

AD-A092 887

GENERAL SYSTEMS GROUP INC SALEM NH
REVIEW OF THE FIRE CONTROL SYSTEM.(U)

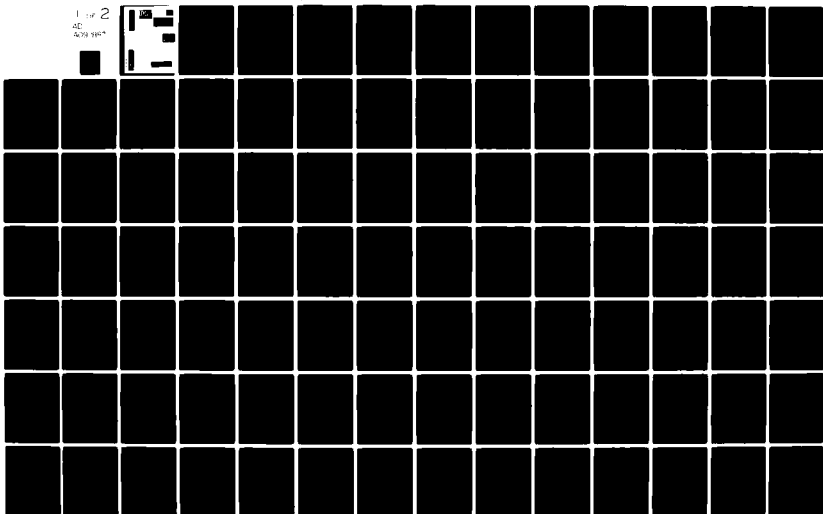
F/G 19/5

UNCLASSIFIED

NOV 80
6560001Z

N00014-80-C-0104
NL

1 of 2
AD
656 985



General Systems Group, Inc.

51 Main Street
Salem, New Hampshire 03079

②

LEVEL II

AD A092887

DEC 11 1980

DDC FILE COPY

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

2

Contract #N00014-80-C-0104

REVIEW OF THE
FIRE CONTROL SYSTEM

Final Report
CDRL #A005

DTIC
SELECTED
DEC 11 1980

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

G. S. G., Inc. 80 12 05 028

Unclassified-

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GSG0001Z	2. GOVT ACCESSION NO. AD-A092 887	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) "Review of the Fire Control System"		5. TYPE OF REPORT & PERIOD COVERED Final 1JUN79 - 30SEP80
7. AUTHOR(s) General Systems Group, Inc.		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Systems Group, Inc. 51 Main Street Salem, New Hampshire 03079		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0104
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 800 North Quincy Street Arlington, Virginia 22217		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) DCASMA - Boston 666 Summer Street Boston, Massachusetts 02210		12. REPORT DATE 18NOV80
		13. NUMBER OF PAGES 139
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) N/A		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) N/A		
18. SUPPLEMENTARY NOTES N/A		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Fire Control System FCS		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report documents a series of studies that GSG, Inc. has performed on the Fire Control System (FCS). The major purpose of our activity was to formulate a set of recommendations on how to best evolve the FCS in light of its present characteristics and the perceptions available of the forthcoming new applications or enhancements of old applications. (continued on next page)		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The method followed in pursuing the above stated goal was to, first of all, become entirely familiar with the technical/design documentation of the FCS Computer System. We then proceeded to examine all available (to us) studies that characterized enhanced applications for the FCS and extrapolated information processing requirements. The latter we examined in a very critical way to determine the soundness of the basic assumptions leading to figures for throughput, and other information processing requirements.

Having done that, and considering the characteristics of the architecture of the FCS, we determined the weak spots of the architecture, in light of the new applications, and recommended a series of moves that could improve it to the point of handling the projected workload.

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

1473B

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REVIEW OF THE FIRE CONTROL SYSTEM

Final Report. 15-77-34

CDRL #A005

Prepared For

SP-23

Department of the Navy

Under

Contract #N00014-80-C-0104

By

GENERAL SYSTEMS GROUP, INC.

12 November 18, 1980

51 Main Street
Salem, N.H. 03070
Tel. (603)893-1000
TWX (710)366-0508
Telecopier (603)893-4723

1700 North Moore Street
Suite 800
Rosslyn, Va. 22209
Tel. (703)524-7242
Telecopier (703)524-0720

G. S. G., Inc.

TABLE OF CONTENTS

	<u>Page</u>
1.0 Introduction	1
2.0 Management Summary	2
3.0 The FCS Operating System	8
3.1 General	8
3.2 Architecture of RTOS	15
3.3 Process Management	22
3.4 Memory Management	34
3.5 Device & I/O Management	38
3.6 File Management	40
3.7 Reliability, Availability, Maintenance	43
3.8 References	48
4.0 The TDCC Architecture	49
5.0 Implementation Tools	54
5.1 Characteristics of the Trident Higher Level Language	54
5.2 THLL Translatability to PASCAL/ADA	58
6.0 Software Development Methodology	63
6.1 Software Development Life Cycles	63
6.2 Programming/Design Practices	67
6.3 Development Environment	74
7.0 TDCC Software States and Operational Sequence	78

TABLE OF CONTENTS

Continued

	<u>Page</u>
8.0 Additional Processing Requirements for the Applications	86
8.1 November 10, 1977 Throughput Working Group Meeting	88
8.2 January 24, 1978 Throughput Working Group Meeting	97
8.3 June 21, 1978 Throughput Working Group Meeting	107
8.4 April 6, 1979 Throughput Working Group Meeting	113
8.5 October 11, 1979 Memory Study	119
8.6 June 10/11, 1980 Dahlgren Program Review	126
8.7 GSG Conclusions	131

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

1.0 Introduction

This report documents a series of studies that GSG, Inc. has performed on the Fire Control System. The major purpose of our activity was to formulate a set of recommendations on how to best evolve the FCS in light of its present characteristics and the perceptions available of the forthcoming new applications or enhancements of old applications.

The method we followed in pursuing the above stated goal was to, first of all, become entirely familiar with the technical/design documentation of the FCS Computer System. We then proceeded to examine all available (to us) studies that characterized enhanced applications for the FCS and extrapolated information processing requirements. The latter we examined in a very critical way to determine the soundness of the basic assumptions leading to figures for throughput, and other information processing requirements.

Having done that, and considering the characteristics of the architecture of the FCS, we determined the weak spots of the architecture, in light of the new applications, and recommended a series of moves that could improve it to the point of handling the projected workload.

2.0 Management Summary

As stated in the introduction of this report, the purpose of the GSC study was to understand the current architecture and design of the Fire Control System (FCS), the list of the proposed new applications for this system, and a series of studies on throughput assessment and other information processing requirement projections, to then formulate directions for evolution of the FCS System.

In doing our task, we were handicapped by a couple of very important elements. The first element being a considerable amount of program redirection that took place while our study was proceeding namely, at the start of the activities the program was pegged around the long term Trident II Program. This program was subsequently replaced by a shorter term and a more evolutionary program, the C-4 Upgrade (C4U) Program. One important characteristic of the C4U program is that it, apparently, has abandoned the premise of the Trident II Program that many of the on board computers could be replaced by redesigned machines.

The second element that influenced our activities was that, contrary to our normal methodology, we were largely left to study this program from the written documentation. In fact, the inputs we received from the project team were extensive amounts of technical documentation and a number of formal briefings that largely depicted the external characteristics

of the system. In other words, there was a dearth of informal, round table, interactive sessions with the key designers of the system. It has been GSG's extensive experience, that reviews of this type performed strictly from formal documentation, or formal briefing material, are handicapped in reaching the proper perception of where the project is at a given point in time. The reason why this is so is that these projects are usually always very dynamic, and situations change too rapidly for the formal documentation to catch up. Furthermore, more, formal documentation very rarely has enough information on the rationale for choices made.

The above two factors, of course, interact strongly with one another, namely, the fact that the program was undergoing a major redirection made it even riskier to study it strictly from formal inputs, or put another way, would have suggested the desirability for extensive and informal communication to supplement the information available through the formal documentation.

The results documented in this report are, therefore, to be examined keeping in mind that they are the logical conclusions that derived from the information available to us. Even with this kind of limitation, we believe that we have achieved a number of important insights in the nature of this program, and that these insights are not necessarily obsolete because

of the unavailability of more recent information that has not been communicated to us. The major conclusions we have arrived at in this study are:

- a. The basic architecture of the FCS Computer System is comprised of the architecture of the computer itself, the TDCC, and that of its operating system, the RTOS. As far as the TDCC is concerned, we see that its architecture is capable of handling forthcoming applications with the exception of two aspects. One is that the addressability of the processor, especially at the level of the application process has to be expanded, the other is that the raw throughput in thousands of instructions per second (KIPS) must be improved by approximately an order of magnitude. The RTOS is a satisfactory design that should be able to handle all projected applications.
- b. The program design, implementation, documentation approaches adopted by the FCS project are quite satisfactory and provide a good base for further evolution of the software.
- c. The sequel of throughput studies that was available to us, shows in an undeniable way that the project does not have sufficient capability for architectural design. Architectural design being concerned with the major partitioning decisions and major design trade-offs which usually determine whether a

computing system will be able to do the intended job. Key indicators that lead us to this conclusion are that, in the throughput studies, a lot of work was expended in arriving at very accurate information on throughput. For instance, instruction mixes were computed for a number of numerical algorithms. The reason why this very high accuracy is an indicator of poor architectural design capability, is that experienced architects know that design quantities, at the broad computing system level, cannot be pinned down with accuracies much better than plus or minus 30%. This is not because the art of designing computing systems is imperfect due to a lack of knowledge, but rather because the intrinsic nature of computing systems is that of being queuing systems. Queuing systems have to be designed with sufficient slack in their resources to avoid saturation and system collapse.

This lack of computer system architectural sophistication, together with the apparent unfamiliarity of the project team with the process of independent reviews, leads us to believe that it would be very important for programs of this type, in view of the crucial national interest involved, that a practice of periodic reviews by independent groups be adopted.

Many a development group has discovered that, especially when one is dealing with complex software systems, the exercise of going through an independent review is most beneficial to the design group itself. The major reason is that it forces the design group to articulate fully their assumptions and rationales so that they can explain it to a set of competent peers. It is the very process of articulating and formulating clearly assumptions and rationales of choice, that forces a better more robust design which has considered more exhaustively all available alternatives.

Besides the above recommendation for adoption of a regular policy of repeated design/program audits, the report gives a number of major recommendations. We believe that our recommendations on how to extend or evolve the FCS System will prove out to be valid even in light of information we may not possess at this time. The reason why we believe that their validity is independent of information that we may not have, is that it became very clear, during the study, that the FCS System is primarily throughput limited and addressing span limited. Our major recommendations are as follows:

- Initiate a study to examine the feasibility of reformatting the instruction layout so that room for a greater span of application process addressability is found.

