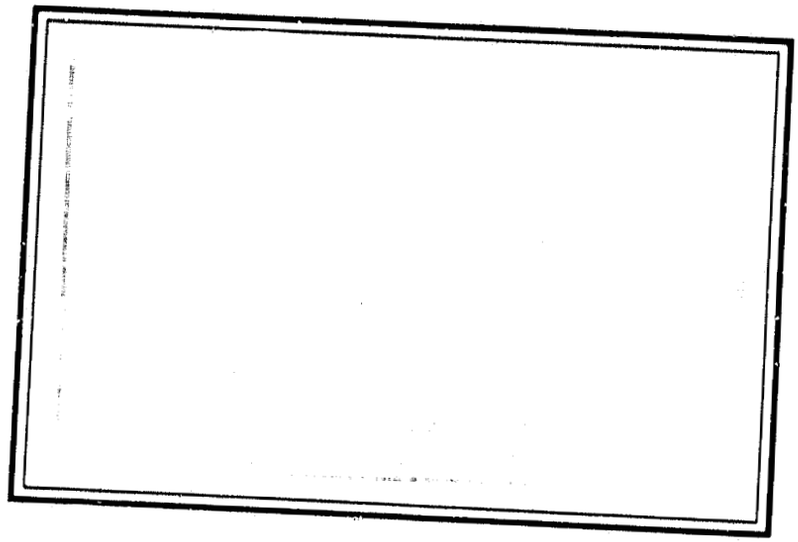


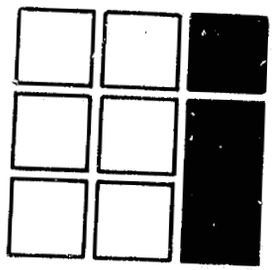
(NASA-CR-161396) NSSC-2 OPERATING SYSTEM DESIGN REQUIREMENTS SPECIFICATION Final Report (Intermetrics, Inc.) 78 p HC A05/MF A01

N80-19861

CSCL 09B

Unclas

G3/61 14915



INTERMETRICS

IR-432
NSSC-II OPERATING SYSTEM
DESIGN REQUIREMENTS
SPECIFICATION

August 15, 1979

(FINAL)

Contract: NAS8-33382

Submitted To: George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama 35812

Submitted By: Intermetrics, Incorporated
3322 S. Memorial Parkway Suite 2
Huntsville, Alabama 35801

T. T. Schansman
J. R. Bounds

PREFACE

This document presents the design requirements for an NSSC-II Operating System.

This work was performed for the Marshall Space Flight Center by Intermetrics Inc. under contract NAS8.

Questions concerning the contents of this document should be directed to T. T. Schansman, Intermetrics, Huntsville.

TABLE OF CONTENTS

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

TABLE OF CONTENTS (Con't)

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

LIST OF ACRONYMS AND ABBREVIATIONS

AEM	Application Error Monitor
DEPI	Dedicated Experiment Processor Interface
EC	Experiment Computer
ECOS	Experiment Computer Operating System
EM	Event Monitor
ESP	Error Signal Processor
GMT	Greenwich Mean Time
IDS	Interphase Data Storage
I/O	Input/Output
IOS	Input/Output Scheduler
IPL	Initial Program Load
ISR	Interval State Recorder
IT	Interval Timer
MET	Mission Elapsed Time
MSFC	Marshall Space Flight Center
NSSC-II	NASA Standard Spacecraft Computer - II
O/S	Operating System
PA	Processor Allocation
PCC	Programmable Crate Controller
PI	Phase Initialization
RTC	Real Time Clock
SC	System Clock
SI	System Initialization
SOS/II	Standard Operating System/II
TMI	Time Interface
TS	Task Scheduling
TSA	Temporary Storage Allocation
TSC	Task State Control
TSE	Test Support Equipment
RAM	Random Access Memory

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

LIST OF FIGURES

LIST OF TABLES

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

1.0 INTRODUCTION

1.1 Purpose

The purpose of this document is to define the design requirements and establish the design principles for an NSSC-II Operating System (SOS/II). The NSSC-II is, in general, designed to support a variety of space applications and is intended to provide experiment and subsystems control during flight. The defined Operating System (O/S) provides a set of software modules which perform supervisory, control, and support functions. A potential user can tailor, with a minimum of ease, the O/S by selecting and adapting those modules which serve his particular needs.

1.2 Operating System Objectives

- Support Real Time Spacecraft applications
  SOS-II is to be designed specifically for this type of application. Such applications have the following typical characteristics:
  - The Software interacts with an environment in which several things may happen simultaneously at unpredictable times.
  - The Software must respond to the environment within controllable and predictable time limits.
  - The Software performs a predictable number of concurrent functions.
  - For any particular application the system configuration is fixed.
- Provide a "flexible purpose" system
  A general purpose system is one which aims to provide all functions which a potential user may need. The disadvantage of such a system is that it is relatively

large and complex. A flexible purpose is intended to provide the most common functions which a user may need. The intent is to let the user choose those (and only those) functions which are of use to him and enable him to perform any needed adaptations with relative ease.

- Provide a modular operating system
  A corollary of a flexible purpose system is that it must be modular, by which is meant "easy to change." Modularity by itself does not guarantee ease of modification. The system must be defined with a clear view of the potential changes which may affect the system. The modularity is then designed to be, at least, adaptable to those changes.
- Operate with current support software
  The applicable NSSC-II Support Software consists of a Fortran Compiler, an HAL/S Compiler, an Assembler, and a Link Editor. The Operating System must be able to accept the output of this support software and cannot impose any modifications upon this support software.
- Support an SPSME I/O configuration
  The major characteristic of a real time system is that it interacts with its environment. Therefore a Real Time Operating System must be able to perform some level of Input/Output functions. On the other hand, I/O interfaces and devices vary considerably among applications. To insure that the O/S concepts could be proven, it was decided to select one, possibly standard, I/O configuration - the MSFC/SPSME configuration.
- Use minimum practical amount of storage
  The NSSC-II storage size is relatively limited. Therefore programs for the NSSC-II generally should

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

be designed to minimize core. This groundrule should only be broken for selected areas, which are critical to the real-time aspects of the system.

A sub-objective is that a useful O/S version can be fit into 25% of the minimum available main memory. As the minimum available is 32K bytes, it is obvious that the minimal subset must be very carefully selected.

1.3 Document Organization

This introduction is followed by Section 2 - Overview. That section contains a (fold-out) operating system diagram showing the system as a whole. General system concepts, approach, and terminology used are also introduced in that section. Section 3 - Subsystem Requirements - provides the detailed design requirements for each identified subsystem. Section 4 - Implementation Standards - is the final section. This section describes the critical groundrules and techniques which must be applied during subsequent phases to insure that the initial objectives are met.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

2.0 OVERVIEW

2.1 Methodology

The approach used in organizing these design requirements differs slightly from the commonly used approach and therefore requires some explanation.

Commonly the requirements are provided in terms of the functions to be performed and the algorithms to be used. Functions and algorithms which are somewhat related are grouped together under the same heading. This grouping is usually somewhat arbitrary and primarily serves tutorial purposes rather than reflect any sort of system organization.

As described in the objectives, "modularity" is of prime concern to the NSSC-II O/S. Therefore we have attempted to address the modularity strongly on the requirements level rather than waiting for the design phase.

The term "modularity" is usually interpreted as referring to the component structure of a system. In fact, however, the component structure does not in the first instance establish the modularity of the system.

A system is a dynamic entity, a process. It is more than an assembly of components. When we break down a system for analytical purposes we search for the processes which interact to fulfill the purpose of the system. We break down the system into "smaller systems" which are called subsystems. The "modules" of a system are therefore subsystems. Note that a subsystem is not a physical entity, it is identified by the process it performs. System modularity is thus, in the first instance, not determined by the physical component structure. System modularity

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

is determined by the types of subsystems selected and their interaction. System modularity is realized by the proper component structure. In other words, the proper component structure is driven by the selected subsystem modularity.

A subsystem (i.e. the process it represents) is executed by physical components. Especially in computer software, a subsystem may use a single component and often gets equated to the component it uses. Therefore subsystems and components are often confused.

A short example may illustrate the above flood of words. Assume that we have to design a system which outputs a series of numbers: 1,1,1,2,3,1,4.... We can decide to form this system from three subsystems as illustrated in figure 2-1. Subsystem 1 outputs a steady stream of 1's. Subsystem 2 adds each number it receives from subsystem 1 to its previous output value and outputs the new result. The outputs are combined to form the desired stream of numbers by subsystem 3.

Now assume that we want to change the system to output a slightly different set of numbers: 2,1,2,2,2,3,2,4.... It is easy to see that there is no simple way to do this without changing both subsystems irregardless of the component structure used by the subsystems. On the other hand, if we had designed the subsystems as illustrated in figure 2-2 the change would have been relatively minor.

It can be argued that the subsystems in this example could just as well have been discussed as components. This is true because of the simplicity of the example. It is not difficult to visualize more complex subsystems however, which for example share components. In such cases it becomes exceedingly confusing to discuss subsystems in terms of their components as the systems modularity is no longer apparent in the component structure.

In this requirement specification we have attempted to put the theory, described above, in practice. The goal is to discuss the system in terms of these entities, the subsystems,

which determine its organization and characteristics and thus determine its inherent modifiability. The associated functions are grouped accordingly. Note that to realize the inherent modifiability, an appropriate component structure must be selected.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

