwait_5,k);   */

      do j = 1 to BYTES_IN;

         LOCAL_DATA_IN = SHARED_DATA_IN;
      end;

      do j = 1 to EXEC_TIME_FACTOR;

         put edit (TASK_ID) (a);
      end;

      do j = 1 to BYTES_OUT;

         SHARED_DATA_OUT  =  LOCAL_DATA_OUT;
      end;

      call  ADVANCE (EVCadvance);

   end;   /* do task reps */

END GENERIC_PROCESS;
```

```
/*************************************************************/
/* 2_1i.pli                          D.GARRETT  NOV 86       */
/*===========================================================*/
/* Initialization module Cluster 2 SBC 1 creating tasks      */
/* A,E and IDLE as MCORTEX processes;                        */
/*************************************************************/

C2_1i: procedure options (main);

      /* initialization for cluster 2 board 1 */

%include  'sysdef.pli';

/* init. for pA */

CALL CREATE_PROC  ('0a'b4,    '02'b4,
                   '0820'b4,    '0800'b4,    '002f'b4,
                   '0439'b4,    '0800'b4,    '0800'b4 );

/* init. for pE */

CALL CREATE_PROC  ('0e'b4,    '01'b4,
                   '0940'b4,    '0800'b4,    '00d1'b4,
                   '0439'b4,    '0800'b4,    '0800'b4 );

/* init. for IDLE */

CALL CREATE_PROC  ('ff'b4,    'ff'b4,
                   '0a60'b4,    '0800'b4,    '016d'b4,
                   '0439'b4,    '0800'b4,    '0800'b4 );

CALL AWAIT  ('fe'b4, '01'b4);

end C2_1i;
```

```
/*************************************************************/
/* 2-1.inp                           D.GARRETT  NOV 86       */
/*===========================================================*/
/* Link86 input file Cluster 2 SBC 1 multiplexing task       */
/* A,E and IDLE;                                             */
/*************************************************************/

C21 =
2-1i [code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
pa,
pe,
idle,
gatemod
```

```
(************************************************************)
(** 2-1.map                                  D.GARRETT  NOV 86 **)
(**========================================================**)
(** Link86 map produced for Cluster 2 SBC 1                **)
(************************************************************)

Map for file:  2-1.CMD

Segments
--------

Length   Start    Stop   Align Comb  Name            Class

2700  (0000:0005-2704)  BYTE  PUB  CODE            CODE
051B  (0000:0100-061A)  WORD  PUB  DATA            DATA
0021  (0000:061C-063C)  WORD  COM  ?CONSP          DATA
0013  (0000:063E-0650)  WORD  COM  ?FPBSTK         DATA
002E  (0000:0652-067F)  WORD  COM  ?FPB            DATA
0002  (0000:0680-0681)  WORD  COM  ?CNCOL          DATA
0009  (0000:0682-068A)  WORD  COM  ?FILAT          DATA
0008  (0000:068C-0693)  WORD  COM  ?FMTS           DATA
001B  (0000:0694-06AE)  WORD  COM  ?EBUFF          DATA
0003  (0000:06B0-06B2)  WORD  COM  ?ONCOD          DATA
0025  (0000:06B4-06D8)  WORD  COM  SYSIN           DATA
0028  (0000:06DA-0701)  WORD  COM  SYSPRINT        DATA

Groups          Segments
------          --------

CGROUP  CODE
DGROUP  DATA  ?CONSP  ?FPBSTK  ?FPB
  ?CNCOL  ?FILAT  ?FMTS  ?EBUFF
  ?ONCOD  SYSIN  SYSPRINT

map for module:  C2_1I

002A  (0000:0005-002E)  CODE
004C  (0000:0100-014B)  DATA

map for module:  GENERIC_PROCESS

00A2  (0000:002F-00D0)  CODE
0023  (0000:014C-016E)  DATA

map for module:  GENERIC_PROCESS

009C  (0000:00D1-016C)  CODE
001E  (0000:0170-018D)  DATA

map for module:  IDLE_PROCESS

0038  (0000:016D-01A4)  CODE
000B  (0000:018E-0198)  DATA

map for module:  GATEM/T

0103  (0000:01A5-02A7)  CODE
0004  (0000:019A-019D)  DATA
```

**** CLUSTER 2 - SBC 2 COMPONENTS ****

```
/*****************************************************/
/** pB.pli                              D.GARRETT  NOV 86 **/
/**=================================================**/
/** Task 'B' process model                          **/
/*****************************************************/
GENERIC_PROCESS: procedure;