- Expand the process address space by at least two bits (i.e., 64KW→256KW). Such an expansion of the process addressability would also solve the memory problem, that is it would allow utilizing fully a one M word memory, which is within the realm of physical possibility at this time.
- Commission a detailed design of a CPU which is (software) upward compatible from the TDCC and has approximately an order of magnitude greater throughput power.

3.0 The FCS Operating System

3.1 General

The RTOS is implemented for the Trident Digital Control Computer (TDCC). This computer features a 32 bit, modern, scientific/process-control, word oriented, architecture (See Sec. 4.0). The only major criticism of the architecture, from an operating system point of view, is that it implements a virtual memory scheme based on the utilization of 16 base registers, each spanning up to 4K words for a total of 64K words. The architecture has two sets of registers, one for the user task and one for the exec.

What this implies is that the virtual memory scheme that can be implemented on this architecture limits the virtual address space of a particular RTOS process (task) to 64K words. At the time that the system was designed, this kind of address space was more than sufficient. However, in current designs, especially commercial ones, the tendency is to be far less restrictive in the structure of the address or name space.

In fact the tendency in commercial architectures, which are 32 bit based, is to achieve as much as 24 bit of addressability on a per process basis, as opposed to the 16 bits of addressability that the TDCC achieves on the per process basis. This means that a modern architecture can easily achieve as much as 4 megawords of addressability on the per process basis. It may rightly be asked why one would need such a huge address space since it is

unreasonable to expect that programs, especially applications, will achieve this sort of size. However, the key point of achieving these very large name spaces is for the handling of data. As we shall discuss in Sec. 8.0, one of the easiest ways to unblock the architecture for advanced applications would be to allow very large data segments and this is unfortunately impossible with an architecture that has only 16 bits of addressability.

Chapter 6 of Ref. 2 is a very detailed, proselike, description of the monitor design. It is well organized and furthermore, Appendix A amplifies this very detailed documentation by giving a set of detailed PDLs that document the design.

What is missing in order to make the documentation of this design absolutely complete are two fundamental things.

The first being a more formal recognition and treatment of the abstract (virtual) machines involved. What we are referring to is that a section such as 5.2 Architecture Overview, Ref. 2, should have been a little more complete. In fact, in Sec. 5.2, the architectural view is broken down by indicating that the system is comprised of four basic abstract machines. They are:

- Basic processor hardware
- Monitor
- State Executive
- Application tasks

(See Fig. 5.2-1 RTOS Structure, Pg. 5-3, Ref. 2.) Our point is that within, certainly the monitor, and possibly the state executive, it is possible to isolate further layers of abstraction, or collections of primitives, which represent the fine modularity of these two portions of the system. By the way, this can also be true of specific application tasks.

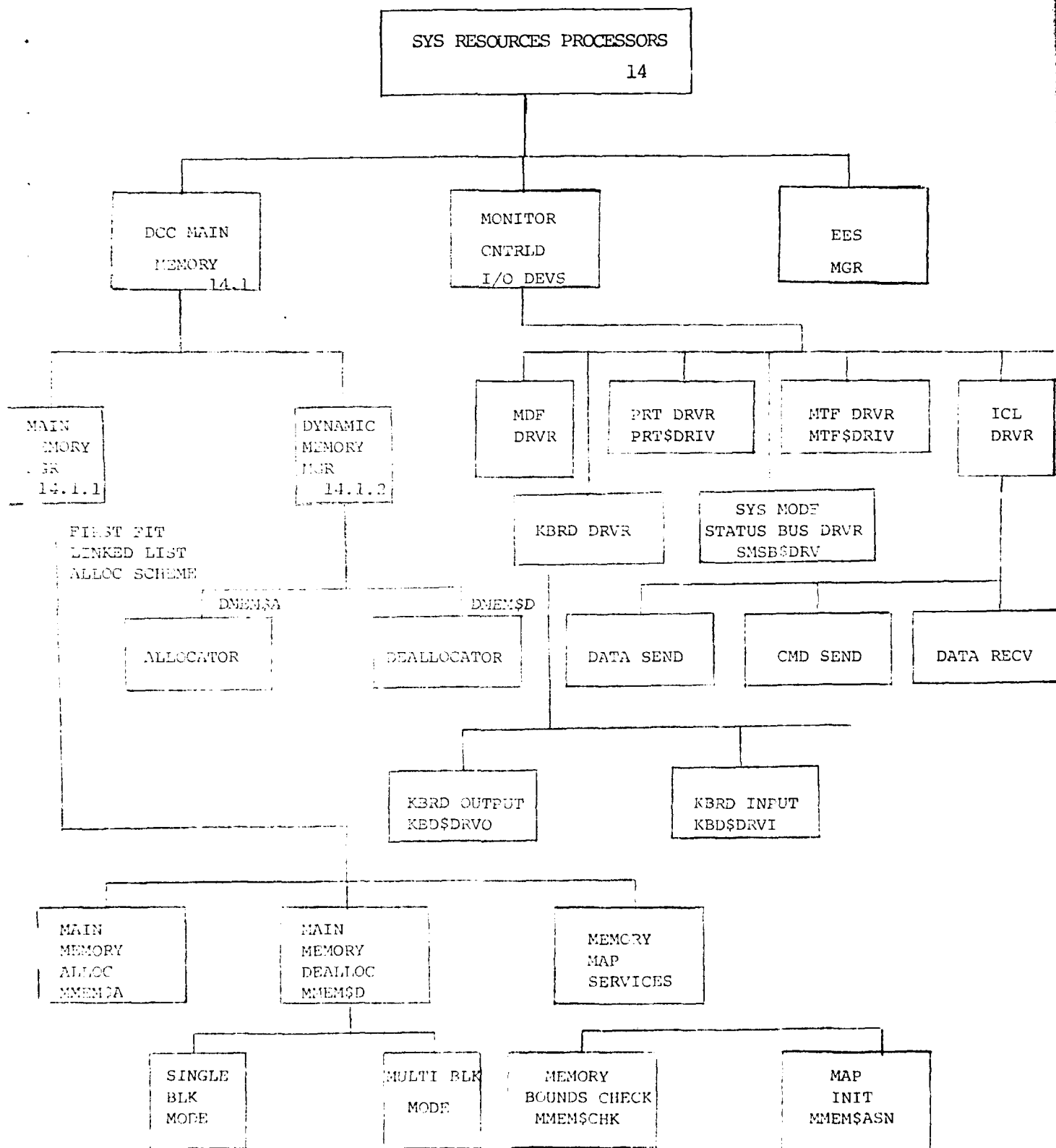
The other missing factor in completing the documentation of this design is a collection of functional diagrams (HIPO-like). To illustrate what we mean by HIPO-like diagrams, we have attached an HIPO-like diagram for the system resources processor functionality. Basically, these diagrams create a road map of which modules implement which functionality, and what is the hierarchical relationship between the different modules.

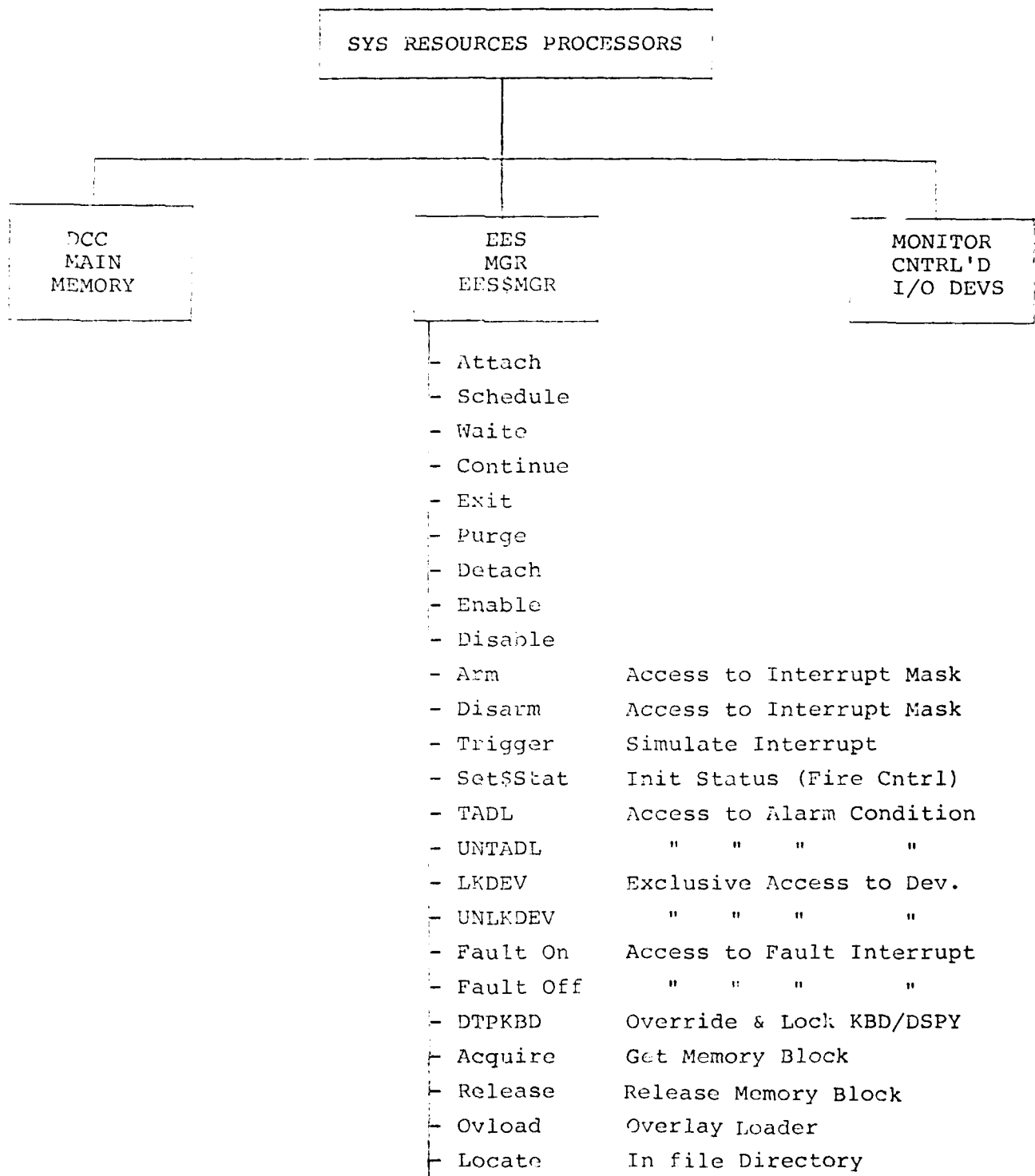
The importance of documenting and abstracting structural information, such as the documentation of the abstract machines involved, and the functional decomposition trees, is that it makes the maintenance and further evolution of the system a lot simpler than conventional documentation.