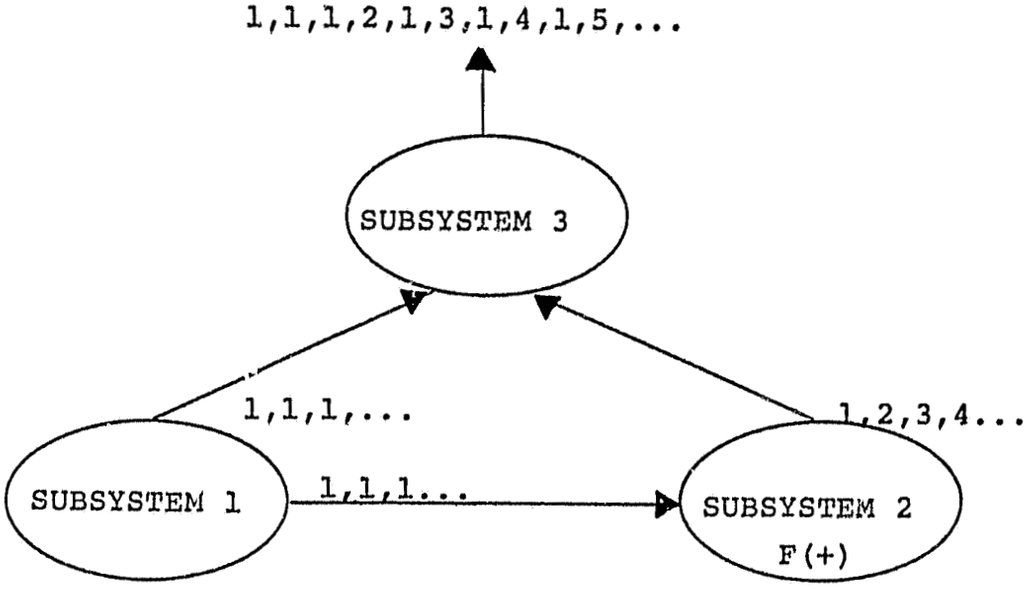


Figure 2-1: Example A System Structure

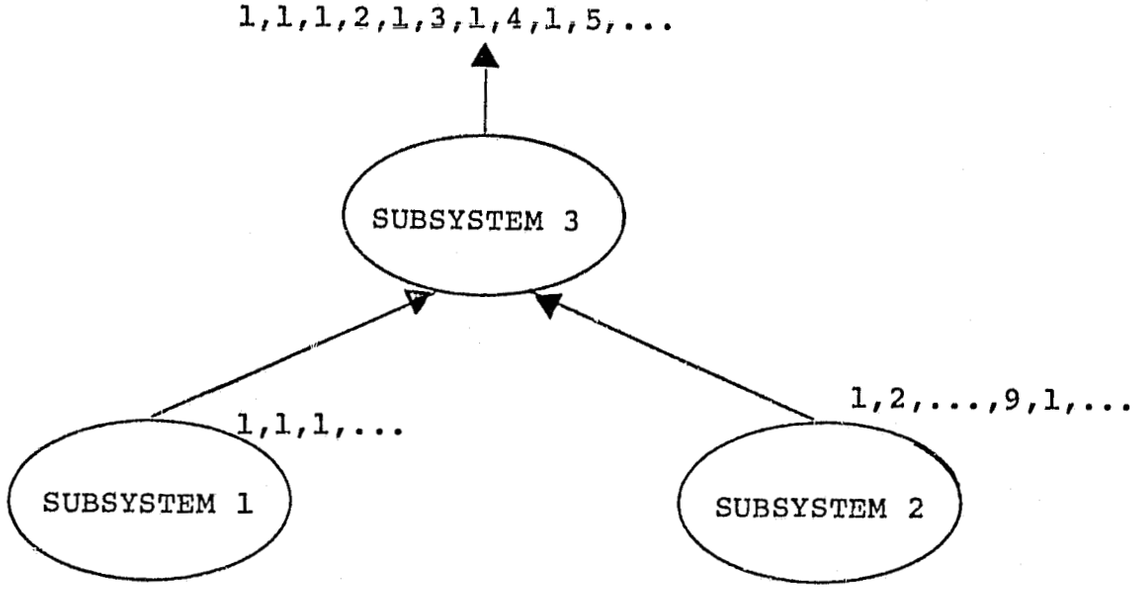


Figure 2-2: Example B System Structure

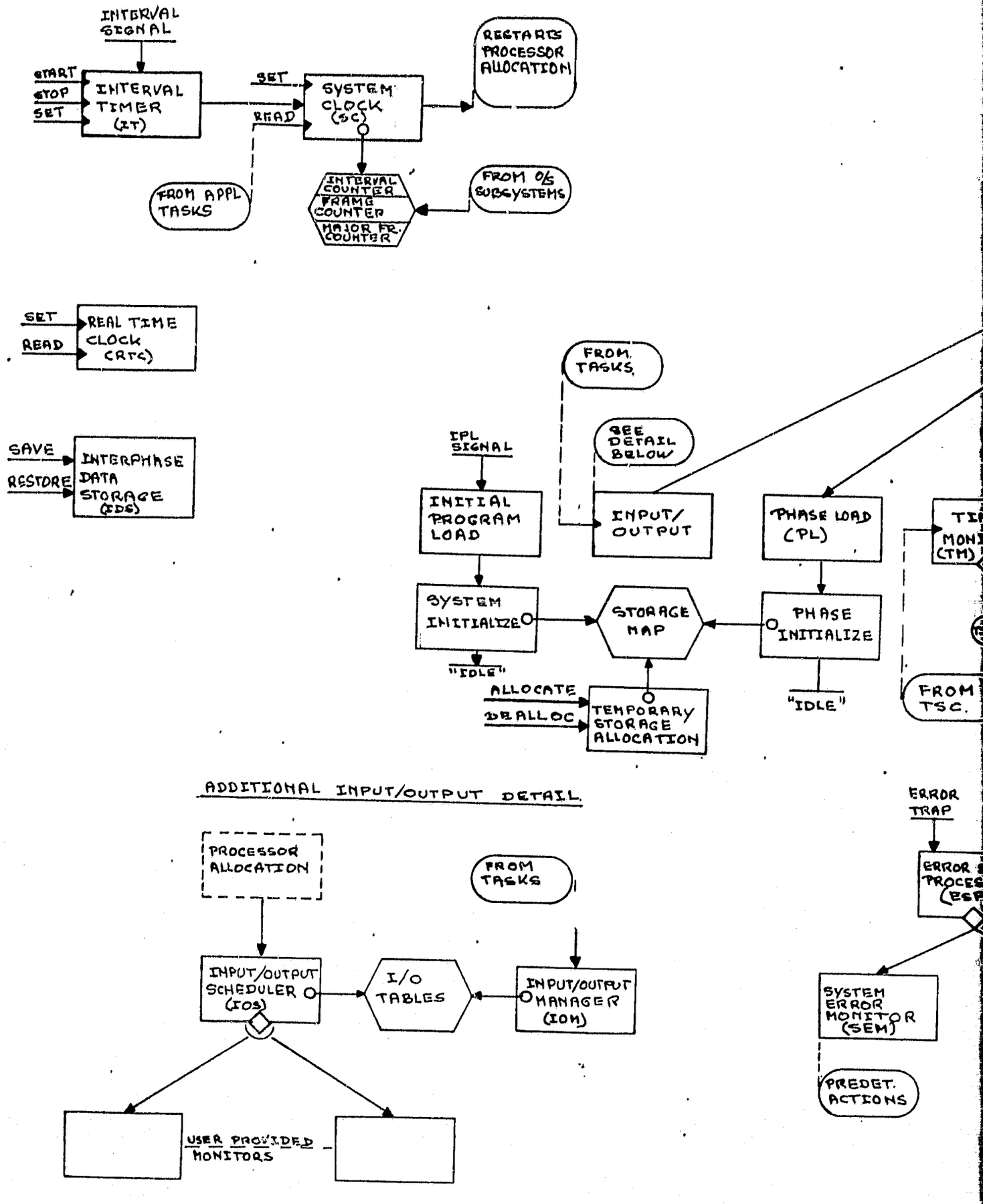
2.2 Notation

Fig. 2-3, a foldout chart, can be read very much like a flow chart. It shows the subsystems breakdown and the flow of control and interaction among subsystems, where each subsystem is represented by one or more rectangles. (Multiple rectangles in fact depict sub-subsystems). Arrows are used in the following manner:

Subsystem AAA invokes Subsystem BBB.

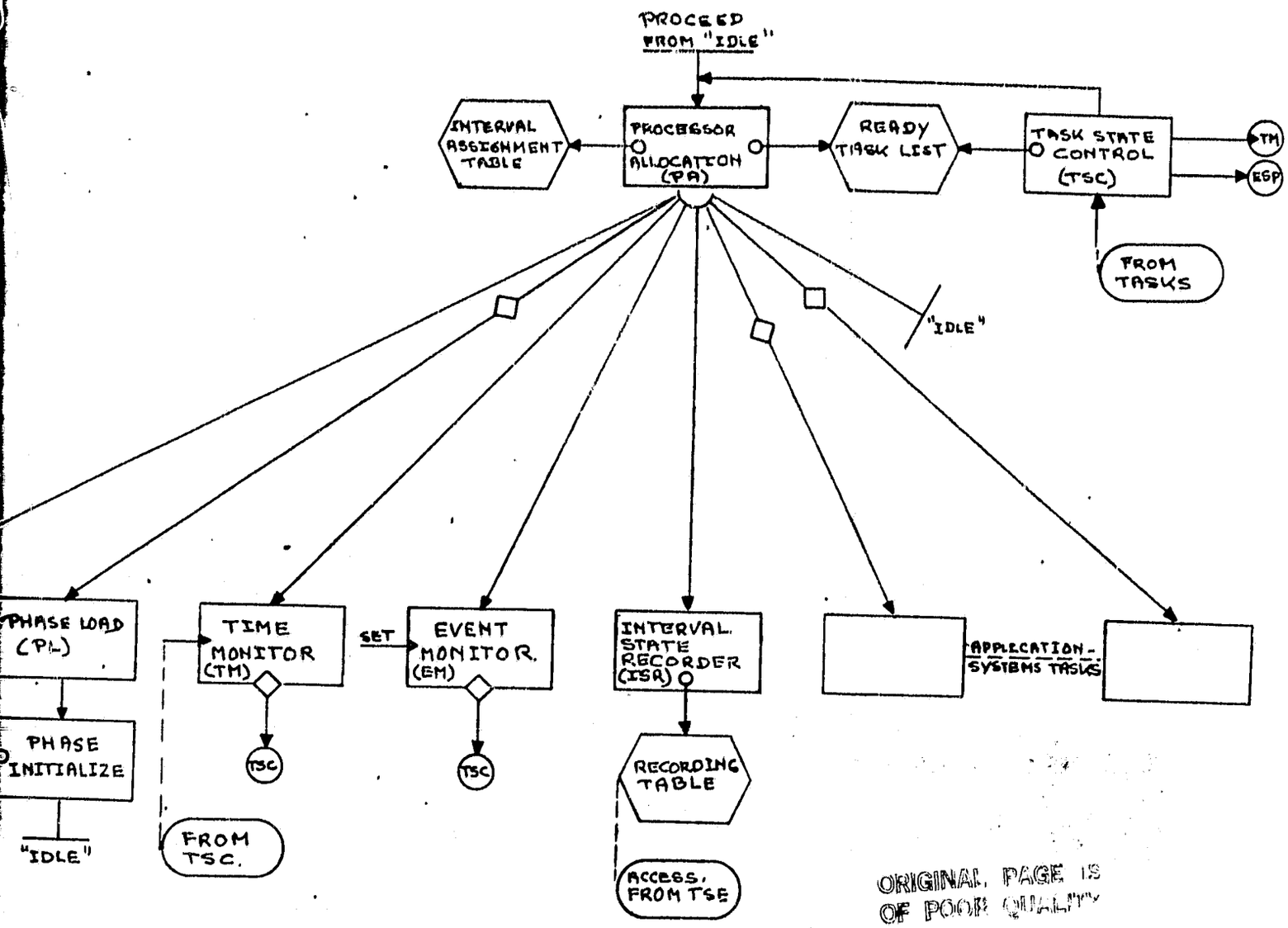
Subsystem AAA performs an operation which affects the process of BBB. It does not invoke BBB.

Subsystem AAA accesses (can be read, write, or both) a data area in common with other subsystems.

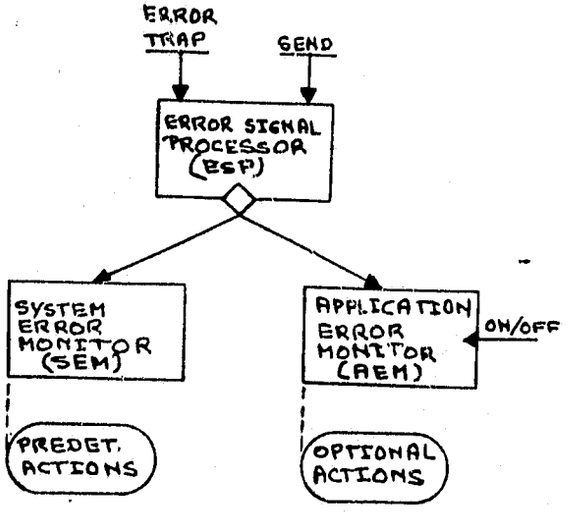


FOLDOUT FRAME

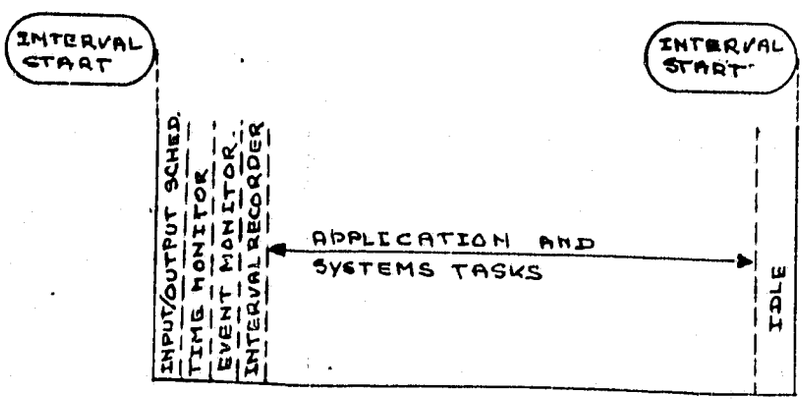
-- FIGURE 2-3: 60SII PROCESS FLO



ORIGINAL PAGE IS OF POOR QUALITY



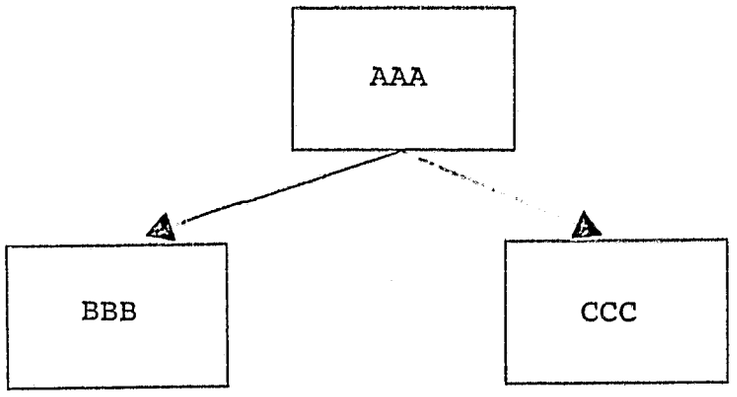
STANDARD INTERVAL TIMING SEQ



FOLDOUT FRAME 2

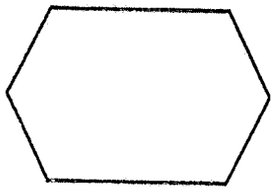
II PROCESS FLOW

Subsystem AAA invokes subsystem BBB under certain conditions.



Subsystem AAA invokes BBB and CCC respectively, sequentially in an order from left to right.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840



Data Structure accessed by more than one subsystem.

Connector.

Annotation.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.3 General Description

2.3.1 System Timing

All timing relations in the system revolve around a basic Input/Output timing interval. All Input/Output (all communication between the NSSC-II application software and the experiments or engineering subsystems) is performed on the boundaries of a fixed (user specifiable) interval.

The highest rate at which periodic Input/Output operations can be performed is determined by this fixed interval. Any application subsystem (Task) can be executed only once during such interval. Therefore the maximum rate at which cyclic tasks can be executed is also determined by this interval.

Subsequent to executing the Input/Output subsystems, any subsystems which may affect Task activation or deactivation are performed. Consequently, all application tasks to be activated during an interval are known before the system begins executing the application Tasks for that interval.

This sequence of events is graphically depicted in Figure 2-4. Note that the intent is to have all active Tasks completed prior to the start of the following interval. This scheme prohibits the use of demand interrupts and/or Task switching interrupts (some exceptions are noted later). There is in fact no need for such interrupts. All potential "demands" are interrogated at the start of the interval and Task activation priorities are established. The Task execution conditions therefore remain stable over an interval.

The disadvantage of this scheme is that Tasks with an execution time which exceeds an interval need to be chopped up in pieces which fit within an interval. Assuming that the NSSC-II can execute 200K instructions/second (Standard Fixed Point Instruction Mix) and that 15% idle time is allowed for within an interval, then a total of 1700 instructions can be

executed per interval.

Note that this approach is theoretically less efficient than a "free running" system. However the pay off in testability and simplicity of design can be significant.

2.3.2 General Process Flow

The O/S subsystems and their process interrelations are shown in Figure 2-3.

Each of the subsystems shown is discussed in detail in the following sections.

The system is driven by the System Clock, which in turn is activated by an Interval Timer set to the minimum desired interval. The System Clock maintains time in terms of number of intervals elapsed and can be interrogated by any subsystem. The System Clock further assures that Processor Allocation is activated each interval. (The system can therefore only "hang up" under highly exceptional conditions). All other subsystems run under control of Processor Allocation with the exception of the Real Time Clock Subsystem.

The subsystems which are directly activated by Processor Allocation are called Tasks. We distinguish between O/S Tasks (subsystems provided as part of the Operating System), Application Tasks (generally application dependent subsystems), and Systems Tasks. The latter are subsystems which are not an integral part of the control and supervisory functions of the Operating System, but nevertheless provide support for a class of applications.