   %replace
            TASK_ID      by    'B', /*  single character */
            EXEC_TIME_FACTOR  by      20;
            EVCwait_1    by '02'b4, /* eventcount id    */
            EVCwait_2    by '04'b4,
     /*     EVCwait_3    by  */
     /*     EVCwait_4    by  */
     /*     EVCwait_5    by  */
            EVCadvance   by  '42'b4,
            BYTES_IN     by       6,
            BYTES_OUT    by       6,

   %include  'sysdef.pli';
   %include  'share.dcl';

   DECLARE   /* process local vars. and data structures */

      LOCAL_DATA_IN   fixed binary (7),
      LOCAL_DATA_OUT  fixed binary (7),

      k bit (16) static init ('0000'b4),
      (i,j) fixed binary (15);

   %include 'pointer.ass';

   do i = 1 to TASK_REPS;

      k = add2bit16(k,'0001'b4);

      call AWAIT (EVCwait_1,k);
      call AWAIT (EVCwait_2,k);
/*    call AWAIT (EVCwait_3,k);   */
/*    call AWAIT (EVCwait_4,k);   */
/*    call AWAIT (EVCwait_5,k);   */

      do j = 1 to BYTES_IN;

         LOCAL_DATA_IN = SHARED_DATA_IN;
      end;

      do j = 1 to EXEC_TIME_FACTOR;

         put edit (TASK_ID) (a);
      end;

      do j = 1 to BYTES_OUT;

         SHARED_DATA_OUT  =  LOCAL_DATA_OUT;
      end;

      call  ADVANCE (EVCadvance);

   end;   /* do task reps */

END GENERIC_PROCESS;
```

```
/*************************************************************/
/** pC.pli                                  D.GARRETT  NOV 86 **/
/**=========================================================**/
/** Task 'C' process model                                  **/
/*************************************************************/
GENERIC_PROCESS: procedure;

   %replace
            TASK_ID      by    'C', /*  single character */

            EXEC_TIME_FACTOR  by       50;

            EVCwait_1    by   '41'b4, /* eventcount id    */
            EVCwait_2    by    '42'b4,
            EVCwait_3    by    '06'b4,
            EVCwait_4    by    '08'b4,
     /*     EVCwait_5    by  */

            EVCadvance_1     by        '02'b4,
            EVCadvance_2     by        '04'b4,
            EVCadvance_3     by        '43'b4,

            BYTES_IN         by         12,
            BYTES_OUT        by         6,

   %include  'sysdef.pli';
   %include  'share.dcl';

   DECLARE   /* process local vars. and data structures */

      LOCAL_DATA_IN   fixed binary (7),
      LOCAL_DATA_OUT  fixed binary (7),

      k bit (16) static init ('0000'b4),
      (i,j) fixed binary (15);

   %include 'pointer.ass';

   do i = 1 to TASK_REPS;

      k = add2bit16(k,'0001'b4);

      call AWAIT (EVCwait_1,k);
      call AWAIT (EVCwait_2,k);
      call AWAIT (EVCwait_3,k);
      call AWAIT (EVCwait_4,k);
/*    call AWAIT (EVCwait_5,k);   */

      do j = 1 to BYTES_IN;

         LOCAL_DATA_IN = SHARED_DATA_IN;
      end;

      do j = 1 to EXEC_TIME_FACTOR;

         put edit (TASK_ID) (a);
      end;

      do j = 1 to BYTES_OUT;

         SHARED_DATA_OUT  =  LOCAL_DATA_OUT;
      end;

      call  ADVANCE (EVCadvance_1);
      call  ADVANCE (EVCadvance_2);
      call  ADVANCE (EVCadvance_3);
   end;   /* do task reps */

END GENERIC_PROCESS;
```

```
/*****************************************************************
** pD.pli                                    D.GARRETT  NOV 86 **
**===============================================================**
** Task 'D' process model                                       **
*****************************************************************/
GENERIC_PROCESS: procedure;

   %replace
            TASK_ID      by    'D', /*  single character */

            EXEC_TIME_FACTOR  by      30;