To illustrate these points, at least to illustrate the technique of an hierarchical functional decomposition tree, we use the example of the monitor system resource processors, that is, the monitor system resource management primitives which include the following:

- Digital Control Computer (DCC) main memory
- Monitor controlled I/O devices
- Enter Executive State (EES) service modules

- Memory management which comprises two major routines, as follows:
 - Main memory manager: this is the space allocator for the tasks (processes).
 - Dynamic manager: this is the allocator that allocates monitor space for control blocks and buffers.





EES MGR. CONTINUED

- Sum	
- Read	Dsk/ MTU → Memory
- Write	Dsk/ MTU ← Memory
- DIO	Direct I/O for Process
- Print	
- Piprint	Periodic Print
- Display	
- ICL	
- Sys\$Err	Define Process Own Sys Error Codes

3.2 Architecture of RTOS

The Real Time Operating System (RTOS) can be configured for each of the system modes of the Fire Control System. The system modes correspond to: the test mode, the tactical mode, which corresponds to the actual operation of the weapons system, the stand-by mode, which allows the updating of various key data bases so that the weapon system is ready for employment, the data entry mode which allows in-putting of new information in the data base in preparation of stand-by and tactical, etc.

This tailorability of the OS to the system mode is accomplished by splitting each OS into two portions, one is the monitor and the other is the executive. The monitor contains all the functions which are common to all the system modes, the executive contains the functions which are needed by a specific mode.

The concept of executive, in the case of the FCS RTOS, is slightly different than the classical concept of executive. The classical concept of executive being, as is well known, a subsystem that implements the operator/user interface to the operating system. RTOS executives do implement this user interface, however, they also include a portion of classical operating system code which is peculiar to a particular system mode. For example, if a particular device is needed only in a specific system mode, the device driver and operating system code related to that device would be part of the executive and not of the monitor.

Page 5.5 of Ref 2 states, "The monitor provides routines for performing I/O operations to the following fire control devices: MDF, MTF, printers, keyboard, ICL, SMCL, HADL, DCSS and the TODSS. I/O operations to the other fire control devices are the responsibility of the state executives."

We presume that the rationale for this division is to guarantee that access to certain fire control devices is only possible when a specific state executive is in control. The system has a state executive for each respective state, a state being a combination of a system mode (of operation) together with a specific configuration.

Each state executive is capable of triggering into execution a number of application programs and the linkage between the application programs and the specific state executive is obtained by entering the file descriptors of the application programs into the file directory of the state executive. That is, the specific application programs associated with a particular state executive are "known to it" in the sense of the general literature on operating systems.

Further tailorability of the system is provided by the notion of tailorable options. The tailorable option is a system function which has to be supplied by the state executive and whose definition therefore is delegated to the state executive. Page 5-21 of Ref 2 gives the list of tailorable options for the state executive as follows:

- Local file directory (required).
- Special enter executive service (EES).
- Special fault processing.
- Special I/O devices and associated interrupts (DRISS, DRS processing).
- Special keyboard processing.
- Free and post processing routines.
- Certain internal and external interrupt processing.
- Special processing for some monitor control I/O devices.
- Task/executive control common area (TECCA).

Since at any given time there is a single OS composed of a combination of a monitor plus a state executive, it is possible to override, in principle, the design time functional split between the monitor and the state executive. In fact, the RTOS system does provide a mechanism for such an override and that mechanism is called the Dynamic Tailorability Processors (See page 6-48 of Ref 2). This mechanism basically amounts to overriding the Dedicated Interrupt Cells (DICS) which have been loaded up at monitor initialization time. Such override would allow a state executive to substitute its own personalized fault/interrupt processing. Consequently the state executive could take over and customize the processing for a DCC internal fault and also to customize the I/O operations to devices which are normally processed by the monitor.

As we understand it, this override facility is implemented by resorting to a number of Executive Special Processing (ESP) words where each bit represents a particular interrupt/fault processing routine. If the bit is set, the corresponding routine will be executed. At execution time when the fault/interrupt occurs, the initial trap is always to the original monitor dedicated interrupt cells. However, examination of the corresponding ESP bit determines whether the fault/interrupt is processed by the state executive routine or the monitor routine.

For obvious reasons, any executive that takes advantage of this override capability must satisfy two conditions. (A) They must provide, at state executive initialization time, the necessary transfer tables, and (B) any processing done by state executive interrupt/fault processing routines will be timed by the monitor. Both conditions are obvious, the latter one, of course, is to prevent a situation where faulty state executives would cause hung conditions.

The architecture of the RTOS system seems to have a basic DIJKSTRA modularity consisting of four layers, the basic hardware, the monitor, the state executive, and the application tasks. It appears that within each one of these elements there is no further DIJKSTRA modularity.

The RTOS system is a multi-tasking system which is capable of full multi-programming since process scheduling (See Sec 3.3) is asynchronously driven by external interrupts.

The above represent the major architectural principles that can be extracted from Ref 2. Ref 2 also gives some indication of some potential problems in the design. In particular, on page 6-130, in discussing the loading of a task from disk, it indicates that the Attach primitive (See Sec 3.3, Process Management) can be called by interrupt routines. Since the Attach primitive can be called by an interrupt routine, it must be short, so this forces deferring the disk/tape load to a later time to be done at a processing level below interrupt processing.

This is an indicator of poor design since, for at least the last decade, operating systems have been designed so that any primitives connected with process management and scheduling are implemented by a mechanism of queues, where the interrupt processing routines only mark the queues. A background scheduler (processing at a level below most interrupt processing) will actually do all of the work that is involved in performing the scheduling primitives.

The mechanism of limiting the interrupt processing to simply marking process descriptors on the scheduler's queues, effectively renders scheduling completely asynchronous from the external interrupts. (This in turn, removes all requirements for scheduling primitives to be very, very short and even more importantly, it removes the requirement for frequent and often extended

disarming of the interrupt system.) The latter is a crude way of protecting extended critical sections.

One other comment with regard to the architecture of the RTOS is that, as presently described by the RTOS monitor design disclosure document, the architecture is captured by a significant volume of prose. Furthermore the prose is not very well organized from a point of view of an architectural description, since for instance, points that have to do with the actual gross structure of the system are scattered over chapter five and six, points regarding process management are scattered again over many, many sections, etc. However, the greatest single deficiency is not to have resorted to some of the standard techniques of documenting the functional decomposition of the architecture. What we are referring to is the use of either Visual Table of Contents diagrams or HIPO charts to indicate how the different architectural modules are decomposed into functions and subfunctions.

We believe that this is an unfortunate deficiency since the design, as we comment elsewhere is quite good, the degree of prose/documentation describing the design is more than adequate, and then finally, the fact that the project adopted the very rigorous discipline of documenting the entire programming design by the use of the program design language (PDL) with the PDL listings

being made an integral part of the design documentation. Having done all this extensive effort, a small amount of additional effort would have made the overall design disclosure document much more effective, especially from the point of view of new personnel trying to delve into a design unfamiliar to them.

3.3 Process Management.

In this section, we will present the information given in Ref. #2 with regard to task scheduling in the perspective of current understanding of the theory of operating systems.

RTOS appears to be an operating system based on the notion of processes. In fact RTOS tasks share most of the attributes of the abstract notion of processes. The RTOS processes (tasks) can be divided into three broad categories; those which are in the dormant state, those which are in the connected state, and finally, those which are in the active state.

Ref. 2 pg 6-102 defines dormant, connected, and active tasks as follows:

- Dormant has no exclusive resources. In particular it has no control point.
- Connected has exclusive resources and at least a control point. The system admits 32 control points with ranks 1 through 31.
- Active. A task is active if it has an activation record, main memory, and is in some stage of execution.

In view of the fact that the control point table appears to be the main entry table for all scheduling decisions we would read this definition of state in a slightly different way. Namely, it

seems to us that the most important distinction is whether or not the task or process is known to the system. As it is customary in the literature on operating systems, an entity is considered known to the system if the control table corresponding to that class of entities has a specific entry relating to the entity. From that point of view, we see that tasks which are dormant are not known to the system since the control point table does not have a corresponding entry for them, while tasks which are either connected or active are known to the system.

Another way to put it is that the transitions from the dormant to the connected state and vice versa, correspond to the macro level of scheduling of tasks, or in other words, introducing tasks for consideration by the system. The transitions between the connected and the active states, and their inverse, correspond to processes being entered or exited from the lower level scheduling process. This lower level scheduling, as it is well known, is the true process scheduling and basically is the activity of resource contention for processes. In other words, once a process enters the active state, it is competing with all other processes for resources such as memory, CPU, channels, and peripherals.

The active state in turn is divided into three sub-states: enabled, executing, and disabled. Once again, referring to standard operating system literatures, they correspond respectively to runnable, running, and blocked.

In the enabled state a process has just come out of a wait, therefore, it satisfied the wait and it is contending for the CPU so to be able to run. In the executing state the process has all the resources it needs in order to execute. In the disabled state the process has been blocked by either requesting an I/O operation, and therefore be forced to wait for its completion, or by executing the primitive that puts it to sleep.

Page 6-107 of Ref. 2 describes the scheduling primitives that the RTOS monitor provides to higher level software such as task executives. The primitives are as follows:

- Attach primitive (ref 2 does not refer to primitives but rather to functions). This primitive effectuates the transition from the dormant state to the connected state. In other words, the Attach primitive presents a process to the system and makes it known to the system.
- Detach primitive. This primitive does the opposite operation of the Attach, namely, it deletes a process from the list of known processes freeing up the control point that was assigned to the process. Thus this primitive effectuates the transition from the connected state to the dormant state.
- Schedule primitive. This primitive effectuates the transition from the connected state to the active state and in particular to the enabled sub-state. In other words, this primitive poses a process for lower level scheduling, that is, for actual resource contention.

- Exit primitive. This primitive executes the transition from the active state, in particular from the executing sub-state to the connected state. In other words, this primitive causes the process to abandon resource contention with other processes and transfer to a condition where it is only known to the system but not competing for resources.
- Purge primitive. This primitive is executed whenever an abnormal termination of a process occurs. In other words, it is similar to exit, except that exit is executed for normal termination of a process while purge is executed when a process causes a fault or other offending condition. This primitive effectuates the transition from the active state to the connected state. In other words, the purge primitive can be executed out of any of the sub-states of the active state leading to the connected state.
- Waite primitive. This primitive effectuates the transition from the executing sub-state to the disabled sub-state of the active state. Thus this primitive causes a running processes to suspend execution and become blocked.
- Continue primitive. This primitive effectuates the transition between the disabled sub-state and the enabled sub-state of the active state. That is, this primitive causes a blocked process to become unblocked or runnable and therefore to queue up for processor scheduling.