Any Task execution is under control of the user. The user can deactivate any O/S Task, change the execution sequence, etc. This is true because Processor Allocation works from a "Ready Task List" and makes no distinction between the types of Tasks.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

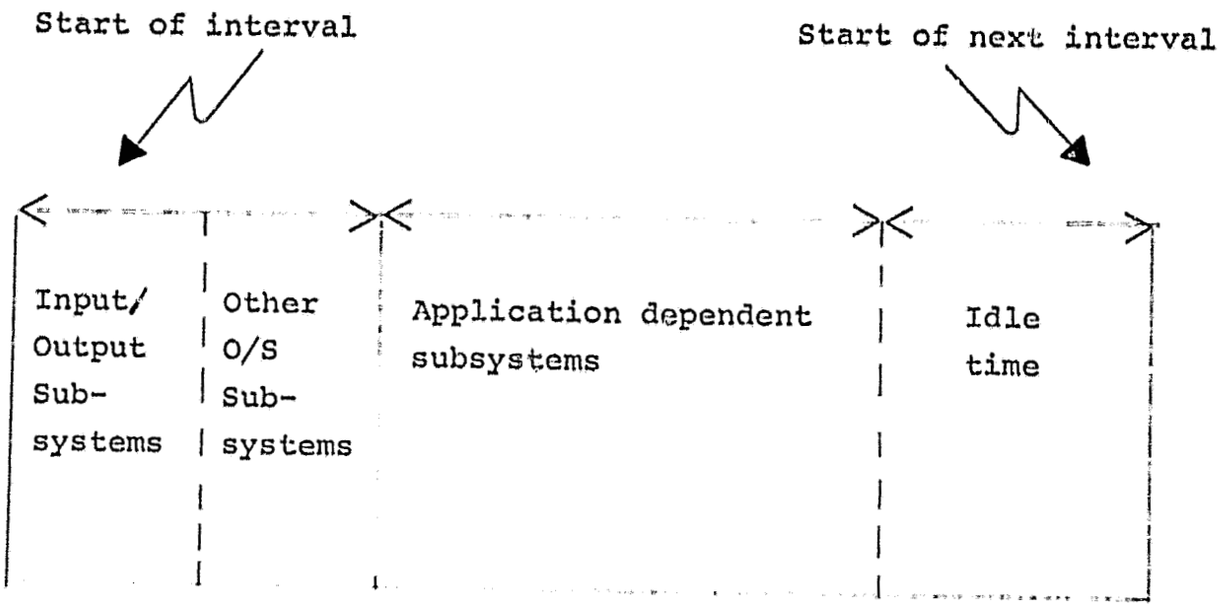


Figure 2-4: Basic Timing Interval

2.3.3 Application Program Organization

An Application Program Complex is a set of related Tasks as shown in Figure 2-5. A Task is pragmatically defined as any program module which has been identified as such to SOS-II. Communication between the Application Programs and SOS-II is strictly through SVC's.

In addition to the Task an Application Program Complex may contain common data areas and common subroutines (e.g. the run-time library). No linking is performed during the loading process. Therefore all related Tasks and their common environment must be contained within the same load module. When different or additional Tasks are required a complete reload of the system must occur.

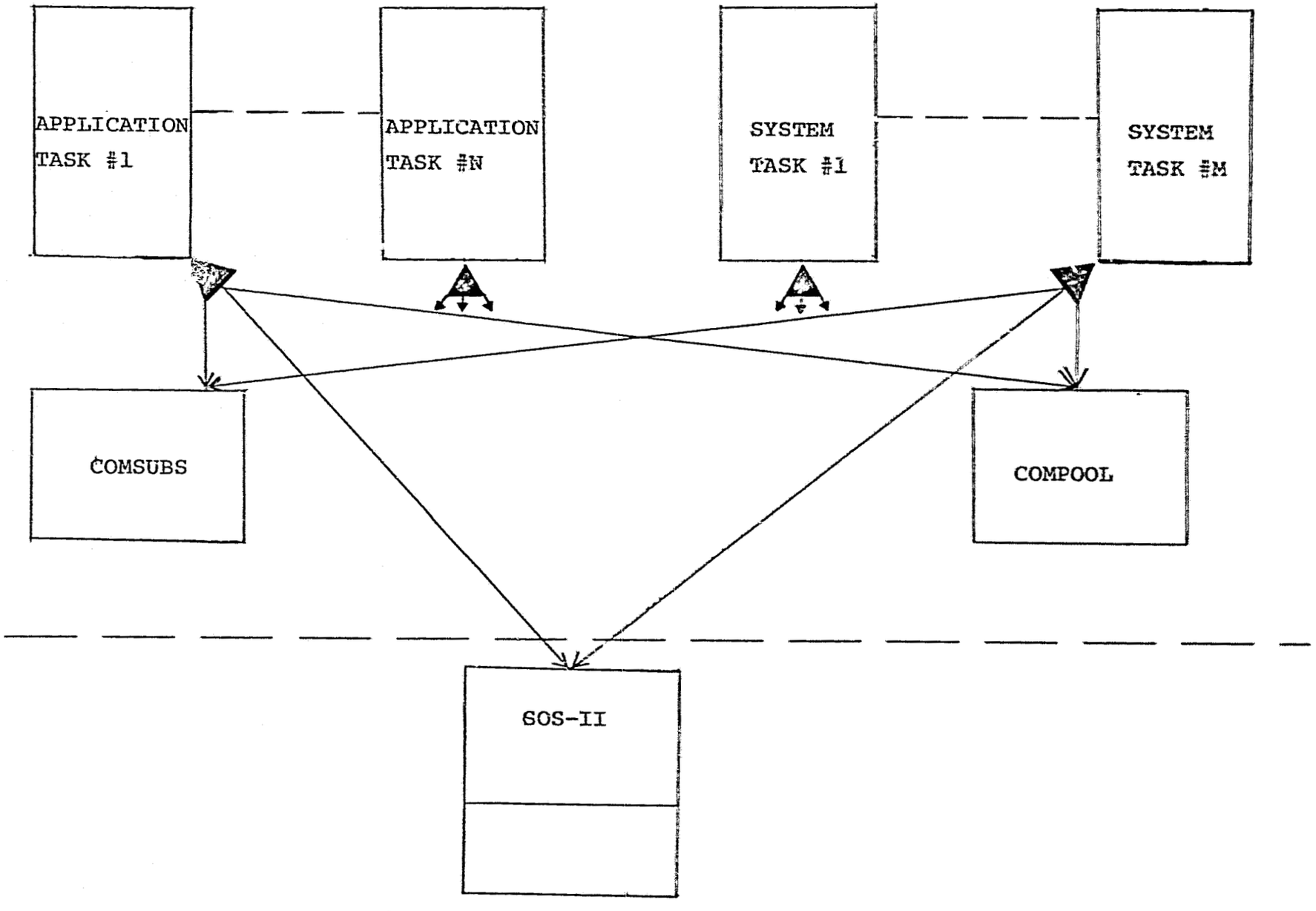


Figure 2-5: Program Organization

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.0 SUBSYSTEM REQUIREMENTS

This section contains a subsection for each of the NSSC-II O/S Subsystems. Each such subsection follows the standard format described below.

Paragraph 1. Subsystem Diagram - This is simply that portion of the overview diagram (provided in section 2, Figure 2-3) which is applicable to this subsystem. In some cases it is slightly expanded and/or contains additional annotation.

Paragraph 2. Subsystem Activation - Describes what activates this subsystem as well as the activation conditions.

Paragraph 3. Activated Subsystems - Other subsystems which are to be activated by this subsystem and their activation conditions, if any, are summarized here.

Paragraph 4. Functional Description - This is the key paragraph. It describes the purpose of the subsystems, its functions, its logical relationship to other subsystems, and specific algorithms to be used in the design.

Paragraph 5. Parameters - These are certain values which are key to the purpose or function of the subsystems, and are changeable by a potential user. The software must be implemented such that a parameter identified under this heading can be changed by changing one, clearly identified, location in a program.

Paragraph 6. Task Commands - Subsystems are controlled by commands executed during a Task process. Those Task commands, and their effect on any subsystem, including itself are described in this paragraph.

Paragraph 7. Initialization - The actions which are required to be performed during system initialization to assure that this subsystem will have the appropriate initial conditions are described here.

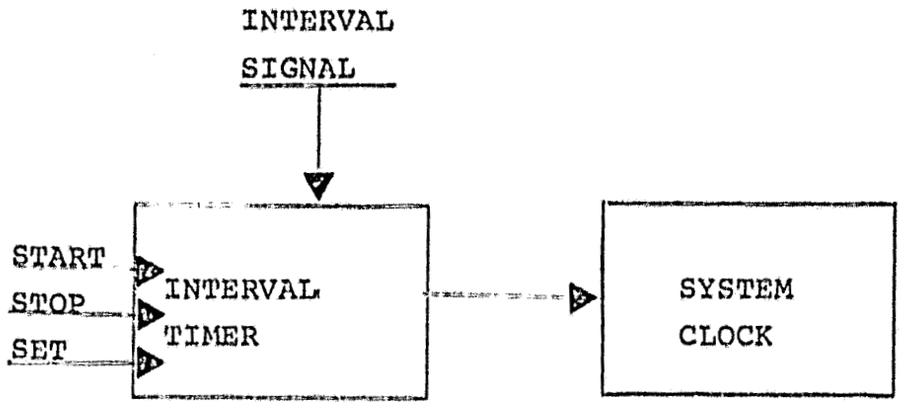
Paragraph 8 Potential Modifications - To insure that the component structure is designed such that the subsystem is easily changed, the specific modifications which must be accommodated are listed here.

Paragraph 9. Implementation Notes - This paragraph first of all describes a minimum useful version of this subsystem. It lists the functions which may not be useful to all users. Other implementation related items critical to the desired subsystem process are also noted in this paragraph.

Note: The term Task Process is used, in this document, interchangebly with executing Task.

3.1 INTERVAL TIMER (IT)

3.1.1 Subsystem Diagram



3.1.2 Subsystem Activation

The IT is activated by an interrupt signal at the end of each interval. This signal can come from the NSSC-II Interval Timer or another external source.

3.1.3 Activated Subsystems

The IT in turn activates the System Clock.

3.1.4 Functional Description

The It is the basic driving mechanism for the system. If the IT is deactivated, the system comes to a soft stop, waiting for an external signal. On the other hand, the IT serves (indirectly) as a watch dog timer. The interval signal is never to be inhibited. The system will therefore,

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

under most conditions, be interrupted out of a system hang-up. As described later, Processor Allocation (Section 3.4) has the logical means to detect whether a system hang-up occurres.

The IT can be started/stopped from any task, even though it normally will only be affected during System or Phase Initialization (Sections 3.11, 3.12).

The interval is dynamically changeable by any task. When running, the interval change takes effect at completion of the current interval.

When activated the IT sets up and starts the NSSC-II Interval Timer prior to transferring control to the System Clock.

3.1.5 Parameters

None.

3.1.6 Task Commands

1. SET INTERVAL - Establishes or changes the interval size.
2. START IT - This initially activates the IT.
3. STOP IT - This, in essence, stops the NSSC-II Interval Timer.

3.1.7 Initialization

During System Initialization the interval must be set. Initialization ends by starting the IT.

3.1.8 Potential Modifications

As described here the IT subsystem uses the NSSC-II Interval Timer. This may be replaced by some other external timer, which may or may not be controllable.

3.1.9 Implementation Notes

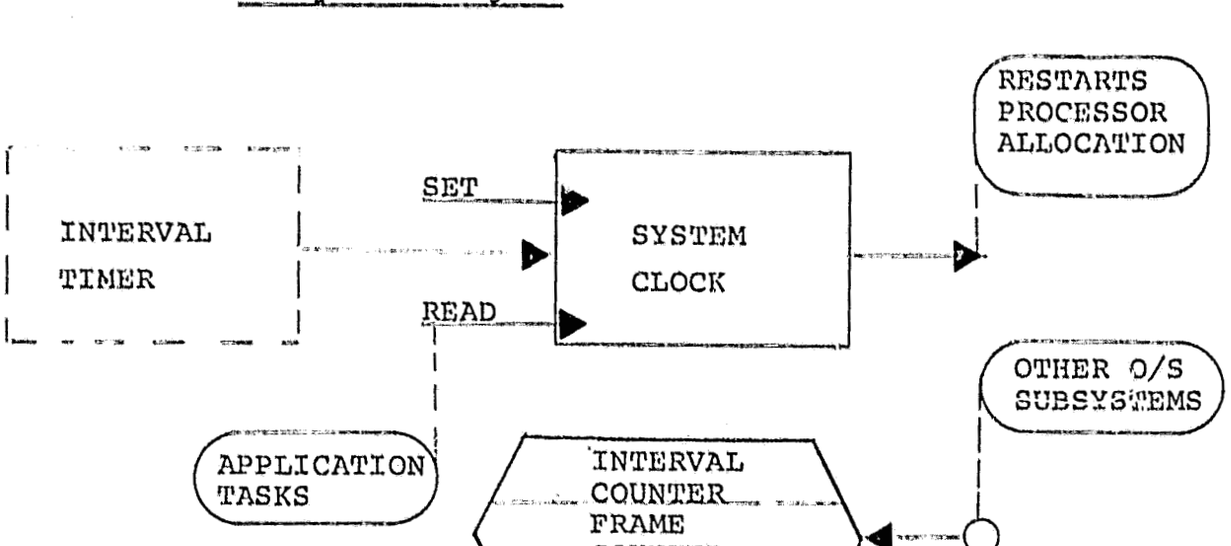
Current implementation plans call for an SPSME I/O Configuration. In this case most of the I/O functions as

well as the interval timer are contained in the SPSME. Thus the interval signal will come from the SPSME and will be used to directly activate the System Clock. The system will therefore be implemented to run either the NSSC-II Interval Timer or a SPSME interval signal.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.2 SYSTEM CLOCK (SC)

3.2.1 Subsystem Diagram



3.2.2 Subsystem Activation

The SC is activated at the start of each interval by the Interval Timer subsystem.

3.2.3 Activated Subsystems

The SC indirectly activates Processor Allocation by allowing the system to proceed from the state it was in when the interval interrupt occurred.

3.2.4 Functional Description

The SC keeps track of the number of intervals elapsed (since an application determined start time). It does this by incrementing up to three counters each time the SC is activated: an interval counter; a frame counter; a major frame counter.

The Interval Counter is incremented by one (1) each interval. When a specified maximum count is reached, the Interval Counter is reset to zero and the Frame Counter is incremented by one. When the Frame Counter reaches a specified maximum count, it in turn is reset to zero and the Major Frame Counter is incremented by one, etc.