            EVCwait_1    by    '43'b4, /* eventcount id   */
     /*     EVCwait_2    by  */
     /*     EVCwait_3    by  */
     /*     EVCwait_4    by  */
     /*     EVCwait_5    by  */

            EVCadvance   by         '06'b4,

            BYTES_IN     by         6,
            BYTES_OUT    by         6,


   %include  'sysdef.pli';
   %include  'share.dcl';

   DECLARE   /* process local vars. and data structures */

      LOCAL_DATA_IN   fixed binary (7),
      LOCAL_DATA_OUT  fixed binary (7),

      k bit (16) static init ('0000'b4),
      (i,j) fixed binary (15);

   %include 'pointer.ass';

   do i = 1 to TASK_REPS;

      k = add2bit16(k,'0001'b4);

      call AWAIT (EVCwait_1,k);
/*    call AWAIT (EVCwait_2,k);    */
/*    call AWAIT (EVCwait_3,k);    */
/*    call AWAIT (EVCwait_4,k);    */
/*    call AWAIT (EVCwait_5,k);    */

      do j = 1 to BYTES_IN;

         LOCAL_DATA_IN = SHARED_DATA_IN;
      end;

      do j = 1 to EXEC_TIME_FACTOR;

         put edit (TASK_ID) (a);
      end;

      do j = 1 to BYTES_OUT;

         SHARED_DATA_OUT  =  LOCAL_DATA_OUT;
      end;

      call  ADVANCE (EVCadvance);

   end;   /* do task reps */

END GENERIC_PROCESS;
```

```
/*************************************************************
** 2_2i.pli                                  D.GARRETT  NOV 86 **
**=========================================================**
** Initialization module Cluster 2 SBC 2 creating tasks    **
** B,C,D and IDLE as MCORTEX processes;                    **
*************************************************************/

C2_2i: procedure options (main);

      /* initialization for cluster 2 board 2  */

%include  'sysdef.pli';

/* init. for pB */

CALL CREATE_PROC  ('0b'b4,    '03'b4,
                   '0863'b4,    '0800'b4,    '0035'b4,
                   '0439'b4,    '0800'b4,    '0800'b4 );

/* init. for pC */

CALL CREATE_PROC  ('0c'b4,    '02'b4,
                   '0983'b4,    '0800'b4,    '00d7'b4,
                   '0439'b4,    '0800'b4,    '0800'b4 );

/* init. for pD */

CALL CREATE_PROC  ('0d'b4,    '01'b4,
                   '0aa3'b4,    '0800'b4,    '0191'b4,
                   '0439'b4,    '0800'b4,    '0800'b4 );

/* init.for IDLE */

CALL CREATE_PROC  ('ff'b4,    'ff'b4,
                   '0bc3'b4,    '0800'b4,    '022d'b4,
                   '0439'b4,    '0800'b4,    '0800'b4 );

CALL AWAIT  ('fe'b4, '01'b4);

end C2_2i;
```

```
/*************************************************************
** 2-2.inp                                   D.GARRETT  NOV 86 **
**=========================================================**
** Link86 input file Cluster 2 SBC 2 multiplexing task     **
** B,C,D and IDLE;                                         **
*************************************************************/

C22 =
2-2i [code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
pb,
pc,
pd,
idle,
gatemod
```

```
(**********************************************************)
(** 2-2.map                                  D.GARRETT  NOV 86 **)
(**=======================================================**)
(** Link86 map produced for Cluster 2 SBC 2                **)
(**********************************************************)

Map for file:  2-2.CMD

Segments
--------

Length   Start    Stop   Align Comb  Name            Class

27C0  (0000:0005-27C4)  BYTE  PUB  CODE            CODE
0565  (0000:0100-0664)  WORD  PUB  DATA            DATA
0021  (0000:0666-0686)  WORD  COM  ?CONSP          DATA
0013  (0000:0688-069A)  WORD  COM  ?FPBSTK         DATA
002E  (0000:069C-06C9)  WORD  COM  ?FPB            DATA
0002  (0000:06CA-06CB)  WORD  COM  ?CNCOL          DATA
0009  (0000:06CC-06D4)  WORD  COM  ?FILAT          DATA
0008  (0000:06D6-06DD)  WORD  COM  ?FMTS           DATA
001B  (0000:06DE-06F8)  WORD  COM  ?EBUFF          DATA
0003  (0000:06FA-06FC)  WORD  COM  ?ONCOD          DATA
0025  (0000:06FE-0722)  WORD  COM  SYSIN           DATA
0028  (0000:0724-074B)  WORD  COM  SYSPRINT        DATA