The three groups of primitives, Attach and Detach; Schedule purge, and Exit; and Continue and Waite, cover all of the transitions of the state diagram which includes the states: Dormant, Connected, Active (Enable, Disable, Executing) except the transition between the enabled and the executing sub-state of the active state. This transition is effectuated by an interrupt processing routine, namely the clock interrupt.

The RTOS monitor has a clock interrupt processing routine which processes a two hundred millisecond clock interrupt. This clock interrupt will take care of:

- I/O time out.
- Printing out monitor messages.
- Hardware alarm processing.
- Read navigation and time of day information.
- Timing out of processes which have been put to sleep by a Waite primitive.

Together with the periodical clock interrupt the system has to process also the interrupts that come from I/O operations and unsolicited interrupts that arrive from operator actions. The interrupt processing routines for all of these interrupts are capable of looking at the set of enabled tasks/processes and picking the next one for execution. In other words, the transition from enabled to executing occurs upon an interrupt processing routine finding out that the current executing task (CET) can no longer run and by looking at whether there is any task that is ready to run and giving to it the processor.

Page 5-15 of reference 2 indicates that the scheduling primitives described above are the sole interface for scheduling functions on the part of user software. In other words, the task executives can set up their own scheduling policy by executing the scheduling primitives described above.

Another way to explain all this is to consider that the transitions between the various states, dormant, connected, enabled, etc., can be performed in two ways. One is by a higher level software executing a scheduling primitive to obtain the services of the monitor for scheduling purposes, the other is for the interrupt processing level of the monitor to actually effectuate some of these transitions. For example, the transition between disabled and enabled can either be effectuated by executing a Continue primitive which is a call on the monitor, or it can be executed by an I/O interrupt processing routine recognizing that an I/O operation has been completed and that a process that was blocked for that I/O operation now is ready to proceed and therefore can be put in the enabled state.

Page 6-107 of Ref. 2 further confirms the above understanding as follows "The scheduler provides the nucleus around which executive dependent scheduling algorithms are built".

Page 6-105 of Ref. 2 describes the activation record for a task as inclusive of 434 words of main memory. This memory being dedicated to storing two stacks and one fixed size storage area. The first stack is 136 words and is the executive stack whose name

is EX\$STK. The second stack includes 256 words and it is used by both the monitor and state executive and it is called with the name DATA\$STK.

The first stack appears to store linkage information. The second stack is stated to store callers context information only.

The fixed storage area of 42 words is used to store the task environment information, i.e., this appears to be a process control block.

To complete the subject of process management, the remaining topic, that needs to be treated, is the issue of process scheduling priority. This issue is treated by RTOS by utilizing the notion of process rank. The system is designed to have 32 distinct ranks or priorities the number 32 being chosen because it corresponds to the size of the computer word so that a particular bit in the word can be utilized to indicate the presence or absence of a task of a given rank. This very fundamental architectural decision is in turn predicated of the idea that this allows very fast scheduling algorithms to be implemented. In fact, logical operations on machine words is all that is required to implement such algorithms. Page B-2 of the Ref. 2 clearly states that this indeed is the mechanism used at the base of RTOS process scheduling. In fact, it says, "There are presently defined in the extended computer, 32 possible control points per machine state. Each is equivalent to a task rank within the MTS. This rank shows the relationship between tasks. Equated to priority as it does with interrupts. It is also a quick and efficient way of identifying a task and its corresponding monitor control information in the MTS".

Page 6-109 of Ref. 2 indicates that the rank from 0 to 31 corresponds to 32 priority levels each allowing a single task, that is, rank is used as an index in the CPCON table of control points.

Obviously for this mechanism of rapid access to the control point table to work, there has to be a single known process associated with each control point. It follows therefore, that this design allows only 32 connected processes or tasks since the maximum amount of control points is 32 and since each control point must have a single known process, that is connected process. This mechanism does not put any particular restriction on the number of dormant processes. Confirmation of the uniqueness of the connected tasks to a control point comes from page 6-129, of Ref. 2, in the discussion of the Attach primitive states: "If some other task is connected at a control point, then a control point conflict error exists." In other words, in the process of making a process or task known to the system (Attach), if the control point or, what is the same the priority, that has been chosen for the new process is already assigned to some other process, the Attach primitive will give an error condition.

Two important algorithms in the monitor are the TAC (Task Activation Controller) and the TTC (Task Termination Controller). The process termination routine TTC is called in the same way as the process activation routine TAC. One key difference is that the TTC runs at priority (rank) 0 while TAC runs at the same priority as that of the process being activated.

TTC running with priority ensures that the release of system resources occurs with the highest priority.

Page 6-113 of Ref. 2 states, "Since only members of the MTS can be terminated, it is sufficient to represent the tasks (to be terminated) list as a word in which each bit set indicates a rank of a task to be terminated.". This particular sentence conclusively proves:

1. At each priority level there can be a simple known process.
2. Hardware Dependency (32 bit word) at the very heart of the process scheduler.

Furthermore, page 6-121 of Ref. 2 indicates that all scheduling queues are implemented as single word bit mask algorithms.

On page 6-114 of Ref. 2, a very detailed verbal description of the process termination algorithm is given. At this time, as well as many other places in the monitor document, the use of a few flow charts could have saved an enormous amount of prose and resulted in greater clarity.

In conclusion, the process management scheme proposed by RTOS is a fairly classical interrupt driven process management scheme and our only serious reservation about it is the architectural decision to use the word dependent quantity of 32 bits as the basic underpinning for process priority handling. We understand of course the rationale for it, which is that once 32 priority levels are chosen, one can associate each of them with the particular

bit position in the computer word and that, in turn, would allow many compare and selection operations to be based of fast logical operations on computer words. On the other hand this kind of architectural decision invariably winds up being a poor one in the long term because it imposes some very hard limits on the expandability of the architecture. In other words, what we are saying is that probably good chunks of the monitor code have this built in constant of 32 and are based, worse yet, on the assumption of the word and its bit array as the underlying structure to keep track of priority. This has two bad consequences, one is that if, in the further evolution of the system, one were to need more than 32 connected (known) processes at any given time in the system, it would be very difficult, with this kind of system, to expand it to accomodate the new requirements. The other bad effect is that, were the system to be re-implemented onto a machine which has a different basic architecture, for instance a machine that were not word oriented, at least not 32 bit word oriented, then the system as such could not be transferred or ported to the new machine architecture. We fully understand that the RTOS monitor is not very portable because it was implemented in assembly language anyway, so this argument could be accused of being specious.

The key point, in all this, is that one of the fundamental advances of computer sciences is that the details of lower layers data structures should not be visible at higher layers of the system. This advance is very fundamental because it is the key to the portability of the higher levels with respect to the changes in the lower layers. Or what is the same, stated in the converse

fashion, it is the key to the freedom of redesign and implementation of the lower layers without unduly impacting the higher layers.

This fundamental principle of computer sciences is deeply violated by the notion of using a data structure of the hardware machine as the basic underpinning of the architecture of the process management layer of the operating system. In fact, what one is doing is using the computer word, which is the data structure of the lowest layer of the system (the hardware machine), with all its gory details of number bits and sequence of bits, mapping priority into least significant, most significant, that sort of thing, and making all this extremely visible to a layer of the operating (the process management), which is not even the lowest layer of the software, in fact the lowest layer being interrupt processing. So there is no question that this is a poor decision which will have, in future, significantly costly consequences.

Furthermore, we believe that the rationale behind this particular choice was eminently wrong since, even though we fully realize that the preoccupation of the designers was to achieve a true real time operating system, experience with the design of many operating systems shows that it is rarely the actual execution of CPU instructions, executing scheduling algorithms, that gets in the way of satisfying operating system performance requirements or even real time scheduling deadlines. What usually gets in the way are waits for secondary resources such as copies of information from disk, access to channels, obtaining enough main memory, etc.

Further proof of the validity of what we are saying is the very modern design of the VAX VMS operating system which is designed to handle, in a quite satisfactory way, real time processes and absolutely shyed away from implementing schedulers in any kind of word mask oriented fashion. In fact, this very successful operating system implements scheduler queues in the classical way that most operating systems implement them, namely as treaded lists of process descriptors.

3.1 Memory Management.

The Memory Management of the RTOS system is a "real memory" management system for automatically handling overlays of application tasks.

The system is articulated around three primitives which are:

- Acquire.
- Release.
- Ovhoad.

The Acquire primitive is used by several classes of users to request, from the Memory Allocator, that a block of a given size be allocated in real memory for their use. The classes of users that can use the Acquire primitive are:

- Monitor.
- State executives.
- Any of a possible set of 32 privileged application tasks.

After customary checks and controls, the Acquire primitive obtains services from the Main Memory Allocator which in turn returns, if available, a block of memory that satisfies the specification in the request. An important point is that the Acquire primitive returns the absolute address of the memory block that has just been allocated.

The Release primitive performs a converse action, namely it can be called by the same users described under the Acquire primitive and it releases memory that these users may have acquired previously through an Acquire primitive. The Release primitive is capable of

either releasing an individual block or all the memory that belongs to that particular user.

The two above primitives are basically the fundamental mechanism for allocation of memory and the memory that is being allocated is primarily the user area. Ref. 2 on page 5-4, fig. 5.2-2 shows the overall memory map of the RTOS system. This map shows that the physical memory is divided into three portions, the area common to the monitor and the executive, the area reserved to the monitor, and an user area. The user area in turn is subdivided into an area dedicated to the state executive and an area that is reserved for the user tasks. Although the monitor will use these allocation mechanisms for allocating its own code, the primary use is really in allocating the user area, that is, the state executive plus application task areas.

Further insight on how specifically the memory management operates is given on page 6-34 of the ref. 2, fig. 6-9.3.2 which gives the memory layout for a typical task. This memory map indicates that the portion of the user area which is dedicated to the application task is further subdivided into a non-overlayable area and an overlayable area. The non-overlayable area, in turn, is divided into three portions, the first being the task activation record, the second being the task control tables, the third being the overlay zero or, that is, the root overlay. The overlayable area includes all of the other overlays.

Page 6-82 of ref. 2, fig. 6.9.3-1 gives a typical overlay structure for an application task. Basically, the point of the figure is to show that a task can build an arbitrary tree structure of overlays. That is, each overlay in the structure can in turn have any number of sons overlays which in turn can have sons overlays and so on and so forth.

The Ovload primitive is a primitive that fetches overlays and locates them or loads them into main memory. The mechanism through which allocation conflicts are completely eliminated so that the Ovload primitive can operate with no fear of resource contention is that when the task is scheduled, the scheduler requests a memory block large enough to include all of the task control tables, including the activation record, and the longest overlay path. The reason why this allocation strategy is pursued is that the way the Ovload primitive works is that it will not load the Nth overlay in a given path of an overlay tree unless the (N-1)th one overlay is already memory resident. Consequently, to guarantee absolute avoidance of memory contention, one must allocate a block of main memory that is large enough to contain all of the overlay down the path of the tree which will require the most memory.