All counters are directly accessible by other O/S subsystems. Application tasks, however, are expected to access the counters through an explicit command to the SC.

The counters can be set to any desired value at any time through the appropriate command.

3.2.5 Parameters

The maximum counts for each counter are "identified constants" within the programs.

The initial values of the counters can also be set by setting constants in the code.

3.2.6 Task Commands

1. SET CLOCK - This command allows any of the counters to be set to any desired value.
2. READ CLOCK - This command allows a Task to access the values retained in the counters with one command. Either two or three values are returned depending on whether the Major Frame Counter is in use.

3.2.7 Initialization

System Initialization must set the desired (application dependent) initial counter values.

3.2.8 Potential Modifications

It may be desirable in some cases to retain an accurate synchronization between elapsed real time (GMT or MET) and the interval counters. The simplest way to do this is to drive the Real Time Clock from the Interval Timer.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.2.9 Implementation Notes

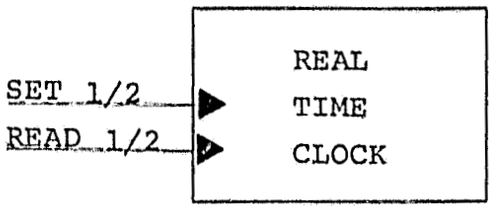
A minimum useful version contains an interval counter which resets after a preset maximum count.

The implementation version will be driven from an SPSME signal. All three counters will be included.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.3 REAL TIME CLOCK (RTC)

3.3.1 Subsystem Diagram



3.3.2 Subsystem Activation

The Real Time Clock is started when a SET command is received and continues indefinitely.

3.3.3 Activated Subsystems

None.

3.3.4 Functional Description

This subsystem contains two clocks (1 and 2) which can be used to maintain two different time bases (e.g., MET and GMT). The time bases are selectable by the user and can be set any time. Either one of the clocks can be accessed any time by any other subsystem.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.3.5 Parameters

None.

3.3.6 Task Commands

1. SET (1 or 2) - Initializes the designated clock to the desired value.
2. READ (1 or 2) - The contents of the clock are provided to the requesting process.

3.3.7 Initialization

System Initialization should be able to SET either one or both clocks as requested by the user.

3.3.8 Potential Modifications

It may be desirable that either one or both clocks be synchronized with a similar clock external to the NSSC-II. This would require a clock to be SET to the external source system during initialization and to be periodically corrected. This subsystem may also be superfluous in configuration with an external source. In such cases this subsystem is nothing more than a device access routine.

It may also be desirable to drive the Read Time Clock from the Interval Timer to insure that these stay internally synchronized.

3.3.9 Implementation Notes

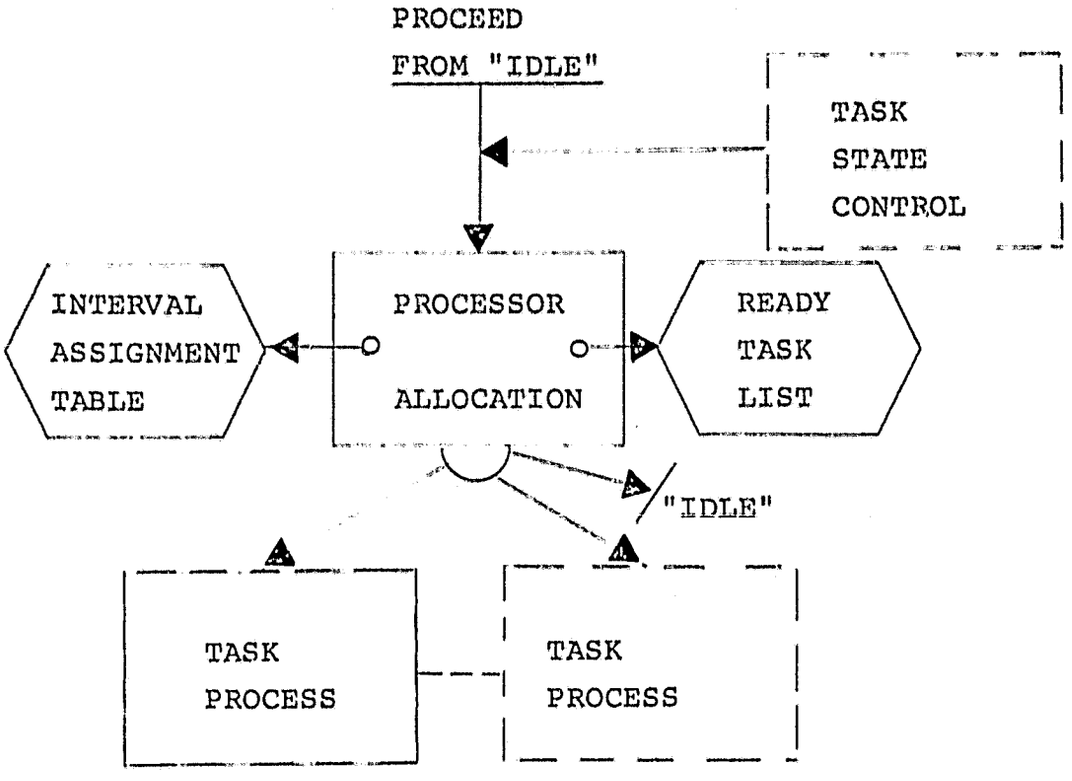
The implementation version will use the NSSC-II Real Time Clock. The time format used will be identical to Experiment Computer Operating System (ECOS).

Note that the configuration is anticipated to contain an SPSME Interface (TMI) Module which also can be accessed to obtain GMT.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.4 PROCESSOR ALLOCATION (PA)

3.4.1 Subsystem Diagram



3.4.2 Subsystem Activation

Processor Allocation is activated each interval by the System Clock (also see Initialization). PA is also activated (through Task State Control) whenever a Task releases the processor.

3.4.3 Activated Subsystem

PA activates Operating System Tasks, System Tasks, and Application Tasks in accordance with the information in the Ready Task List and Interval Assignment Table.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.4.4 Functional Description

The purpose of Processor Allocation is to activate the tasks, which are "ready", during their assigned intervals in order of their assigned priorities.

3.4.4.1 Processor Allocation Method. The Ready Task List always contains the addresses of the Tasks which are ready for execution. The Interval Assignment Table describes which Tasks may be executed during a particular interval. The Interval Assignment Table is the means whereby the user distributes the potential execution of his tasks over the intervals such that they can be executed in a timely manner without exceeding the amount of time available in each interval.

Each Task can be assigned a simple priority number which determines when it will be executed relative to other Tasks executable within an interval. Processor Allocation does not perform any "Task switching". Each Task is allowed to complete its execution even if a higher priority task becomes ready for execution. Basically the priority only determines the sequence of Task executions during an interval.

PA thus determines from the Interval Assignment Table which Tasks may be executed, determines from the Ready Task List which of those Tasks must be executed, and uses the priority numbers to determine the sequence in which the Tasks are to be executed.

3.4.4.2 Interval Overflow. The interval timer also serves as a watch dog timer. When PA is activated by the System Clock (each interval), it will check whether at the time of the interval interrupt if it was in the appropriate idle condition. If it was not, Task execution exceeded the interval time and a system error is signalled.

To allow the amount of mandatory idle time to be minimized, two exceptions are allowed: an "occasional" overflow; and background processing.

3.4.4.3 Occasional Overflow. The total execution time of a set of tasks to be executed during an interval must be sized such that a certain amount of idle time remains during an interval. The total execution time may vary within certain limits, thus the worst case condition (even if it is exceptional) must be accounted for. This may result in an excessive amount of idle time during normal conditions.

To allow such idle time to be minimized the PA will be designed to accomodate an "occasional" overflow. When PA detects an interval time overflow condition, it will interrogate the Task control information to determine if this was an "interruptable" Task. If it is not, it will signal a system error. If it is, it will allow resumption of the interrupted (by the interval signal) task before executing any other task.

3.4.4.4 Background Processing. Certain applications may have useful, but not time critical Tasks to perform in conjunction with their real time control tasks (foreground tasks). The system efficiency can be improved by allowing such Tasks to be executed during the interval idle time (background tasks). This implies that switching from "background to foreground" Tasks must be allowed.

To accomodate this possibility, a two level priority scheme is defined. In general, the tasks are assigned simple priority numbers and the ready tasks with the smallest priority number are put into execution when the processor becomes available. The two-level priority scheme is used to be able to identify tasks which may be interrupted. For example, we can establish a range of 1 - 10 for the foreground tasks. If a task with a priority number of 12 is interrupted by the interval signal, Processor Allocation will recognize that this was not a cycle overflow. In addition, all foreground tasks which are ready will be executed before Task 12 is allowed to resume execution.

The background Tasks may not directly activate any of the O/S subsystems, but they may set Events. They may also communicate with real time tasks through a common data area. Sharing of common data is totally under control of the application program.

3.4.5 Parameters

The maximum number of consecutive cycle overflows allowed and the foreground/background priority number boundary must be modifiable by the user.

3.4.6 Commands

None.

3.4.7 Initialization

PA works directly from the Ready Task List and the Interval Assignment Table. System initialization must insure that the Ready Task List is initialized appropriately to the application.

3.4.8 Potential Modifications

Processor Allocation is one of the subsystems which establishes the most basic system concepts. No modifications can be made to the basic operation without affecting other subsystems. Modifications may be desired to omit unnecessary features associated with cycle overflow and background processing. The suggested incremental capabilities are as noted in the following paragraph.

3.4.9 Implementation Notes

PA should be implemented such that the following incremental capabilities can be selected:

- Foreground Processing, no interval overflow;
- Foreground/Background Processing, no interval overflow;
- Foreground Processing, interval overflow; or
- Full capabilities.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.5 TASK SCHEDULING (TS)

Task Scheduling contains three subsystems (more precisely subsubsystems): Task State Control, Time Monitor, and Event Monitor (Figure 3-1). Together they control the state transitions (Figures 3-2 and 3-3) for the Tasks. A Task is pragmatically defined as any set of software modules which are activated through Processor Allocation.

Task State transitions occur when Task Scheduling commands (summarized in Table 3-1) are executed by a Task process or when certain conditions internal to O/S are detected. Task Scheduling commands are indicated by capital letters in Figures 3-2 and 3-3. Detected O/S conditions are indicated by lower case letters.

Two READY states and two WAIT states are indicated on the figures. READY 1 and WAIT 1 signify tasks which are ready or waiting for initial task execution. READY 2 and WAIT 2 indicate tasks ready or waiting to continue execution. Under normal conditions these distinctions have no significance, but they are important when the tasks are abnormally terminated as shown in Figure 3-3.

Task Scheduling recognizes the existence of cyclic tasks. These are tasks for which the initial WAIT conditions are restored each time such a task exits normally. The cyclic attribute of a task can be removed by execution of a CANCEL command. This command does not affect the current state of the task, it only affects subsequent state transitions.

Task Scheduling supports a Forced End Concept. The purpose of the Forced End Concept is to invoke a "clean up" or "safing" procedure when a real time task is forced to terminate prior to taking a normal exit. Such a Forced End procedure can be used to put the instruments/subsystems which the task was controlling, into a harmless state.

As shown in Figure 3-3, a TERMINATE is used to designate a forced abnormal ending of a process. If the

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

process is in the Ready 1 or Wait 1 state, it has never been in execution and a clean-up procedure should not be necessary. Therefore, such a process is simply put in the inactive state. If the TERMINATE'd task is in any other state, the execution (or continuation of execution) is forced to continue from the Error Signal Processor. The error code to be transmitted to the Error Signal Processor is accessed from the control information retained for each task.

Task State Control and the Event Monitor are discussed in more detail in the subsequent functions. The Time Monitor, which has a very simple function, is discussed under Task State.Control.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