Groups          Segments
------          --------

CGROUP  CODE
DGROUP  DATA  ?CONSP  ?FPBSTK  ?FPB
  ?CNCOL  ?FILAT  ?FMTS  ?EBUFF
  ?ONCOD  SYSIN  SYSPRINT

map for module:  C2_2I

0030  (0000:0005-0034)  CODE
0062  (0000:0100-0161)  DATA

map for module:  GENERIC_PROCESS

00A2  (0000:0035-00D6)  CODE
0023  (0000:0162-0184)  DATA

map for module:  GENERIC_PROCESS

00BA  (0000:00D7-0190)  CODE
0033  (0000:0186-01B8)  DATA

map for module:  GENERIC_PROCESS

009C  (0000:0191-022C)  CODE
001E  (0000:01BA-01D7)  DATA

map for module:  IDLE_PROCESS

0038  (0000:022D-0264)  CODE
000B  (0000:01D8-01E2)  DATA

map for module:  GATEM/T

0103  (0000:0265-0367)  CODE
0004  (0000:01E4-01E7)  DATA
```

# APPENDIX F

## POWER UP AND APPLICATION LOADING

I. Power Up: Cluster 1 and/or 2

1. Turn on the MULTIBUS power cage.
2. Turn on the iSBC 201 disk drive.
3. Turn on the Micropolis hard disk drive.
4. Turn on the required ADM-3A terminals.

II. System Boot: Cluster 1 and/or 2

1. Press the RESET button on the MULTIBUS power cage.
2. Insert the CP/M-86 System Disk in drive zero (A).
3. Insert application or utility disk in drive one (B).
4. Terminal 1 (Master) type: U
5. Following prompt type: gffd4:0 <RETURN>
6. Following prompt select console 1, disk A.
7. After A> type: ldcpm <RETURN>
8. After A> type: ldboot <RETURN>

For each remaining required terminal:

9. Type: U
10. Following prompt type: ge000:400 <RETURN>
11. Following prompt select unique console and disk.

III. MCORTEX System Loading: Cluster 1 and/or 2

1. Terminal 1 (Master) type: MCORTEX <RETURN>
   note: MCORTEX can be loaded from any disk holding the MCORTEX.CMD file however the Driver module will read relation.dat and address.dat only from the disk from which MCORTEX is loaded.
2. At prompt for global memory loading type: Y <RETURN>
3. At remaining terminals type: MCORTEX <RETURN>
4. At prompt for global memory loading hit: <RETURN>

IV. Application Loading: Cluster 1 and/or 2

1. At ANY Terminal other than 1 (master) type:
   <disk>:<cluster initialization filename>.CMD <RETURN>
2. At remaining required terminals type:
   <disk>:<application module filename>.CMD <RETURN>

# APPENDIX G
# GLOSSARY OF KEY TERMS

**CLUSTER** - a tightly coupled assemblage of asynchronous computing nodes capable of synchronous activity around common cluster resources;

**COMMON MEMORY** - 32K, 8 bit cluster level RAM resource resident on the Multibus established under MCORTEX and accessible only by the MCORTEX kernel for system variables and data structures, addressable under segment E000H offset 0000H to 7FFFH;

**DISTRIBUTED EVENTCOUNT** - MCORTEX variable specifically created for synchronizing the execution of MCORTEX user processes resident in different clusters;

**E-MCORTEX** - (Extended-MCORTEX) currently implemented executive providing primitives for synchronous action of parallel processes and application declared interprocess data sharing throughout the RTC* architecture;

**EVENTCOUNT** - primary MCORTEX system variable created for application process synchronization, instantiated with two digit hexadecimal names these variables are initialized to zero and hold positive integer values, MCORTEX allows creation of 100 eventcounts per cluster;

**MCORTEX PROCESS** - an application system process identified and established under the MCORTEX executive through the function CREATE_PROC;

**LOCAL EVENTCOUNT** - MCORTEX system variable used for application process synchronization through MCORTEX function calls operating strictly within a single cluster;

**RTC*** - a hierarchically bus structured, multi-microprocessor system architecture designed and developed under the AEGIS Modeling Group for embedded, concurrent real time software system modeling and development at the Naval Postgraduate School, Monterey, California;

**SHARED MEMORY** - 32K, 8 bit cluster level RAM resource resident on the Multibus designer configured under MCORTEX for application interprocess data sharing, addressable under segment 0800H offset 8000H to FFFFH;

**VIRTUAL PROCESSOR** - an application process created as a MCORTEX process via appropriate CREATE_PROC function call from a cluster initialization module, virtual processes are scheduled for execution only on the computing node at which they are multiplexed and loaded;

## LIST OF REFERENCES


1. Dilmore, W. D., *The INTEL MCS-86 as a Functionally Dedicated Micropressor in AN/SYP-1A Radar Control*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1980.