This memory management scheme is extremely simple and rudimentary, and while having all the benefits of simplicity, it has the fundamental disadvantage that it could result in significant inefficiency in the utilization of main memory. However with the current trends towards packaging and cost of memory, memory efficiency should not be a major concern of a program of this type.

This scheme is adequate as long as the number of application tasks competing for main memory allocation is limited. If the system were to evolve in a direction where a large number of truly con-current tasks are competing for main memory allocation, it would be highly desirable to evolve the memory management towards a more truly virtual memory orientation.

3.5 Device and I/O Management.

The RTOS design does not make a very marked difference between the I/O management and file management. Basically the monitor provides the following I/O primitives:

- Locate.
- MDF/MTF Read/write.
- DIO.
- Print.
- PIPRINT.
- ICL Send/Receive.
- Keyboard/display.

In this section we will discuss most of these primitives except the Locate and Read/Write primitives which shall be discussed in Sec. 3.6. File Management.

The DIO primitive (device I/O) basically allows an application task to set up a channel program (DCWs). The point being that, obviously, the control of the DCWs has to be in the hands of the monitor for system integrity. However through the DIO interface the application task has functionally the same kind of control as a monitor program could have on a channel program.

The DIO interface is available for the following devices:

- MDF.
- System printer.
- Computer printer.
- MTF.

The DIO interface, by utilizing the Device Control Block (DCB) concept attains a degree of device independency with respect to the application tasks.

The print primitive provides low level printer I/O for both the system printer (HADL alarms, FCS alarms, test results) and the computer printer which is used for normal printing usage.

The keyboard/display primitive appears to provide traditional I/O support for a keyboard display complex.

The ICL Send/Receive primitives appear to be specialized I/O services to allow two basic kinds of high speed transfers. The first transfer is swapping arbitrary amounts of memory content from the user task space to the global disk space and vice versa. The other specialized transfer is the copying of a maximum of ten words of ICL buffer space out of the Monitor Executive Common Communication Area (MECCA) in or out of the user task space.

In balance the I/O services provided by the monitor are pretty standard low level I/O interfaces to those devices which are managed by the monitor so that task executives or application tasks can gain access to the full low level functionality of the device without compromising the integrity of the resource which remains controlled by the monitor.

3.6 File Management.

The file system of the RTOS is based on the file directory which appears to be the collection of the File Control Blocks (FCB).

Page B-4 of Ref. 2 indicates that there are two directories, a global file directory and a local file directory. The local file directory is specific to each system state. At any given time the file directory is really a combination of the local state directory with the global directory.

A footnote on this page indicates that some system programs or FCBs are in Moncom and that is, they are not really normal FCBs, but special ones. The special FCBs are for the Task Termination Controller (TTC) and the Hardware Alarm Detection Logic (HADL) which need these quasi FCBs for scheduling them as processes.

Normally these kinds of solutions of having pre-defined objects which are handled in special ways are indicators that the architecture of the system has not been thought out correctly for limiting conditions. In other words, we consider the existence of these special FCBs and the special mechanisms for handling them as an indicator of potential problems with the monitor.

From the memory maps of the monitor and the monitor plus the executive given on page 13 of appendix D of Ref. 2, it appears that the global directory and the local directory are resident, respectively, in monitor and executive memory. This type of design would be justified

if only few files are involved and certainly would have the advantage of cutting down of disk accesses. However, as all these kinds of choices, it would get in the way of future expansion of the system.

The file directory capability present in RTOS introduces the notion of logical file names, so that it makes application programs invariant under the reconfiguration of the disk, in other words, the introduction of a rudimentary file directory capability prevents having to recompile the application programs every time the files are moved around on the disk.

In the case of RTOS it is not quite possible to make a very clear distinction between file management and I/O or device management. However, we will discuss in this section only those primitives which can be more normally equated with file management. The monitor includes the following I/O primitives:

- Locate.
- MDF/MTF Read/write.
- DIO
- Print.
- PIPRINT
- ICL Send/receive.
- Keyboard/display

In this section we will discuss the Locate, and the Read/Write primitives, since they are the only ones that more properly could be considered related to file management.

The Locate primitive searches the file directory utilizing a logical name and returns the address of the FCB corresponding to that file name, if it is available at all.

The Read/Write primitives treat the MTF as an extension of the MDF, in fact the same file directory is utilized for both devices. All operations are file name oriented. Essentially what the Read/Write primitives offer is a sub-set of the natural Read/Write operations of the respective devices.

In other words, in this file system there is no record abstraction. One could say, in terms of modern nomenclature, that it implements byte stream files. The operations are very low level and the only real abstraction that is involved is introduction of a file name orientation to the operation so that recompilation of application programs can be avoided in restructuring mass storage.

Furthermore, the Locate does not "open" a file since the Read/Write primitives also search for the FCB corresponding to a particular file name on the file directory.

In balance, this file management system is extremely primitive and not very different from a standard low level I/O control system. Just about the only thing it introduces is the notion of symbolic file name or a logical file name.

3.7 Reliability, Availability, Maintenance

Sec. 5.6 Fault and Recovery Processing (Pg. 5.27 of Ref. 2) illustrates the basic philosophy for fault detection and recovery for the RTOS System. The basic philosophy can be summarized by saying that the design stresses the importance of detecting very early and in a very comprehensive way, all the faults that occur, however, the monitor does not attempt any recovery actions whatsoever and recovery is considered the responsibility of either the operator or specific task executives.

The basic design philosophy in this critical area of the system is best captured by quoting verbatim from the reference mentioned above.

"For all fault interrupts which have not been tailored by the executive, the BP will be halted, thus preventing the completion of a critical sequence with the existence of a fault. Software detected faults may or may not halt the BP, depending upon the seriousness of the offense. It is important to note here that fault processing is a tailorable feature, permitting executives to by-pass the standard monitor fault processing if they so desire.

Elements of the philosophy consist of the following:

- a. The OS design includes modules which vigorously attempt to detect faults as early in the sequence as possible.

- b. To facilitate fault analysis at GEOS and NSWC/DL, sufficient information about the environment of the fault will be gathered in a central memory area, such that the fault may be easily traced to its source. This information, called the fault signature, is not intended for use by the fire control operator, but instead should be captured on a tape cartridge via the DATALOG feature of the maintenance panel whenever a computer stoppage occurs, and returned to NSWC/DL or GEOS for analysis purposes.
- c. As a standard philosophy, automatic recording of the fault signature data on a mass memory device (MDF or MTF) will not be performed.
- d. Retries are attempted on certain I/O devices. The number, N, of retries is a device dependent characteristic and may be input at system generation time. The device is considered failed only after N successive unsuccessful retries.
- e. After a fault has been processed, the OS does not initiate any automatic recovery sequence, but instead permits the operator to initiate any recovery deemed necessary, probably as written in the standard operating procedures (SOPS).

f. Requests for monitor services are checked for validity.

Invalid requests will result in the requestor being purged. Valid I/O requests which generate device failures will result in the requestor being informed (via posting) of this condition.

g. All CPU fault interrupts are catastrophic. If such faults occur, the monitor will (after collecting the fault signature in a central memory area) halt all system activity. In this case, the operator will be forced to perform a bootstrap operation.

Realizing that executives are a part of the OS, and furthermore, that fault processing is a tailorable feature, executives may substitute special executive fault processing routines and thus by-pass the standard OS fault processing philosophy. In such cases, executives may provide for recording fault signatures on mass memory, initiating an automatic recovery sequence, etc. This is the only case where the standard OS philosophy may be circumvented."

Pg. 6-187 of Ref. 2 and many other places before and after, indicate that any scheduler, I/O, and other OS primitives, upon discovering an error condition, will call a routine named BAD\$FPT. It is evident, therefore, that BAD\$FPT is a common error handling routine for the monitor.

Putting this element of the design together with the philosophy stated above, one has to conclude that BAD\$EPT is the agent that signals to the caller of the primitive, which would have better intelligence on how to handle the error condition, the existence of an error/abnormal condition.

We interpret the above design philosophy as being motivated by a desire to guarantee that the system is always in positive control condition. That is, our interpretation is that specific executives are allowed to take up the automatic fault handling and recovery since specific executives may be carrying out non-critical portions of the mission while, of course, other executives, primarily those of the tactical mode, cannot be so allowed. If the automatic error handling and recovery were to be imbedded into the monitor, then all executives and therefore, all modes of the FCS system would proceed with automatic recovery. By architecturally assigning the task of recovery to the executives, one can guarantee that certain kinds of recovery situations are handled strictly through human intervention.

It so happens that this particular philosophy, which obviously was dictated by the special mission of the FCS, dovetails with current thinking about how error handling should be architected. In fact, modern thinking about the architecture of operating systems is increasingly affirming the point that the inner layers of the operating system (in this case, the monitor) have only the function of detecting errors and signalling to higher

callers the condition. It is the higher level callers, which have knowledge of the context, which are allowed to do automatic recovery or reconfiguration or whatever.

3.8 References

- 1) Trident I FCS RTOS Monitor Design Disclosure Document,
NAVSEA OD 45601 Vol. 2, Part 1.

- 2) Trident I FCS RTOS Monitor Design Disclosure Document,
NAVSEA OD 45601 Vol. 2, Part 1, January 15, 1978 vs.

4.0 The TDCC Architecture

The Trident Digital Control Computer (TDCC) architecture represents a conventional approach to providing general purpose computing power to the Trident II Fire Control System (FCS). Whereas it possesses some architectural attributes which facilitate real time processing, it has not been specifically tailored to FCS functions.

The MK 98 MOD 0 is defined to include the Central Processing Unit (CPU), its main memory, a communications controller and the multiplexor portion of the Data Communications Processor (DCP).

In general, it is a sequential, register oriented, thirty-two bit word machine. It possesses multiple interrupt levels (forty-eight in all), dual state (executive and task) instructions and registers, instruction look ahead and some degree of micro programming. Of these characteristics the only unusual item is the duplication of both general purpose registers (16 per group) and the base/protection registers (32 per group). These registers are separated into two groups: the task and executive. This distinction was purportedly, made in order to simplify processing by allowing access without cross-state register contention. There are some registers which permit either state access and thus facilitate executive-task communication.

Conventional addressing schemas are utilized through a combination of real and base register offset addressing. All addressing is on word boundaries with partial word offset capabilities (quarter word, half word, full word, double word, variable length). Maximum real addressability is 256K words (due to the 18 bit address field). There is no distinction between instruction and data specific addressing. An operand only stack capability is also present. Multiple stacks are permitted and may be placed at any point in memory. The stack technique implemented is a simple one which, although it has all necessary stack instructions (PUSH, PUSH DOUBLE, POP, POP Double, LOAD from Stack, Store to Stack, Load Double from Stack, Store Double to Stack and modify Stack Description), makes no attempt to speed execution by placing top stack elements in cache, registers or other faster than normal access memory.