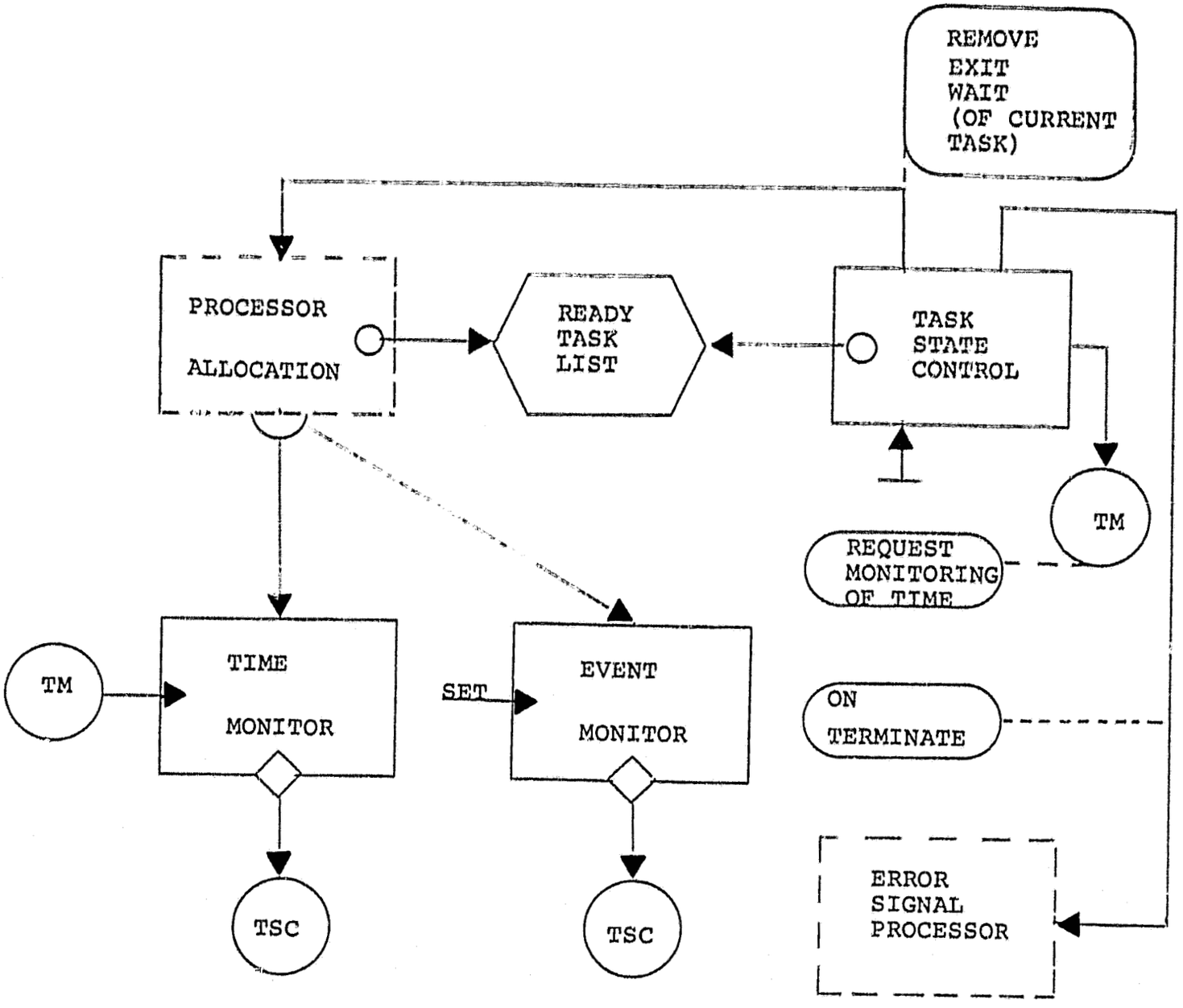


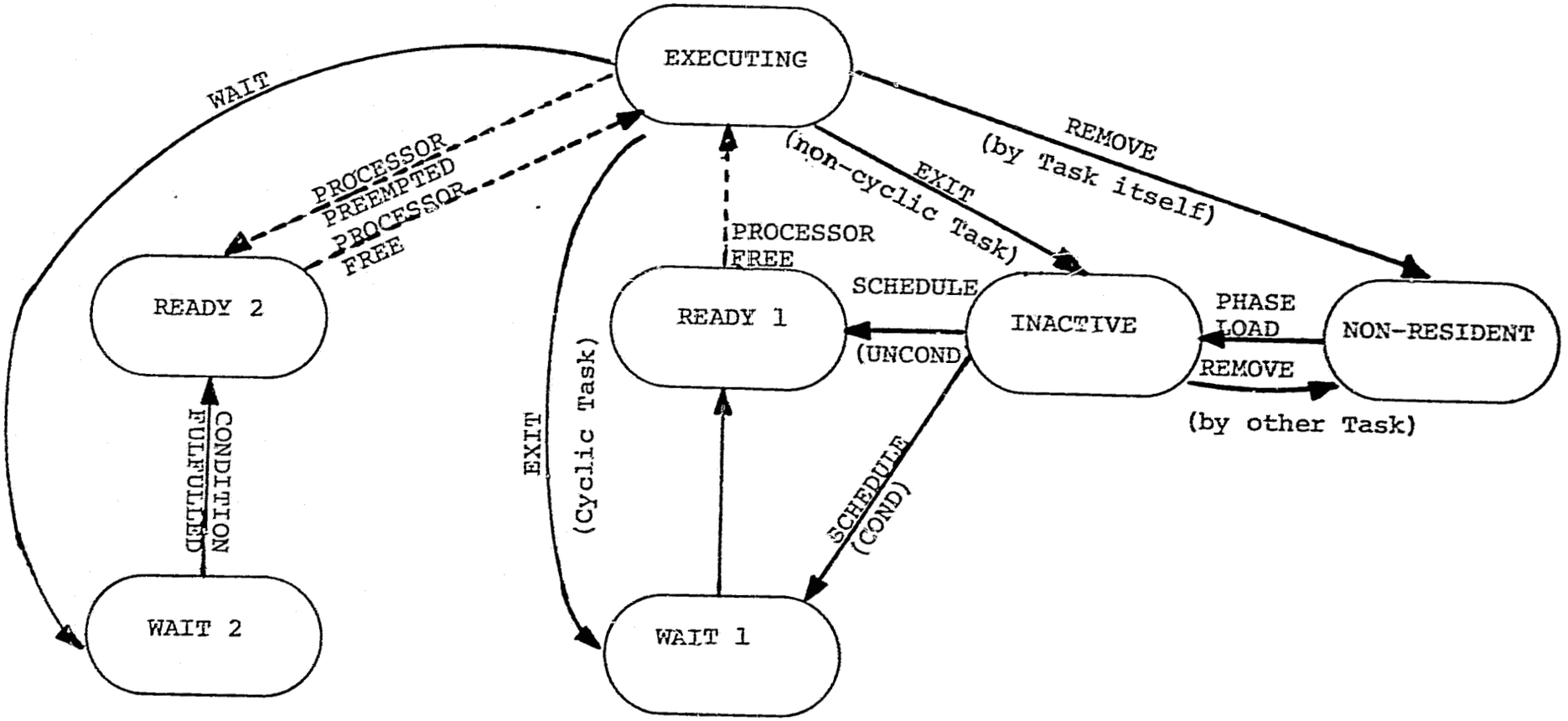
Figure 3-1: Task Scheduling Subsystems

TYPICAL MNEMONIC	BY TASK ITSELF	BY OTHER TASK	PARAMETERS
1. LOAD	------PHASE LOAD ONLY------		PHASE ID, LOAD SOURCE
2. REMOVE	YES	YES	TASK ID
3. SCHEDULE	NO	YES	TASK ID, CONDITION (TIME, EVENT) CYCLIC/NON-CYCLIC , PRIORITY
4. EXIT	YES	NO	NONE
5. WAIT	YES	NO	CONDITION (TIME, EVENT)
6. TERMINATE	YES	YES	TASK ID
7. CANCEL	YES	YES	TASK ID
8. CHANGE PRIORITY*	YES	YES	TASK ID, PRIORITY

* DOES NOT AFFECT STATE

TABLE 3-1: TASK SCHEDULING COMMANDS

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840



-----Transition not under Task State Control

Figure 3-2: TASK STATE DIAGRAM
(NORMAL CONDITIONS)

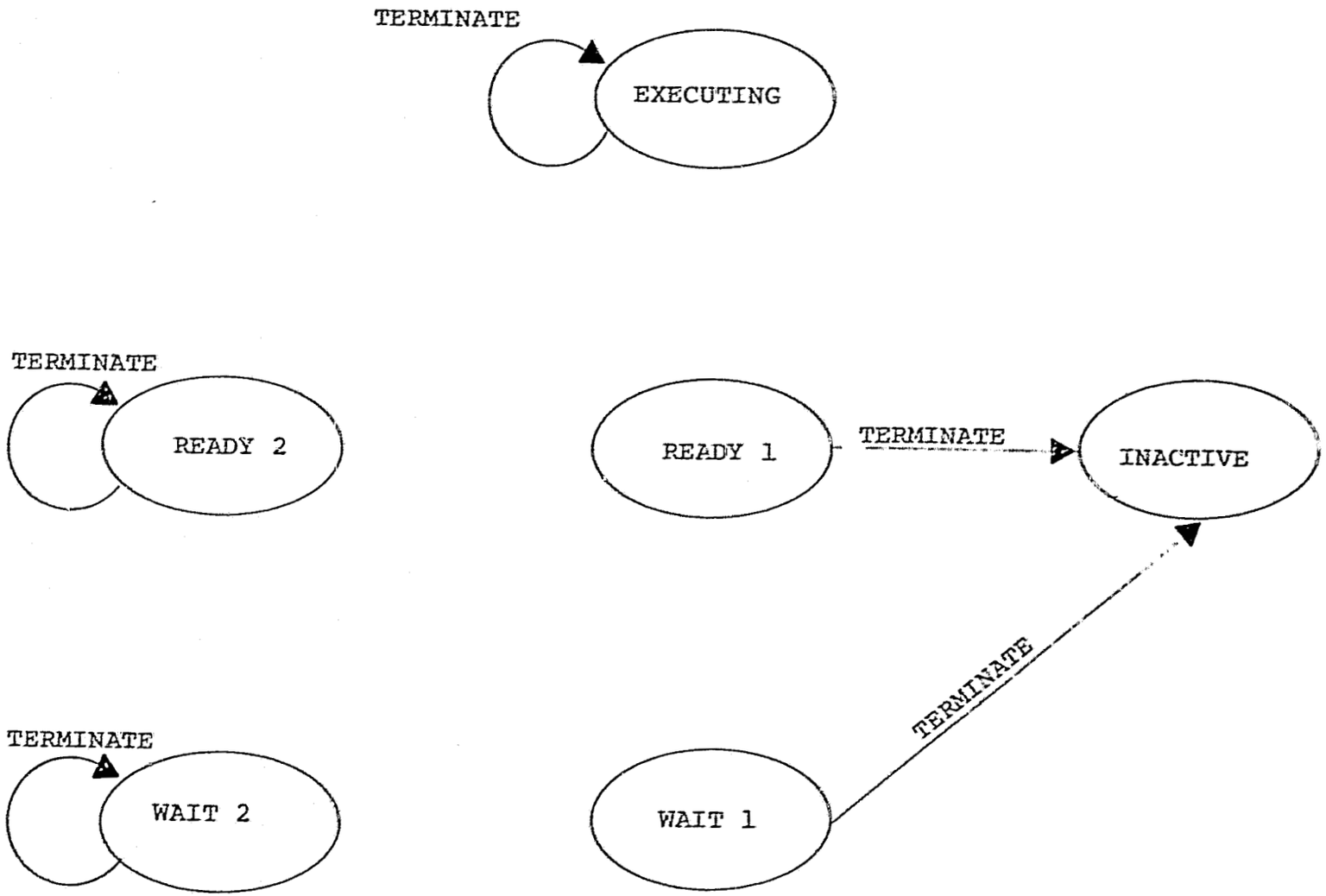


Figure 3-3: TASK STATE DIAGRAM
(ANOMALOUS CONDITIONS)

3.5.1 TASK STATE CONTROL (TSC)

3.5.1.1 Subsystem Diagram. Reference Figure 3-1.

3.5.1.2 Subsystem Activation. TSC is activated when a process executes one of the Task scheduling commands listed in Table 3-1. It is also activated by the Time Monitor and the Event Monitor when either one detects a condition which affects the state of a Task.

3.5.1.3 Subsystems Activated. TSC activates Processor Allocation when an executing task signals that it releases the processor (REMOVE, EXIT, or WAIT).

The Time Monitor is activated when TSC needs the Time Monitor to "wake up" TSC at a specific time.

The Error Signal Processor is activated after a TERMINATE command has activated Task State Control.

3.5.1.4 Functional Description. TSC is responsible for maintaining the appropriate states of the Task from the inactive state through the ready state. All Task Scheduling commands (see Table 3-1) are handled in the first place by TSC. TSC insures that a Task appears only once in the ready state. Attempts to ready a Task which is already in the ready state result in a system error. However no such tests are performed for any of the other states.

TSC handles "wait for time" conditions in conjunction with the Time Monitor. It insures that the Time Monitor is always aware of the next wait time which conditions the state of a Task.

TSC handles "wait for event" conditions in conjunction with the Event Monitor. The Event Monitor provides TSC, at the start of an interval, with a list of events set during the previous interval.

TSC activates the Error Signal Processor when a currently executing task signals a TERMINATE. The Error Signal Processor, when activated, runs as a subprocess of the executing Task process. If the currently executing Task process signals termination for another task, TSC assures that execution will, eventually, resume at the Error Signal Processor.

3.5.1.5 Parameters. None.

3.5.1.6 Task Commands. The Task commands affecting TSC are summarized in Table 3-1, Task Scheduling Commands.

3.5.1.7 Initialization. System Initialization must initialize the states of all Tasks loaded into the system to the appropriate state. The desired initial states of the Tasks are somewhat application dependent, but at least one task must initially be set to ready, or be waiting for a time condition.

3.5.1.8 Potential Modifications. For specific applications several possible simplifications (resulting in execution time and core savings) can be visualized.

First of all, the Forced End Concept, may not be used or may be simplified. The Forced End logic is rather cumbersome. Any application should carefully review the real need for the Forced End Concept. A simplification can be to make the state transitions from Ready 2/Wait 2 identical to that for the Ready 1/Wait 1 states.

Furthermore, the Task Scheduling commands may not all be necessary or specific commands may not require all possible parameters. For example, the REMOVE command may be superfluous or even the "time wait" condition may be unnecessary.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.5.1.9 Implementation Notes. TSC is one of the most commonly used O/S services. It is therefore important to optimize the appropriate components for time. As it is also one of the most core consuming subsystems, some difficult trade-offs must be anticipated during the implementation.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.5.2 EVENT MONITOR(EM)

3.5.2.1 Subsystem Diagram. See Figure 3-1.

3.5.2.2 Subsystem Activation. EM activates Task State Control each interval to signal which events are set.

3.5.2.4 Functional Description. Events are used in a SCHEDULE or WAIT command to indicate an activation condition. Events can be set from any Task. The setting of Events is totally under user control. There are currently no system events defined.

Events stay set until an action results. That is until they have been used as an execution condition. The Event Monitor signals Task State Control which Events are set. If a Task process is waiting for any of these Events, the process will be released and the event will be reset. If no Task process was waiting the Event will remain set.

3.5.2.5 Parameters. The maximum number of events used must be changeable by the user.

3.5.2.6 Task Commands. SET - is used to turn on the desired Event.

3.5.2.7 Initialization. System initialization must insure that all Events are reset.

3.5.2.8 Potential Modifications. This is a subsystem that could be subject to expansion. The additional capabilities described in the HAL/S Language specification may be desirable for certain applications.

3.5.2.9 Implementation Notes. This subsystem does not strictly have to be handled as a Task. The reason for signalling all Events set to Task State Control at one specific time is that it may be more efficient than to invoke TSC each time an Event is set. Also, it may enhance the testability of the system. This approach may be changed during the design phase if that turns out to be necessary.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.6 INPUT/OUTPUT

A basic knowledge of the SPSME modules, especially the SPSME PCC is necessary to understand these I/O subsystem descriptions.