2. Gayler, R. A., *A Multimicroprocessing Approach to the AEGIS Combat System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1980.

3. Riche, R. S. and Williams, C. E., *A Software Foundation for AN/SPY-1A Radar Control*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1981.

4. Wasson, W. J., *Detailed Design of the Kernel of a Real-Time Multiprocessor Operating System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1980.

5. Swan, R. J. and others, "Cm* - A Modular Multi-Microprocessor," *Proceedings of the National Computer Conference*, 1977.

6. INTEL Corporation, *iSBC 86/12 Single Board Computer Hardware Reference Manual*, 1976.

7. Lear Siegler, Inc., *ADM-3A Operators Manual*, 1979.

8. Intel Corporation, *Intel MULTIBUS Interfacing*, 1979.

9. Intel Corporation, *Diskette Operating System, MDS-DOS Hardware Reference Manual*, 1976.

10. Micropolis, *Model 1220/1200 Series Technical Notes.*

11. InterLAN Corporation, *NI3010 MULTIBUS ETHERNET Communication Controller User Manual*, 1982.

12. Xerox Corporation, *The ETHERNET - A Local Area Network: Data Link and Physical Layer Specification*, Version 1.0, September 1980.

13. Reed, D. P. and Kanodia, R. T., "Synchronization with Eventcounts and Sequencers," *Communication of the ACM*, v. 22, pp. 115-123, February 1979.

14. Rapantzikos, D. K., *Detailed Design of the Kernel of a Real-Time Multiprocessor Operating System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1981.

15. Cox, E. R., *A Real-Time, Distributed Operating System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1981.

16. Klinefelter, S. G., *Implementation of a Real-Time, Distributed Operating System for a Multi Computer System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1982.

17. Rowe, W. R., *Adaption of MCORTEX to the AEGIS Simulation Environment*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1984.

18. Brewer, D. J., *A Real-Time Executive for Multiple Computer Clusters*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1984.

19. Heager, R., *Process Synchronization and Data Communication between Processes in Real Time Local Area Networks*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1985.

20. Digital Research, *PL/I Language Reference Manual*, 1982.

21. Digital Research, *PL/I Programmer's Guide*, 1982.

22. Digital Research, *Programmer's Utilities Guide*, 1982.

## INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943-5002 | 2 |
| 3. | Department Chairman, Code 52<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 4. | Dr. Uno R. Kodres, Code 52Kr<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943 | 3 |
| 5. | CDR Gary S. Baker, Code 52Bj<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 6. | Daniel Green, Code 20F<br>Naval Surface Weapons Center<br>Dahlgren, Virginia 22449 | 1 |
| 7. | CAPT. J.Hood, USN<br>PMS 400B5<br>Naval Sea Systems Command<br>Washington, D.C. 20362 | 1 |
| 8. | RCA AEGIS Repository<br>RCA Corporation<br>Government Systems Division<br>Mail Shop 127-327<br>Moorestown, New Jersey 08057 | 1 |
| 9. | Library (Code E33-05)<br>Naval Surface Weapons Center<br>Dahlgren, Virginia 22449 | 1 |
| 10. | Dr. M. J. Gralia<br>Applied Physics Laboratory<br>John Hopkins Road<br>Laurel, Maryland 20707 | 1 |
| 11. | Dana Small, Code 8242<br>NOSC<br>San Diego, California 92152 | 1 |
| 12. | LT D. R. Garrett<br>621 Mango Dr.<br>Virginia Beach, Virginia 23452 | 2 |

Thesis
G1985 Garrett
c.1 A software system implementation guide and system prototyping facility for the MCORTEX executive on the real time cluster.