Also present in the addressing schema are multiple and indirect addressing and indexed (offset) with base register addressing. Virtual memory capabilities were also described, (64 K words segmented into 16, 4 K word sections), however, their actual method of use, or benefits were not clearly discussed.

Areas of interest for possible future enhancements include re-implementation of MK98 with newer technology, enhanced stack capabilities, more optimized addressing schemas, and use of specialized processors. We would like to briefly discuss

some ideas with regards to methods by which the MK98 FCS could be enhanced without catastrophic effects on existing software systems. The basic premise is that although the MK98 is an architecturally adequate CPU, some areas could be optimized without significantly altering the FCS as a whole.

Processor re-implementation is probably the simplest option available. The MK98 has been benchmarked at between 129 and 235 KIP's depending on the actual instruction mix being executed. It is probable that, with new technology alone these performance factors could be at least doubled and very likely tripled. These factors could be further enhanced by some efforts to optimize instruction operator codes with regards to frequency of occurrence and machine representation. However, this complicates implementation by requiring recompilation or possibly translation of current software.

Stack operations could also be enhanced by implementing top level stack positions in high speed cache memory. Current memory access times were stated to be 540 nsecs (word or byte, memory cycle time or real access time, read or write?), with current technology this time can certainly be decreased to less than 200 nsecs or better, with such an approach. This option requires more effort to implement so its desirability depends directly on degree of stack usage.

The need to increase machine memory capabilities was also mentioned in several documents which GSG received. Throughout the discussion of memory expansion was the thread of minimum effects on operational software. An option mentioned but never really discussed was the distinction between instruction and data address spaces (I and D space). This distinction was never really discussed in the documents received but nevertheless represents a very practical method by which machine addressing may be optimized to either maintain current addressing limits with decreased address field size, or to maintain address field size and increase total memory which may be accessed.

Finally, the use of specialized processors is another way by which the MK 98 may be relieved of some rather significant, and in some cases very specialized, processing requirements. These options were mentioned to some degree in papers discussing the "missilized" versus the "phased" approach to the IAP. The use of specialized processors makes very good sense, particularly in the areas of array processing, for applications such as the FCS. However, the degree of effectiveness is directly affected by the method of implementation. A key example is the method of data communication. In other words, does the specialized processor have Direct Memory Access (DMA) privileges or does it have to process data through an intermediary communications

controller? Such considerations are and must be addressed if the final results are to be the best possible.

5.0 Implementation Tools

5.1 Characteristics of the Trident Higher Level Language (THLL)

The user's guide provided by NSWC provides substantial detail concerning the THLL implementation currently utilized by SP-23 applications. THLL itself is a block structured, procedure oriented language. Data typing is extensive and once defined is not runtime changeable within a specific procedure. The THLL compiler is a four pass implementation which produces directly executable MK98 and MK88 FCS-DCC (Fire Control System - Digital Control Computer) direct code.

THLL possesses very flexible execution control facilities (such as GOTO, SWITCH and CASE statements) however, NSWC internal policy directs that structured programming techniques be used wherever possible. Language facilities necessary for mathematical FCS solutions are provided in abundance through both the standard arithmetic operators and extensive built-in functions including specific macros to facilitate Cartesian and Polar coordinate geometries. THLL provides the capability to define multiple LIFO operand stacks of differing types along with the minimum instruction set necessary for their use. Although only minimal character manipulation facilities are provided this is not deemed to represent a deficiency for fire control applications.

As mentioned, data typing of variables is mandatory for the THLL compiler. Six discrete data types are available for procedure use (half, integer, double, real, pointer, and alpha). A seventh type (no type) is used for valueless items such as statements. Once a variable has been typed, its type may not be runtime modified within a procedure. Strong data typing protocols are established for the results returned by all operators (assignment type conversion tables are found in Appendix B of the THLL user's guide). Procedure typing is also used to control the types of values returned by called procedures. Untyped procedures indicate no value is to be returned. Although the data typing is not so strong as that of PASCAL, we believe it is more than sufficient for SP-23 purposes.

Strict operator precedence is enforced with addressing operators having the highest precedence immediately followed by binary operators. Normal mathematic, logic, and assignment operator precedences follow these. As in other HOL compilers, normal order of execution is left to right, in accordance with specific operator precedences. This order may be modified through the conventional use of parenthesis.

THLL provides visibility of RTOS services through a series of "service procedures". Capabilities provided include: Attach a task to the multitask set (MTS); Schedule a task for execution; Purge a task; Detach (remove) a task from the

MTS; Exit (terminate) program execution; various program wait options; continue execution (privileged instructions); multiple interrupt control options; overlay controls; multiple procedure control options; system error handler, and data block summation. We detected no deficiencies in the RTOS options available for applications task use.

In summary THLL appears to be competent for the specific task it was designed to satisfy. It satisfies the basic MOL definition of one source code instruction yielding more than one object code instruction. It has many of the language characteristics and conventions of ALGOL while also permitting users to directly manipulate memory locations at the lowest level (i.e. bit manipulation of a specific real or virtual location). Instruction repertoire necessary for efficient output formatting and character string manipulations are deficient in a general sense, however, they appear satisfactory for the task at hand. Flexible program sequence controls are available and although their use is restricted by internal convention, embody the most desirable features of other languages such as FORTRAN and ALGOL.

The only area of concern is the direct ability of programmers to access real machine addresses and bit manipulate data. Such abilities while they enable more efficient memory use will tend to increase the complexity, development and maintenance costs of FCS software. If the bit manipulation and real address capabilities are extensively used the cost savings

realized by requiring structured programming techniques may be significantly offset. Such usage will tend to decrease program maintainability somewhat as well as increasing the time necessary to train new programming personnel.

5.2 THLL Translatability to PASCAL/ADA

This section addresses the translatability of THLL based software into other HOL languages, in this case PASCAL and ADA. Compiler availability, although critical in real terms, is not considered at this point. Rather we are simply discussing the similarities and potential difficulties to be faced if and when a decision is made to move away from THLL.

While, today, such a move within SP-23 is not likely, other organizations, some of which are Navy, have found that internal support of systems support software such as communications systems and compilers has proven too costly to sustain. It is true that such internal support yields specialized capabilities, but there is a definite price to be paid. This price is realized in terms of organizational resources dedicated to development and maintenance of existing capabilities, develop new capabilities and most importantly to train new personnel. An additional, but not always obvious side effect, is the inability of internally supported organizations to rapidly capitalize on new technology. This inability will often result from available personnel assets being consumed in day to day evolutions - thereby making them unavailable to sufficiently analyze new technologies.

For all of the reasons mentioned above GSG feels that it is appropriate to discuss, at a general level, the feasibility of THLL translation. This discussion concentrates on such areas as the basic context and logical structure of the languages (is it block structured), available data types, basic operators, available macros or functions (e.g. trigonometric functions), data precision, degree of programmer separation from the physical machine, the ability to interact with the operating system etc. All of these areas must be addressed if one is to reasonably assess the feasibility of a language to language translation.

With regards to the first of these points - the basic context and logical structures of the three languages - the common ALGOL ancestry proves beneficial. All three follow the same structural concepts, consistency of data typing within procedures, clear beginning and ending of blocks, recursive ability, locality of data values etc. Thus, an attempt to convert THLL to ADA or PASCAL is not likely to encounter significant difficulties in these areas.

Although the locality of data type definition is consistent among all three languages, the actual availability of data types is not. THLL, for example, possesses seven types - half, integer, double, real, pointer, alpha, and no type. Standard PASCAL on the other hand has only four actual types - real, integer, boolean and character, a fifth type (user defined)

is available but only allows type definition and assignment of allowable values. ADA is more nearly in line with THLL by allowing character, boolean, integer, real (with variable precision - there is no double precision per se), array, records and access types. Thus, with regards to data type availability, ADA is easily compatible, PASCAL (in its common form) may cause difficulties with regards to unavailability of double - precision and the inability to specify digits of precision. Another deficiency of PASCAL, when compared to THLL is the lack of a pointer capability. ADA possesses an "address-specification" feature which provides a capability similar to the pointer of THLL.

Built-in function availability for PASCAL includes the standard sine, cos, cosine, square, square root, absolute value, arctan, exponential, natural logs and others. ADA prefers to rely on the availability of specialized libraries or programmer defined programs for these functions. Therefore, for an ADA implementation FCS required library functions such as sine, cosine, arctan etc. would have to be provided in some manner. They would not appear as normally supplied ADA intrinsics.

Programmer separation from the physical machine refers to the ability of a programmer to address specific real machine addresses and to "bit twiddle". Both THLL and ADA support these

facilities at about the same level, common PASCAL does not directly support either capability. A conversion from THLL to ADA is feasible from this point-of-view, however, some recoding would be necessary to reflect type specification differences. This statement does not, however, address the desirability of allowing programmers to gain physical machine access. In general, it may be said that such accessibility is undesirable from a system maintenance point-of-view, unless a strong performance case can be made for essential functions.

A final area of significant concern is the one of language interaction with the operating system. Both THLL and ADA provide such a capability through interrupt and exception handling definition features, PASCAL does not normally provide such a capability.

For an operating environment such as encountered by SP-23 applications, this is viewed as the most significant, and in this case a fatal deficiency. Such flexibility is mandatory, we feel, for a real time, stand alone operation such as the FCS environment.

In summary, although all three languages - THLL, ADA and PASCAL share a common ancestry (ALGOL), they differ in their current implementation. THLL maintained all the capabilities necessary for its target applications and while not ignoring other functionalities such as character or string manipulation, did not provide the same degree of

flexibility. ADA on the other hand is a more general purpose language which provides all the flexibilities of THLL plus additional enhancements which would provide little benefit to SP-23 applications. Significantly absent from ADA, however, are the mathematic and trigonometric built-in functions of THLL and PASCAL. These must be provided externally from ADA itself.

If a translation from THLL becomes necessary, then in GSG's opinion, it is feasible with some normal conversion difficulties, to translate THLL source to ADA source. It would not, however, be feasible to perform such a transition from THLL to PASCAL due to data type and precision, physical machine access and operating system interface inconsistencies.

One final note is that these comparisons are based on preliminary ADA specifications as published in June, 1979. ADA specifications are subject to review and redefinition until such time as the final reports are issued.