The Input/Output subsystem is one of the largest subsystems. The requirements of the Input/Output are based on an SPSME I/O configuration. The basic design approach as reflected in Figure 3-4 is applicable to any I/O configuration. However, to be able to discuss the functions performed by the subsystem as well as to describe the commands invoking the functions a specific type of I/O configuration must be selected. For the NSSC-II Operating System the SPSME configuration was designated.

The purpose of the Input/Output subsystem is to:

1. Simplify time related execution of Input/Output,
2. Simplify coordination between Input/Output and related tasks,
3. Relieve the user of coding DEPI communication sequences, and
4. Relieve the user of coding NSSC-II I/O sequences.

The Input/Output subsystem does not perform any type of data conversion. The user must be fully cognizant of the contents of the PCC and the associated data formats.

When (eventually) standard equipment, such as a particular type of mass storage or a particular type of display system, is selected it will be possible to add "device managers" to provide the user with a "higher level" of support. It is also possible to provide more "symbolic" means for the application programs to select the appropriate PCC sequences. However these are not now part of the requirements.

The Input/Output subsystem contains two major sub-subsystems: The Input/Output Scheduler and the Input/Output Manager. The Input/Output Scheduler is invoked on interval boundaries, and performs those operations which

result in information transfer between some device and the SPSME/NSSC-II. The I/O Manager interfaces with the application programs. It transforms I/O request into appropriate commands to be executed by the I/O scheduler (at the interval boundaries) or to be executed immediately. These two subsystems are discussed in more detail in the following sections.

There are two major types of commands: DELAYED and IMMEDIATE DELAYED. DELAYED commands result in instruction sequences which are executed on the timed, interval boundaries. These are instruction sequences which result in communication between the PCC and the devices attached to the dataway. IMMEDIATE commands are executed any time during the interval.

There is also an INVOKE command. This command allows user provided subroutines (called USER MONITORS in Figure 3-4) to be activated, at specified intervals, by the INPUT/OUTPUT Scheduler. They are activated by the INPUT/OUTPUT Scheduler after it completes the appropriate lists of Input/Output instructions and before it exits.

These commands are described in further detail in the following sections.

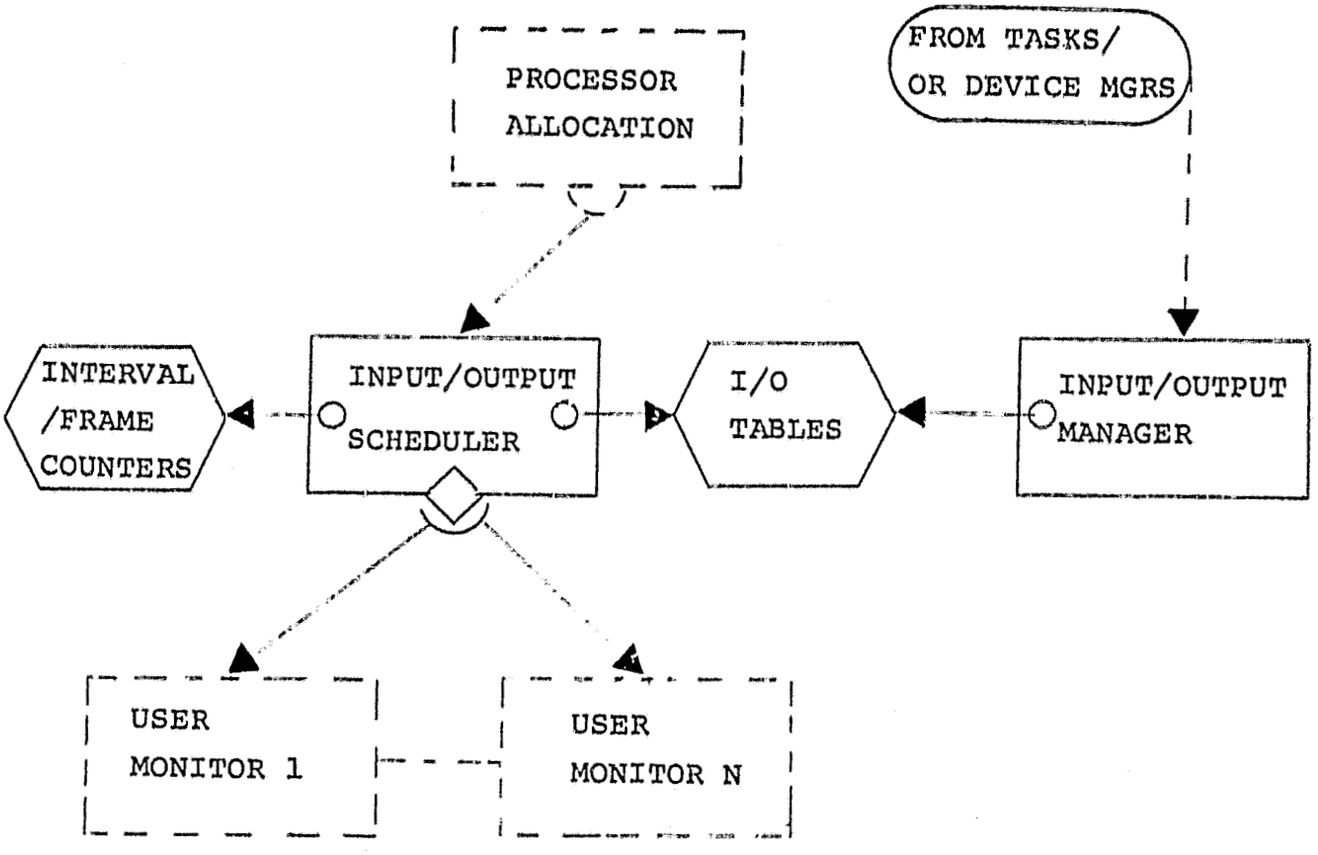
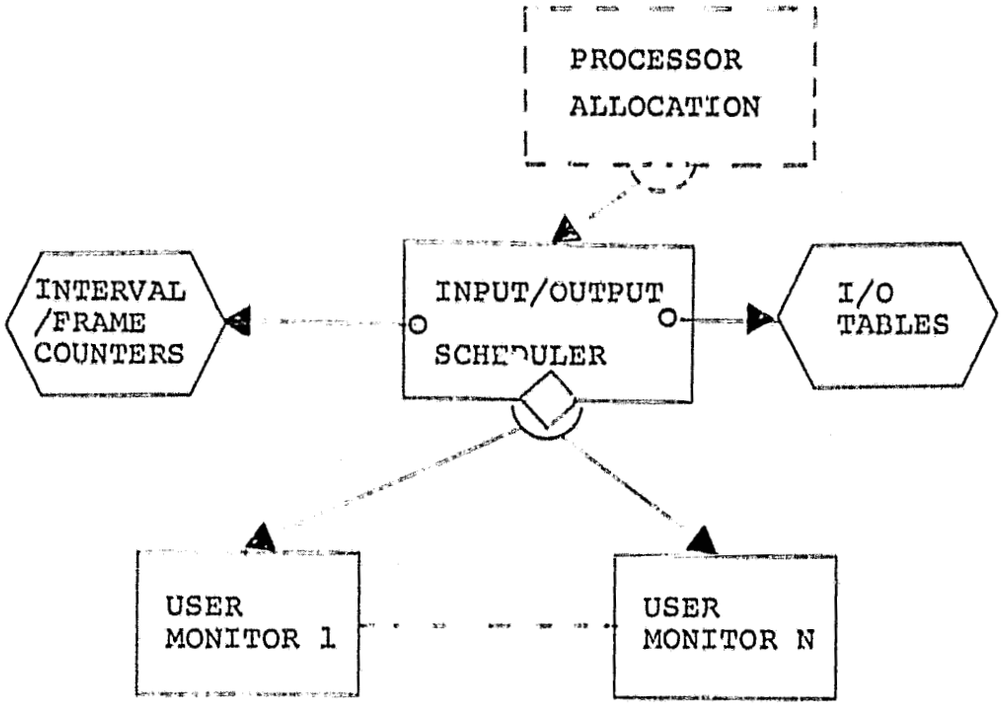


Figure 3-4: INPUT/OUTPUT SUBSYSTEMS DIAGRAM

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.6.1 INPUT/OUTPUT SCHEDULER (IOS)

3.4.1.1 Subsystem Diagram



3.6.1.2 Activation. The I/O Scheduler is activated through Processor Allocation, normally as the first task within an interval.

3.6.1.3 Activates. User provided subroutines (User Monitors) are activated by the Input/Output Scheduler if commanded to do so.

3.6.1.4 Functional Description. The Input/Output Scheduler executes the instruction sequences contained in the I/O Tables at interval times indicated in the I/O Tables. The I/O Tables contain instruction sequences which result in issuance of device commands by the PCC and/or the transfer of data between the PCC and NSSC-II.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

After the I/O Table execution has been completed for a specific interval, the IOS may invoke one or more user provided routines if it has been commanded to do so. This capability is provided to have a quick and simple method for the user to check for certain data conditions and react accordingly. The user routine can perform any function normally allowed to a task. It may be used as an alternate Task scheduling method.

3.6.1.5 Parameters. The maximum number of user routines as well as the maximum size of the I/O Tables activated must be specifiable by the user.

3.6.1.6 Commands. The I/O Scheduler works strictly from the I/O Tables. All commands are interpreted by the I/O Manager (reference Section 3.6.2). The DELAYED type commands affect the I/O Tables. These commands (almost) directly match the available DEPI Service Task actions and are summarized in Table 3-2.

The INVOKE command also affects the I/O Tables and is used to signal which user provided should be activated by the I/O Scheduler.

Any command can be subject to a time condition of the form FROM....TO....EVERY... to indicate the period of time over which the command is executed at the specified cycle rate. If no time condition is specified the command is executed one time at the next interval boundary.

3.6.1.7 Initialization. System initialization needs to reset the I/O Tables.

3.6.1.8 Potential Modifications. Very obviously, different "I/O Boxes" may be used in various applications. This does not need to affect the scheduling concept of the Input/Output Scheduler. The format and dimensions of the I/O Tables may

drastically change however. If the I/O Scheduler is designed such that it "does not care" which I/O instructions are executed, modifications could be limited to the I/O Table access mechanism.

Sooner or later it may become desirable to allow the application programs to access the data from/to external devices symbolically, e.g. by specifying a measurement number. This implies certain additional on-line and off-line supporting functions.

3.6.1.9 Implementation Notes. The (to be designed) DEPI may simplify the I/O software to some degree. For example, multiple word Read or Write logic may be handled in the DEPI. Also the fixed PCC storage locations for pointer blocks may be wired into the DEPI. However, no assumptions have been made in these requirements in regard to any such capabilities.

It must be possible to replace the Input/Output Scheduler with an I/O simulator module for debug purposes. This module, which is part of the implementation version must be able to scan the I/O Tables and provide simulated data for these commands.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

SOS-II SPSME COMMANDS

COMMAND	PARAMETERS
• EXECUTE SEQUENCE (NO RAM)	1) SEQ. PTR. SELECT VALUE ADDRESS 2) ADDRESS TO STORE STATUS
• EXECUTE SEQUENCE (W. RAM)	1) SEQ. PTR. SELECT VALUE ADDRESS 2) RAM. PTR. SELECT VALUE ADDRESS 3) ADDRESS TO STORE (OR'D STATUE
• PUT TO RAM	1) RAM. PTR. SELECT VALUE ADDRESS 2) RAM. PTR. INCR., WRITE DATA ADDRESS n) RAM. PTR. INCR., WRITE DATA ADDRESS
• GET FROM RAM	1) RAM. PTR. SELECT VALUE ADDRESS 2) RAM. PTR. INCR., READ DATA ADDRESS n) RAM. PTR. INCR., READ DATA ADDRESS
• EXECUTE CMND (PCC)	1) SEQ. PTR. SELECT VALUE ADDRESS 2) WRITE DATA ADDRESS OR READ DATA ADDRESS 3) ADDRESS TO STORE STATUS
• EXECUTE CMND (NSSC-II)	1) CNAF CODE ADDRESS 2) WRITE DATA ADDRESS OR READ DATA ADDRESS 3) ADDRESS TO STORE STATUS
• GRADED L	---
• EXECUTE USER MONITOR	

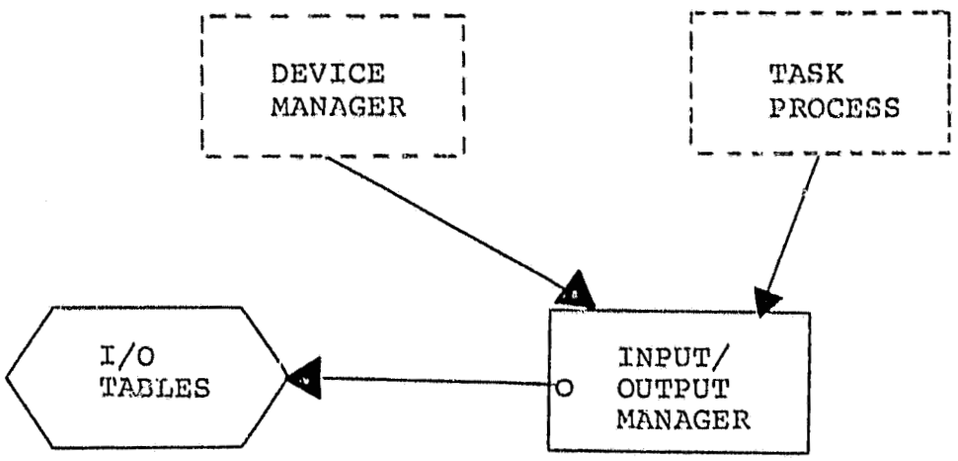
NOTE: 1. ALL VALUES ARE IMMEDIATELY ACCESSED AND STORED IN I/O TABLES

2. COMMANDS CAN BE TIMED: i.e. FROM...TO...EVERY

TABLE 3-2: SOS-II SPSME COMMANDS - DELAYED

3.6.2 INPUT/OUTPUT MANAGER

3.6.2.1 Subsystem Diagram



3.6.2.2 Activation. The I/O Manager is activated any time during an interval by a Task process (which could be a Device Manager).