6.0 Software Development Methodology

6.1 Software Development Life Cycles

The backbone of the software development methodology of the FCS project is a life cycle management system broken up into discrete phases with corresponding baselines. In other words, the backbone of the methodology reflects state-of-the-art concepts on the management of software projects. The specific phase method utilized by this project has the following phases:

- System Definition: During this phase, functional and performance requirements are generated and documented.
- System Design: During this phase the actual design specifications are generated.
- System Implementation: During this phase the software is actually programmed and a detailed design results in a number of design documents.
- System Test: During this phase the actual debugging of the system takes place. Documentation produced during this phase consists of test plans and procedures.
- System Deployment: This is the phase in which the system is installed in the fleet. Documentation produced are the fleet documents.

The life cycle management scheme used by the FCS anticipates that the V&V activities are performed throughout the last three phases above.

The V&V activities, in themselves, have a flexible approach to recycling information relating to problems discovered in the software. They are based on classical, well proven approaches, such as controlled release of patches for short-term fixes, to be followed by long-term design changes or implementation changes, which are brought about by the system developers, and are authorized and approved by a Change Control Board. An interesting concept that is utilized in this program, to guarantee the accuracy of computational results, is the notion of having a specific critical algorithm implemented twice by two distinct set of people coding it for two distinct computers and in distinct languages. In one case the algorithm is coded for the CDC 6700 and in the other case, it is coded for the Fire Control Computer, that is the TDCC. The two versions of the algorithms generate results which are compared via a compare program. If the results are equal then this is a very strong validation of the functionality of the algorithm. Clearly, extreme measures, such as these, are required because of the obvious criticality of the mission of this weapon system.

The methodology also envisions a system for generating trouble and failure reports (TFRs) and a process to collect the TFRs

from the fleet with a centralized processing of the TFRs at Corona del Mar. The central point in Corona del Mar, after analysis and synthesis, feeds back the important TFRs to the developers so that the problem can be diagnosed and a correction be generated.

In summary, we can say that the life cycle, baseline, and V&V processes that are adopted by this program are comparable to some of the best in the computer industry, with an actual added emphasis on V&V activities because of the obvious critical nature of the mission of this subsystem.

At present the entire management process is basically in the maintenance mode, characterized by a basic development cycle of approximately 18 months, and a delivery of a new release to the fleet every twelve months. The management process is also capable of handling more than one release at any given time, so that, the functional specification phase of one release can overlap with the implementation phase of another, with the V&V activities of yet another release overlapping with both.

Changes and release planning are both controlled by a Change Control Board headed by SP-23 so that formal configuration management control is exerted on this process. At the present time, most of this activity is concentrated in changes to the application programs and only occasionally

in correcting bugs in the monitor and executives. This, of course, would change drastically if a modernization program were to get under way.

6.2 Programming/Design Practices

A good documentation of the actual practices used in the programming and design of the Fire Control System Application Tasks, is given by the manual NAVSEA OD46099 "Trident 1 and Trident 1 Backfit Fire Control System Software Programming Guidelines". The structure of this document is as follows:

- Section I Introduction
- Section II references to all other relevant documentation
- Section III provides an overview of the Fire Control System
- Section IV presents the guidelines for programming and designing executives
- Section V gives guidelines for the programming and designing application software or application tasks
- Section VI gives the guidelines for I/O
- Section VII presents conventions of the methodological type, such as top-down structured programming, top-down development, etc. That is, Section VII is the more traditional programming guidelines material.

To give the flavor of the guidelines for writing executives, we will quote a few below.

Page 4 - 19 Section 4.2.2.2.5 Processors for Memory Assignment

"When executives require that a buffer be allocated dynamically, they must communicate this request via the memory assignment processors. The monitor will call the appropriate allocation/deallocation modules to honor the request. Direct executive communication with the memory allocation/deallocation modules is prohibited."

Page 4 - 7 Section 4.3.1.2.3 Waite Continue Functions

"The Waite function should never be used from an interrupt routine. In the case of a Waite executive action, the executive is responsible for the corresponding Continue function and causes the transition from the active/disabled substate to the active/enabled."

Page 4 - 43 " A. An interrupt routine can never suspend itself by using the Waite macro and can never use I/O system calls such as read/write etc., with the Waite option."

Page 4 - 45 "Certain generalities about the nature of all executives, however, can be made. First, each one must have a "root" segment. (See Figure 4-9.) As with the root segment of the monitor, the root segment for every state executive must include: (A) Executive common, (B) The code and data for any interrupt routines that are to be directly connected to the DICS. The root of the executive can only be assembled by using any EBR within

the range of four to seven. (See Table 4.1.) Once the EBRs for the root of an executive have been established, they cannot be used to span any portion of any other segment."

Section 4.4.9.1 Stacks. "No program, monitor, executive, or application task should write data on the executive stacks. These stacks are used for linkage information only."

Page 4 - 47 "When generating the loader load module, it is desirable to logically concatenate the control tables of all state executives to the control tables of the monitor. Since the virtual origin for the monitor's control tables is BR-13, displacement 0, the virtual origin for the control tables for all state executives will be BR-13, displacement ALPHA, where ALPHA is the length of the monitor's control tables."

A few examples of guidelines for the writing of the application tasks follow.

Page 5 - 6 Section 5.2.3. Monitor Task Communication

"A task desiring services from the monitor (or from the executive) must use an EES Call. These Calls will be "procedures" described in the THLL documentation."

Page 5 - 8 Section 5.2.4.3 Transferring Control to other Tasks

"A task executing under a state executive will not be allowed to jump directly to another task operating under the same executive. It will be required to return control to the state executives via the monitor, which in turn, will relinquish control to the second task."

As it can be seen, these guidelines are essentially prescriptions on how to properly use the services of the monitor in writing executives and of the monitor and the executives in writing application tasks. It is not clear from this documentation whether these rules are stated, because the rules are enforced through a programming discipline, or they are rules which are enforced by the system design itself and are listed here as a reminder to programmers. In other words, what we are saying are that a number of these rules could easily be implemented as illegal conditions in a suitable abstract machine of the system. We are not clear on whether this has been done or one is relying largely on programming conventions to impose them.

Section VII software standards and conventions starts with a very nice description of the software development environment and tools in this program.

In Section 7.1.2.1 Top Down Development Testing, the manual explains that the project is committed to the use of standard top-down development method and furthermore, it indicates that it, indeed, is top-down development as opposed to top-down

design, in fact, it says: "the top-down approach to test and integration, parallels that of the development process, and, in fact, overlaps this process. Testing is a continuous integration of a consistently growing program. The first step involves execution of the control structure only, and then building the program in the sequence previously defined".

As it is well known, this is by far, the best way to develop and test programs in accordance with both theoretical insights and extensive positive practical experience.

Section 7.1.3 Structured Programming Requirements briefly explains the whole concept of structured programming methodology and makes it a required approach for this program.

Section 7.1.5 Trident Program Design Languages (TPDL), states the following: "The logic for a program will be represented in TPDL. For Trident software, the language is structured English with six allowable control structures. (By the way, these are the very same six control structures which are allowed in the structured coding of the actual programs.) Indentations are required to show the various levels of control. To show the domain of a structure, three additional reserved words are used: They are ENDIF, ENDDO, and ENDCASE.

Statements inserted between the IF and ELSE, ELSE and ENDIF, DO and ENDDO, and CASE and ENDCASE must be so indented that the domain is visually recognizable."

What is most significant of this approach is the fact that the control structures are exactly the same as that of the programming language. This implies that the outer syntax of the Program Definition Language is the same as that of the programming language with the inner syntax being an informal English like language. The advantage of this approach is that the design can very smoothly evolve through refinements into the actual programming.

Another interesting point is made on Page 7 - 14 in connection with guidelines for top-down/structured design. One of the guidelines is: "(1) Partition the system into functions and organize related functions into level based on system usage and services provided; attempt to follow LISKOV level of abstraction and partitioning rules." This is interesting because it shows that the project is aware of the notions introduced by LISKOV on level of abstraction which are key to the proper understanding and formulation of architectural design of complex software systems. However, the rest of the documentation, we have studied, shows no real distinction between the notion of structured programming and that of architectural design, so it might be that the understanding of the LISKOV notions is not sufficiently profound.

In summary, these guidelines seem to be quite good, they are basically a set of reference informations that allow the programmer to design correctly executives and application tasks, and a set of methodological prescriptions which, basically, recommend the use of all the latest methodology.

6.3 Development Environment

By development environment we mean the run-time support of the programming language as well as all of the other tools which are required for the generation of programs, such as, debuggers, linkers, loaders, editors, source code control systems, etc.

In determining how many development environments are being utilized by a project, one must find out how many implementation languages are actually being used and how many machine/operating systems are being used as a run-time support environment for the various sets of tools. In the case of this program, it appears that the monitor, all of the GEOS developed test programs, and the ASW program were developed in assembly language because the THLL was not available at the time. The rest of the software is developed in THLL. Thus, from a system implementation language viewpoint, one would expect two basic development environments; one pegged for the THLL and the other for the assembler for the TDCC.

The document OD45986 "Trident Software Development Plan", while being out of date from the point-of-view of a software development plan, sheds considerable light on the issue of the development environments being used in this program.

Section 2.1.1 of this document indicates that the host computers that have to be considered include the CDC/6700, G/635, and XDS SIGMA 5. It further goes on to indicate that there is a variety of programs which will be able to execute on all of the host computers and that they are all characterized by being written in a subset of the ANSI standard FORTRAN IV. Amongst these are included the following:

- TASS: Trident Assembly & Simulation System (6700)

This system includes the following elements:

- TASS Executive
 - Control Card Pre-processor
 - File Editor
 - Assembler
 - Linker
 - Instruction Simulator
- TASS (635) This system includes the same list of sub-systems as the CDC/6700 version.
- TASS (SIGMA 5) This system includes the same list of sub-systems as above.

In other words, what we seem to have here, is a portable developer environment TASS written in a subset of the ANSI standard FORTRAN IV and rehosted in the following computers: CDC/6700, GE/635, and XDS SIGMA 5.

Section 2.1.2 of the same document indicates that there is a version of the Trident Assembly System (TAS) which is written in Trident Assembly Language and that this particular system has exactly the same list of subsystems as the TASS stated above except for the simulator.

The entire collection of Trident Assembly Language Programs which basically form the TDCC hosted development environment is as follows:

- Check-out Monitor
- On-line Debugger
- TAS
- Trident Loader
- TDCC TAS Accounting Program
- SYSGEN
- TDCC Support Software Quality Test
- Utility Package

The conclusion we are driven to from this information is that the complete TDCC assembly language development environment is replicated in two forms; one is a set of subsystems written in FORTRAN IV, and the other is a set of subsystems written in the TDCC assembly language itself.

The first is implemented in a rehostable implementation that has been rehosted to the three machines mentioned before. The last implementation is, of course, specific to the TDCO and allows the development of programs right on to the TDCO, obviously, the TDCO version does not require an instruction simulator since the machine itself provides that capability.