3.6.2.3 Activates. None.

3.6.2.4 Functional Description. The I/O Manager handles all I/O commands. DELAYED type commands (described under the I/O Scheduler) result in entries to the I/O Tables. IMMEDIATE commands are executed prior to returning control to the application program.

Thus the main purpose of the I/O Manager is to coordinate the execution of I/O commands and the conversion of I/O commands to physical I/O sequences.

3.6.2.5 Parameters. None.

3.6.2.6 Commands. All I/O commands, in fact, pass through the I/O Manager. The DELAYED type commands result in entries to the I/O Tables. All commands listed in Table 3-2, with the exception of EXECUTE USER MONITOR, can be executed immediately. Additionally there is a CAMAC INITIALIZE command which is always executed immediately.

Note however that, to maintain the intended concept of the O/S organization, the DELAYED commands should be normally used. IMMEDIATE COMMANDS should be used only when absolutely necessary.

3.6.2.7 Initialization. None.

3.6.2.8 Modifications. The I/O Manager is wholly affected if other than an SPSME configuration is used. The I/O commands from the task, their interpretation, etc. is likely to be completely different. The subsystem, in such cases, must be replaced in total.

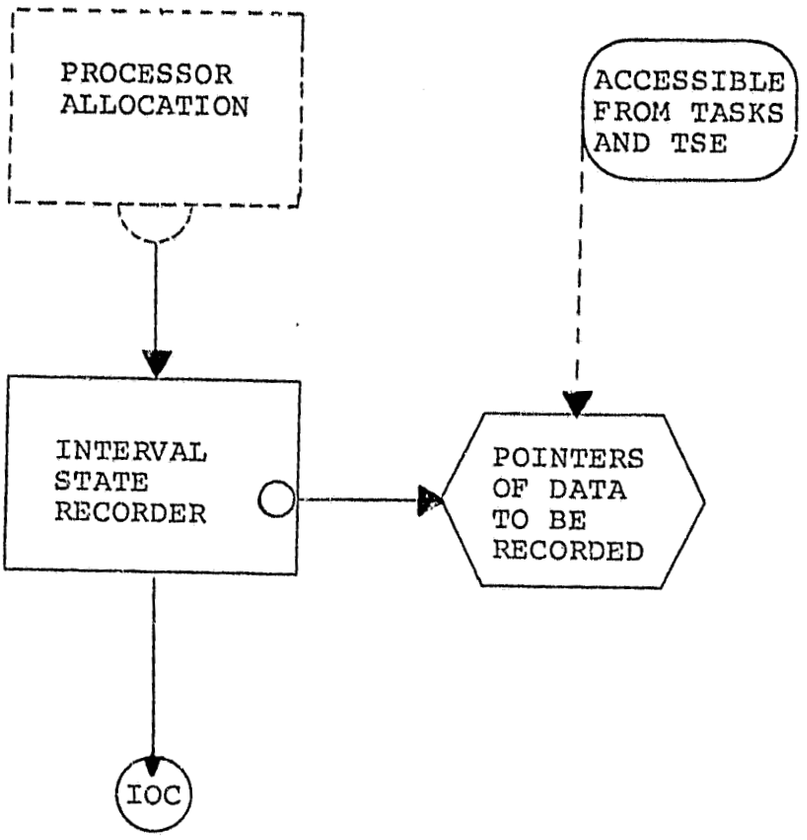
3.6.2.9 Implementation Notes. The Input/Output System may eventually include device managers for standard devices attached to SPSME. Standard devices are defined to be devices for which the functions and communication protocol are known and identical across applications. These may include displays, mass storage, and other processors.

For the initial implementation one device manager needs to be developed to proof the concept. We propose to implement a computer communication module in accordance with the Spacelab EC protocol.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.7 INTERVAL STATE RECORDER (ISR)

3.7.1 Subsystem Diagram



3.7.2 Subsystems Activation

The Interval State Recorder is considered a System Task. As such it is activated under the control of Processor Allocation. This activation is subject to the standard TSC commands.

3.7.3 Activated Subsystems

None.

3.7.4 Functional Description

The ISR is a diagnostic aid. The purpose is to record the state of key tables, events, and other items at that

point in an interval where the Task Process environment has been determined and none of the Application Tasks Processes have yet been started. Of course, other recording points within an interval may be used by simply changing the priority of the ISR.

The intervals during which specific items need to be recorded can be selected by a combination of two methods. First of all, the execution conditions of the ISR can be controlled through the Standard Task State Control commands. Therefore, the ISR can be activated at various periodic rates or upon occurrence of certain events. Secondly, the ISR contains internal logic to record items for selected intervals within a frame.

Note that the amount of information to be recorded must be carefully selected. First of all, all tasks executing during an interval must be completed during an interval. The information to be recorded during an interval is, therefore, restricted by the amount of execution time allocated to the ISR. In early stages of testing, the System Clock may be slowed to allow the desired recording to take place. For operational testing, however, the amount of information to be recorded must be carefully controlled. Secondly the total amount of information to be recorded over a period of time cannot exceed the recording rate of the device used to retain the recorded data.

3.7.5 Parameters

None.

3.7.6 Task Commands

None.

3.7.7 Initialization

None.

3.7.8 Potential Modifications

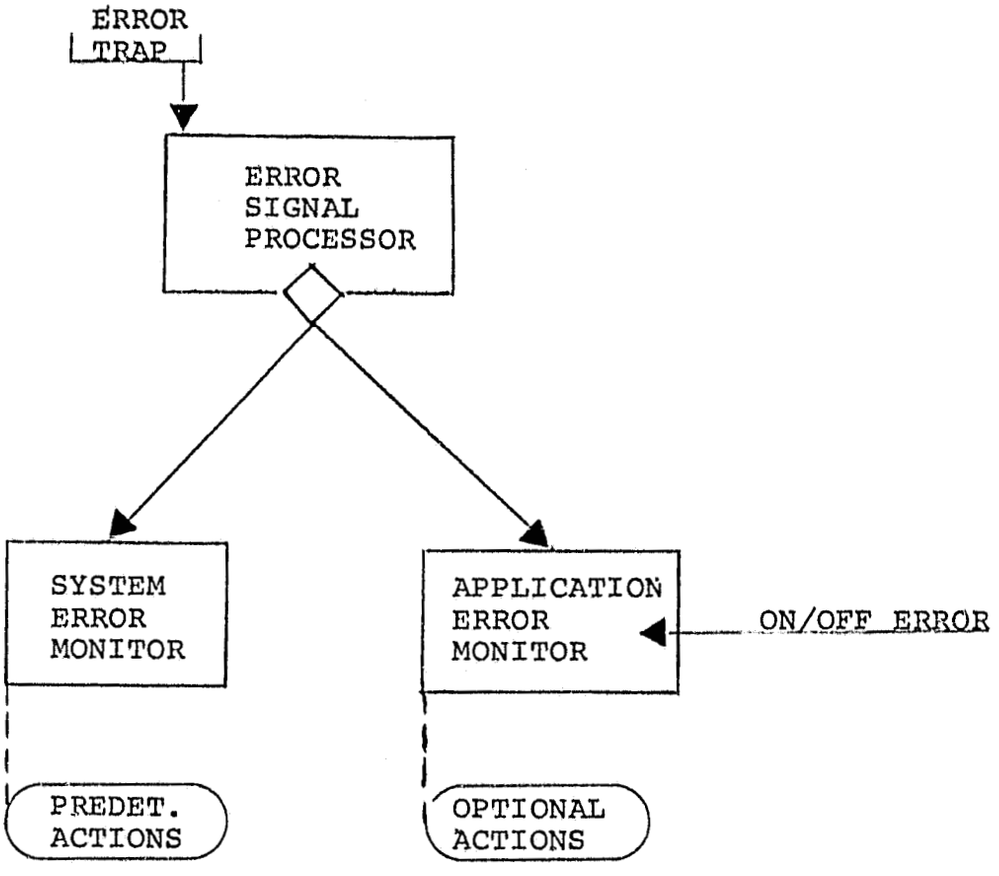
None.

3.7.9 Implementation Notes

The ISR can be used in two versions: a ground version and a flight version. The implemented version will include a component to allow access to the Data Pointers Table from the TSE keyboard. Also the recording component should be easily replaceable.

3.8 INTERNAL ERROR PROCESSING

3.8.1 Subsystem Diagram



3.8.2 Subsystem Activation

The Error Signal Processor is invoked either through an NSSC-II machine check interrupt, program check interrupt, or a SEND ERROR command.

3.8.3 Activated Subsystems

None.

3.8.4 Functional Description

The Error Signal Processor (ESP) analyses the invoking error (identified by a number) and transfers control to either the System Error Monitor (for fixed predetermined action) or to the Application Error Monitor (for dynamically assigned, user defined, actions).

The Systems Error Monitor handles those errors which, in the context of the System, are considered irrecoverable. The action taken is to gather pertinent diagnostic information, whenever possible, and enter the NSSC-II Wait State.

The Application Error Monitor (AEM) transfers control to the location specified (by the user) for the particular error received.

3.8.5 Parameters

None .

3.8.6 Task Commands

1. SEND ERROR - is used by the task to invoke the ESP and to indicate (by the error number) the reason for the invocation.
2. ON ERROR - is used by task to indicate to the AEM where to transfer control for a specific error number.
3. OFF ERROR - is used by a task to undo a previously specified action.

Note: Each of these commands has as a parameter an error number. The error number consists of two integers; the first one indicates an error group, the second indicates the error number within the group. If only one number is given, the action pertains to all errors within the indicated group.

ON ERROR also has as an additional parameter the label where control needs to be transferred.

3.8.7 Initialization

None.

3.8.8 Potential Modification

It is possible that a user may want to modify the predetermined system actions or add additional actions.

The Application Error Monitor may be omitted in a case where the desired actions are few and/or do not vary dynamically. It may also be omitted where AEM is contained within the task complex, as is the case for HAL compiled tasks.

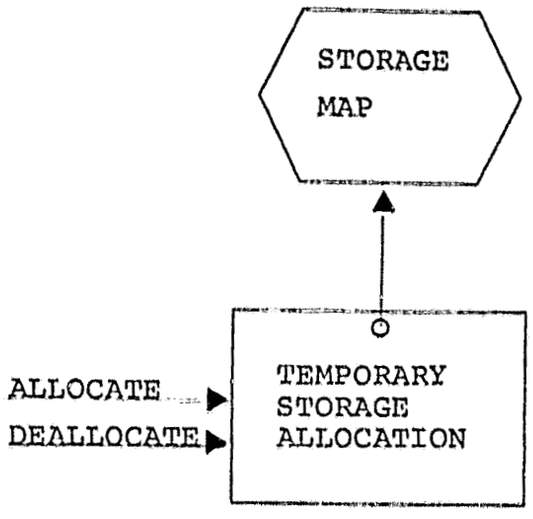
3.8.9 Implementation Notes

The implementation version will assume that the AEM is contained within the Task Complex.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.9 TEMPORARY STORAGE ALLOCATION (TSA)

3.9.1 Subsystem Diagram



3.9.2 Subsystem Activation

TSA can be activated from any Task.

3.9.3 Activated Subsystems

None.

3.9.4 Functional Description

Upon request TSA searches the storage map for the requested amount of storage, updates the storage map, and provides the starting address of the assigned storage block. For deallocation the storage map is merely updated.

3.9.5 Parameters

None.

3.9.6 Task Commands

1. ALLOCATE - used to request assignment of a number of storage blocks (block size TBD). Unavailability results in a System Error.
2. DEALLOCATE - used to request release of a previously allocated number of storage blocks.

3.9.7 Initialization

The storage map must be initialized by System - a Phase initialize.

3.9.8 Potential Modifications

None.

3.9.8 Implementation Notes

This subsystem is useful only for programs translated from Assembly Language. The utility is therefore limited. It can be omitted from the implementation version.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

3.10 INTERPHASE DATA STORAGE (IDS)

3.10.1 Subsystem Diagram

3.10.2 Subsystem Activation

IDS can be activated from any Task.

3.10.3 Activated Subsystems

None.

3.10.4 Functional Description

The purpose of IDS is to retain data in a set of locations which are not part of the Task complex and return this data to the Task complex upon request.

It is particularly useful, when Phase Loads are used, to transfer data between phases.

3.10.5 Parameters

The amount of storage to be set aside within the O/S to store Interphase Data must be modifiable by the user.

3.10.6 Task Commands

1. SAVE is used to indicate a consecutive block of data to be safeguarded.
2. RESTORE is used to return the data to TASK space.

3.10.7 Initialization

None.

3.10.8 Modifications

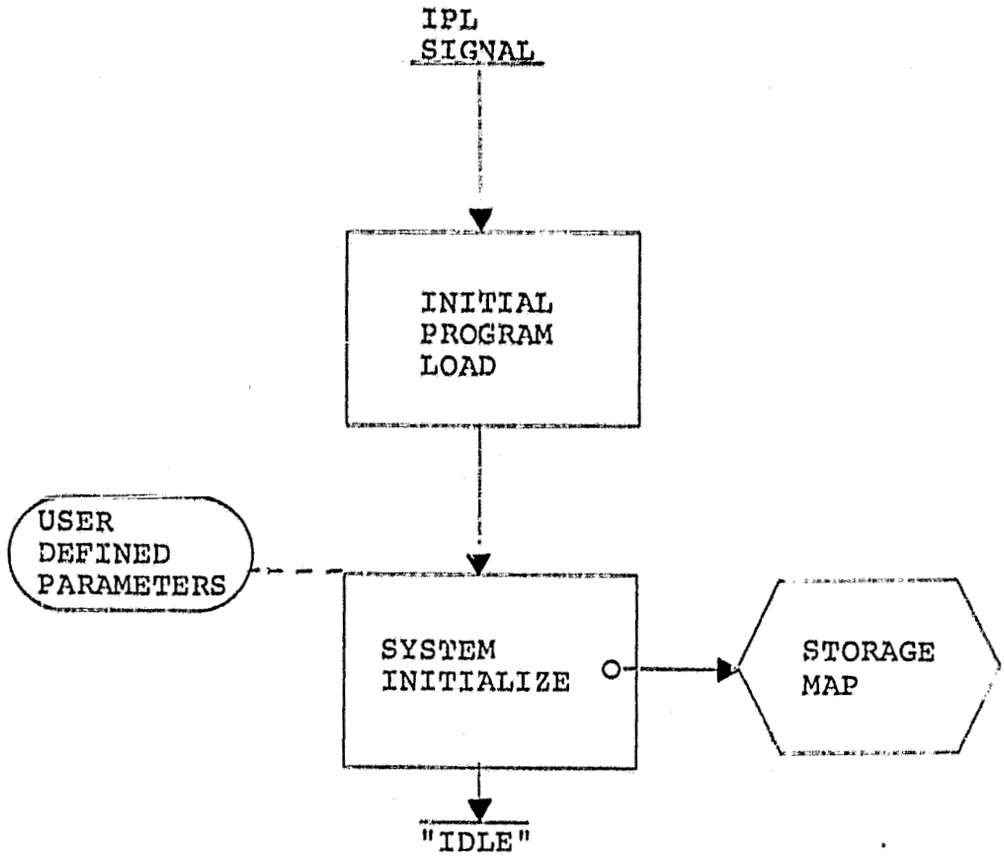
None.

3.10.9 Implementation Notes

None.

3.11 SYSTEM INITIALIZATION (SI)

3.11.1 Subsystem Diagram



3.11.2 Activation

System Initialization occurs whenever the NSSC-II IPL signal is detected.

3.11.3 Activates

None. System Initialization terminates in an "Idle" state waiting for the first interval interrupt to

occur, which in turn allows Processor Allocation to start.

3.11.4 Functional Description

The IPL signal invokes the (microprogrammed) NSSC-II IPL function, which loads the software from a specified device. The load module must be structured such that the first process to be executed, after Initial Program Load, is the System Initialization.

System Initialization performs the initialization for all subsystems in accordance with user provided direction. SI also generates a storage map which is subsequently to be used for storage allocation and/or phase loads.

3.11.5 Parameters

None.

3.11.6 Task Commands

None.

3.11.7 Potential Modifications

A user may want to add application dependent initializations to this module.

3.11.8 Initialization

None.

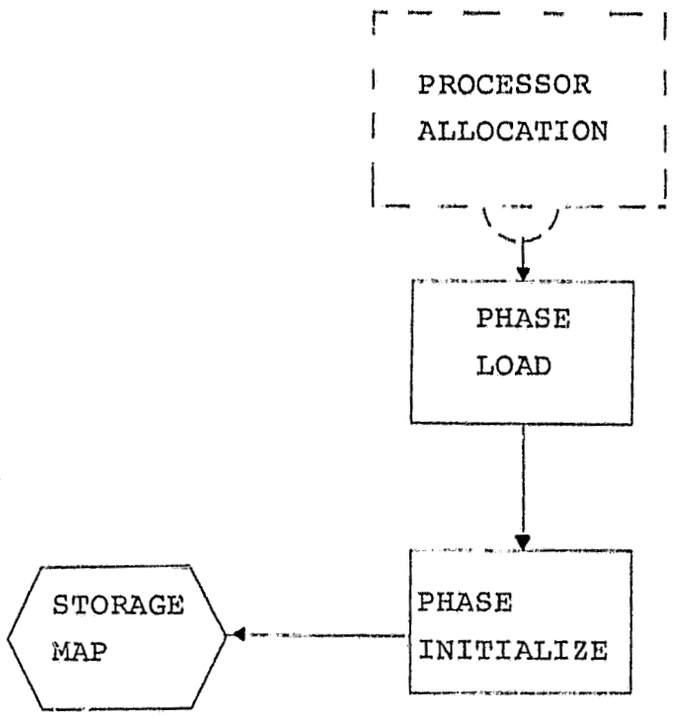
3.11.9 Implementation Notes

It is not currently clear how the IPL function operates in conjunction with SPSME. That is, the DEPI for the NSSC-II is not yet defined in detail.

For the implementation version the IPL can always be performed from the TSE.

3.12 PHASE INITIALIZATION (PI)

3.12.1 Subsystem Diagram



3.12.2 Subsystem Activation

Phase Initialization is treated as a system task. Activation, therefore, is through Processor Allocation.

3.12.3 Activated Subsystems

None .

3.12.4 Functional Description

"Idle" Phase Initialization is the equivalent of a partial System Initialization. Additional tasks are

loaded into storage through Phase Load which transfers control to Phase Initialization. Phase Initialization updates the Task State Control information pertinent to the new Tasks, updates the storage map and exits.

Note that PI will continue over a number of intervals. Prior to activation, therefore, other tasks need to be deactivated, otherwise resource access conflicts may occur. The first application task to be activated must be specified to Phase Initialization and must, upon activation, reactivate the other Tasks appropriate to this Phase.

3.12.5 Parameters

None.

3.12.6 Task Commands

None.

3.12.7 Initialization

None.

3.12.8 Potential Modifications

It is anticipated that most applications will not utilize this subsystem.

3.12.9 Implementation Notes

The implementation version will not contain this subsystem.

4.0 IMPLEMENTATION STANDARDS

The implementation standards that we are concerned with in this requirements document are those critical to the objectives of the NSSC-II Operating System. General standards concerned with project control, documentation, coding, and testing are purposely not discussed here as a wide variety of satisfactory approaches exist. Any comprehensive approach compatible with NASA/MSFC guidelines is therefore acceptable.

The standards discussed in this section are those associated with Application Program Interfaces, Performance, and Component Modularity.

4.1 Application Program Interfaces

The NSSC-II O/S must accept programs generated by the S/360 FORTRAN IV G/H Compiler, the HAL/S - NSSC-II Compiler, and the S/360 Assembler in the format output by the S/360 Link Editor. All communications between application programs and the O/S will be through SVC's.

HAL/S and FORTRAN use significantly different call sequence formats. It is generally more efficient to convert a FORTRAN call format to HAL/S call format than vice versa. Therefore the O/S routines will be implemented such that they can be called directly from HAL/S compiled programs. FORTRAN calls are passed through a FORTRAN call conversion routine prior to entrance to the actual O/S routine. Assembly language routines can use either format.

Other interface standards are described in the HAL/S Operating System Interface Specifications for the NSSC-II. These are compatible, i.e. do not interfere with, the proper operation of FORTRAN or Assembler generated programs.

4.2 Performance

Generally the O/S is to be implemented to minimize core usage because of the limited amount of memory available on the NSSC-II.

However this does not preclude the most frequently used O/S services from being optimized for time. Specifically the following subsystems will have components which must be implemented to minimize execution time: Interval Timer, System Clock, Processor Allocation, Task State Control and Input/Output.

The storage utilization goal is to be able to fit a minimum useful set of O/S services within 8K bytes while the maximum should not exceed 16K bytes for the complete operating system. This may result in some scaling down of the requirements.

The time utilization goal cannot be so clearly stated. That is the maximum amount of time which may be taken up by the O/S services is highly dependent on the usage by the application programs. However there is a fixed minimum amount of time which is absorbed by the Interval Timer, the System Clock, Processor Allocation, and those routines which are normally invoked at the start of each interval. Therefore there is a minimum loss of time which is incurred even if no O/S services are explicitly invoked by the application programs. Although we have no rational basis to select a specific number, it seems reasonable to aim for a maximum time loss of 1-1.5 msec per interval.

4.3 Component Modularity

As discussed in Section 2.0 OVERVIEW the "subsystem modularity" does not imply any specific component modularity. The subsystem requirements specify that there are to be groups of components which, when executed, perform the specified processes and associated function. To establish a desired component modularity, additional guidelines must be provided. It should be noted that such standards may be formed to conflict with the performance goals during the implementation. Trade-offs which weaken somewhat these standards set may occur. It should also be noted that these "standards" are described in terms of our goals rather than in terms of size, scope, and interaction constraints. The latter are well documented in the literature and are to be considered techniques available to meet our standards.

4.3.1 Subsets

The component modularity must be such that O/S subsets can be generated relatively straightforward through selection of subsets of modules. The capabilities of such subsets are identified under Implementation Notes for each subsystem.

The design rules closely related to this requirement are:

1. One function per module
2. Loop free "uses hierarchy"

4.3.2 Modifications

The component modularity must be such that possible changes (as identified under Modifications in Section III) can be demonstrated to be possible.

The design rules closely associalted with this standard are:

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

1. Locate specialized components in separate modules
2. Intermodule interfaces must be designed to be insensitive to the potential changes

4.3.3 Testability

The major potential problem in a "flexible purpose" system is the reverification of a specific, adapted, implementation. If the modularity standards described in the preceding paragraphs are indeed strictly followed, testability is naturally facilitated. However as noted before, practical compromises may be necessary during implementation. The point is that the most important design rule applicable to testability is the strict control of the loop free "uses hierarchy", i.e. an explicit layering of components.

Note that the hierarchy is strictly a component hierarchy, not a subsystem hierarchy. The sequences of components which are executed to accomplish a subsystem process form the hierarchy to which we are referring here. Generally the subsystems start executing a component on some level of the hierarchy and progress through to the lowest level. The currently visible major levels of the hierarchy are illustrated in Figure 4-1.

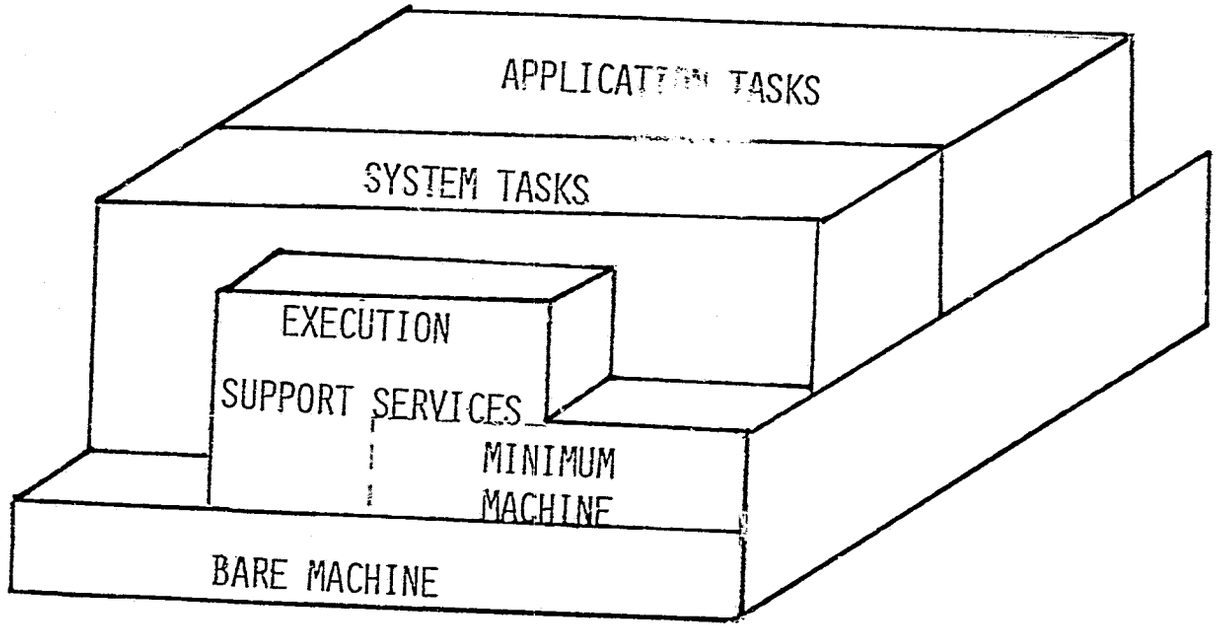


Figure 4-1: Initial Layering Hierarchy

5.0 HAL/S SUPPORT

SOS-II must be designed to support programs compiled by the HAL/S NSSC-II compiler. In simplified terms this means that all SOS-II services can be accessed by HAL/S compiled programs. It also means that certain useful HAL provided functions contained within a HAL program complex require cooperative interfaces with SOS-II.

The purpose of this section is to provide an overview of which HAL/S Language constructs have a meaning within the context of SOS-II and which ones have not. Furthermore, it describes how SOS-II functions, which do not have a counterpart within the language, are accessed. The HAL/S Programmers Guide (IR-63-4) is to be used as the reference for this section. HAL/S statements which interface with an operating system, if available are contained in the following sections of the referenced document.

Section 31 - Interfaces with non-HAL/S code
Section 12 - Input/Output
Section 13 - Real Time Programming I
Section 22 - Additional Input/Output Features
Section 23 - Real Time Programming II
Section 24 - Real Time Programming III
Section 25 - Error Recovery and Simulation
Section 26 - Data Storage and Access
Section 27 - HAL/S and Reentrancy
Appendix B - Built in functions

Section 31 - Interfaces with non-HAL/S code

SOS-II services which are not provided through HAL/S statements are accessed through the O/S SVC macro.

Section 12/22 - Input/Output

The HAL/S Input/Output statements are to be used only to address TSE devices (keyboard, printer, magnetic tape).

Section 13/23 - Real Time Programming I/II

Although the HAL/S Real Time statements are actively used under SOS-II, their interpretation is not identical to that implied by the HAL/S real time concepts. Their factual meaning is as described under SOS-II Task Scheduling.

Major Differences are as follows:

- All task processes are independent.
- Time is interpreted as a frame/interval count.
- Event expressions are not allowed. Only single events are handled.
- All events are unlatched, i.e. they are reset when interrogated.
- SET is used to turn an event on.

Section 25 - Error Recovery and Simulation

SOS-II distinguishes between system errors and application errors. SOS-II receives control whenever an error occurs or is sent. The associated error number is analyzed by SOS-II. If it is a system error, SOS-II handles it directly. If it is an application error, control is transferred to the Error Recovery Executive which is part of the HAL/S program complex.

Section 26 - Data Storage and Access

SOS-II does not support the Update Block concept.

Section 27 - HAL/S and Reentrancy

SOS-II does not protect non-reentrant routines.

Appendix B - Built in functions

Functions which require operating system support are listed under Miscellaneous Functions. Their applicability is described below.

CLOCKTIME - returns GMT

ERRGRP - as specified

ERRNUM - as specified

PRIO - not provided

RANDOM - as specified

RANDOM G - as specified

RUNTIME - returns values of interval counters

NEXTIME (a) - not supported