The information available to us does not indicate how THLL is introduced into this basic development environment. We presume that the THLL compiler outputs compiled units which are processable by the same linker and loader of the TASS system. We, also, presume that the THLL compiler is written in THLL itself and, therefore, it is bootstrappable under a number of environments. However, these are suppositions on our part, we do not have any direct evidence of their validity. If they are true then the fundamental development environment TASS and TAS would be easily extended to include the THLL development environment.

7.0 TDCC Software States and the Operational Sequence

The Real Time operating System (RTOS) of the TDCC consists (at any one time) of the Monitor program and one of eight unique task Executive programs. The Monitor and three of the Executives are written and maintained by the Naval Surface Weapons Center (NSWC), Dahlgren, Virginia. The remaining five Executives are written and maintained by General Electric Ordnance Systems (GEOS), Pittsfield, Mass. (A ninth Executive, used for ASW Torpedo Fire Control, was written by Librascope, but is not applicable to the Trident PCS). The combinations of the monitor and each of the eight executives define the eight operating systems, also called states. The names of the executives and corresponding operating systems are identical:

<u>Executive Name</u>	<u>Maintenance Organization</u>
1. Data Entry	NSWC
2. Standby	NSWC
3. Launch (or Tactical)	NSWC
4. Test and Evaluation (T&E)	GEOS
5. Fire Control Test (FCT)	GEOS
6. Missile Test (MSI)	GEOS
7. Fuze Test	GEOS
8. Training Without Guidance (DUMMY/TWOG/TWOGI)	GEOS

The first three operating systems will be described first, as they relate to fire control modes which are used at sea to directly implement the Trident F/C mission.

When the submarine is at sea, in a patrol area, an increased readiness condition is set, known as Condition 2SQ. This condition implies that the submarine can achieve a battle-ready condition (1SQ) in a relatively short time. At 1SQ the submarine is ready to launch (or simulate the launch) of its missiles as soon as the Commanding Officer's permission-to-fire-switch is thrown.

At condition 2SQ, both TDCC's are normally supporting the Standby F/C mode using the Standby Executive. In this mode, computational programs (including 9 unique task programs) are executed. These routines are described below.

During condition 2SQ, one of the TDCC's can be switched by the operator to activate the Data Entry F/C mode using the Data Entry Executive. In this mode data processing programs (including three unique task programs) are executed to allow the operator to enter mission information (targeting, geophysical, system, and missile data) into the FCS and to insure that the correct data is stored on both MDFs. The main functions of the data processing routines are performed in the following order:

1. Operator-initiated input of data (mission information) from data sheets and magnetic tape cartridges. Input to the computer is via the keyboard and MTFSS, respectively.
2. Writing and storage of data on the MDF connected to the DCC that is processing data entries.
3. Transfer of data to the secondary DCC via the ICL and storage of data on the secondary MDF.
4. Verification of entered data by means of automatic and operator-requested printouts on the computer printer.

During the data processing state, the operator is also allowed to modify or delete previously entered mission information.

Operator actions associated with the data processing state are taken in accordance with standard operating procedures (SOP's). During performance of these procedures, the operator must ensure that both disk packs installed on the MDF's are compatible (i.e. both are the same software revision and targeting revision).

During Standby Mode, the standby operating system and standby program are loaded into DCC main memory and are executed. The main routines of this mode are a computational initialization routine, a range check routine, trajectory simulation routines, and an onboard footprinter (OFT). These routines perform the following:

- Prepare for target package computations.
- Compute the simulated flight of all the missiles in each of the designated target packages to their assigned footprints (burst point patterns).
- Determine if each footprint is achievable from the current ship position.
- Generate footprints from data entered during data entry mode.

In order to achieve battle condition 1SQ in the submarine, many conditions are changed, among them:

- The submarine crew is called to man their battle stations.
- The submarine itself is brought to launch depth and special ballast tanks are readied for missile launch conditions.
- The ship's nuclear propulsion and electrical systems are reconfigured for maximum reliability.
- The missile firing message is decoded and checked by at least two people.
- The launch tubes are pressurized and the missile gyros are energized and stabilized.

While all of the above is occurring, the Launch (or Tactical) operating system is loaded into one (or both) TDCC's and one of three possible F/C modes are selected:

- Tactical (TACT) mode for firing actual missiles.
- Training with Guidance (TWIG) mode for simulated launch.
- Training with Guidance and Launcher (TWICL) mode for simulated launch (MK98 FCS only).

The launch sequence, whether normal or simulated, is divided into two major phases or conditions of readiness. The first phase is from 2SQ to 1SQ, and the second phase is from 1SQ to Missile Away. Each phase consists of a set of routines (including 10 tasks) that execute together to prepare and launch missiles. These routines are:

- Transition computations
- Stellar selection
- Platform positioning initial velocity computations
- Alignment, erection, and elevation check routine
- Denote routines

The transition computations develop targeting offsets, time of flight, etc., required for each missile assigned to a footprint in the active target package.

The stellar selection routine selects a star for each missile that is assigned to a footprint in the active target package. Once selected, this star is available for a predetermined amount of time. At the end of this time, an alarm is displayed on the operator control panel and the missile may be restarted (2SQ to 1SQ) by reinitiating the guidance initialization (GI) sequence, which is referred to as re-GI, so that selection can be restarted.

The Platform Positioning Initial Velocity (PPIV) computations routine performs platform positioning computations, transfers the PPIV data and program to the guidance computer (GC) in the missile, and performs a check to ensure that the data was sent correctly.

The alignment, erection, and elevation check routine controls the transfer of navigation data to the missile guidance system (GS) and evaluates the performance of the guidance platform positioning sequence. A platform positioning complete signal is generated in the missile and the operator control panel is updated via the Task Control Bus (TCB) indicating the sequence is complete for that missile (FC ISQ).

The ISQ to missile away phase consists of sequencing a missile at a time through a series of denote and prepare routines.

During denote, the routines of the launch state perform transition computations, monitor the alignment, erection, and elevation process, and keep the entire weapon system updated through the launch executive interrupt routines.

During prepare, final launch preparations are completed which are time critical:

- Missile presettings (final target and position vectors) are sent to the GS.
- Fuze information is sent to the missile command sequencer.

- Loops open sets the GS inertial and transfers the GS to internal power.

The following is a description of the Test, Evaluation, and Training states, developed and maintained by GEOS, and which are not normally used at sea during patrol conditions.

The Test and Evaluation (T&E) state is used to support shipyard installation checkout and testing. There is no explicit F/C mode corresponding to this state.

The Fire Control Test, Missile Test, and Fuze Test states all execute test and data handling programs to verify that the fire control, missile, and fuze equipments are operational or to isolate detected failures in the equipment.

The DUMMY/TWOG/TWOGGL state provides an operator interface simulation. Operator inputs at the firing panel are processed, firing panel status indications are raised at the appropriate time, keyboard options are displayed, and errors in utilized equipment are detected with the appropriate alarm printed and/or indicated.

The following table summarizes the relationships between possible software executives, programs executed, and fire control modes supported:

STATE EXECUTIVES	PROGRAMS EXECUTED	FIRE CONTROL MODES
1. Data entry Exec	1. Data Processing Programs (3 tasks)	1. Data entry mode
2. Standby exec	2. Computational Programs (9 tasks)	2. Standby mode
3. Launch (or tactical) exec	3. Launch programs (10 tasks)	3. Tactical mode 4. Training w/ guidance (TWIG) 5. Training w/ guidance & launcher (TWIGL)
4. T&E exec	4. Test & Evaluation programs	-----
5. FCT test exec	5. Test programs and data handling programs	6. FC test mode 7. MSL test mode 8. FUZE test mode
6. MSL test exec		
7. FUZE test exec		
8. DUMMY/TWOG/TWOGL Exec	6. DUMMY/TWOG/TWOGL programs	9. DUMMY/TWOG/TWOGL modes

8.0 Additional Processing Requirements for the Applications

In this section of the report, we will examine the opportunities for extending the architecture of the TDCC to accommodate additional requirements. We will start the section by an extensive review of the many studies that have been performed by NSWC, GEOS, and HAC on this general subject.

The latter part of this section presents our views on how, in light of the available evidence, we would recommend evolving the architecture of the system.

The evidence available to us on the subject is somewhat limited. The limitation arising from two fundamental factors, the first being the obvious secrecy that surrounds this kind of program, the second being that the project community has been less than favorably inclined to meet in an open, round table mode of discussion, on this subject matter. What we are saying is not that the project team has deliberately avoided contact, quite the contrary, we were briefed on a number of occasions, however, there has been a feeling on the part of the project team that GSG could not really do much to help them and consequently, it was almost a waste of time to sit down and talk about difficult problems. We regret that this has happened, since our study of the available written evidence shows, in an undeniable way, that

the community supporting this program has limited computer system architecture capabilities, and that it has struggled, for over three years, with problems that a competent system architect could have solved in much less time. Furthermore, judging from the most recent documents available to us, the project is still headed in the wrong direction and because of that, it seems not to be able to make up its mind on which way it wants to go.

The information available to us is comprised of four set of minutes of the Trident II Fire Control Computer Throughput Working Group. They are the minutes of the meetings of the working group held on November 10, 1977, January 24, 1978, June 21, 1978, April 6, 1979. In addition, we have a study made by Hughes Aircraft Company (HAC) on options for memory addressability dated October 11, 1979. Finally, the last set of data was the one we obtained during the JSC Program Review in Dahlgren on June 10-11, 1980.

These sources of information, of course, are reviewed strictly from the point-of-view of what evidence they offer for increased requirements in either computational power, memory size and addressability, or mass storage performance and capacity. In other words, only for evidence that they may offer for the need for a revised architecture of the FCS system.

8.1 November 10, 1977 Throughput Working Group Meeting

This is the first set of minutes of the working group in our possession. It shows that most of the preoccupation of the group is centered around the computational requirements posed by the High Frequency Gravity (HFG) compensation. As we shall see, HFG dominates the discussion also of the other three sets of minutes.

During this meeting, three methods for performing the HFG calculations are discussed. They are the Stokes integral, the Continuity Integral, and the Poisson (upward method) technique. A very important point in these minutes is that it appears that this is the first time that the group was discussing the possibility of cutting down by an order of magnitude the number of vectors to be computed. In fact, evidence is being presented that something like 18 vectors would achieve acceptable accuracy. The reduction of the number of vectors is obtained by utilizing a spline fitting approximation scheme with only a limited number of data points, namely, the vectors to be computed. The minutes explicitly comment that the accuracy achieved is the same as that previously expected to require something like 100 vectors.

The minutes also reflect the understanding that improvements of the computation of the Waiting Matrix (W-Matrix), regardless what factors are chosen to be improved, imply no

significant increases of the throughput of the Basic Processor (BP). This further stresses the point that, as far as computational needs are concerned, the only problem that seems to be confronting the working group is HFG.

One thing that it is extremely interesting to us, as computer system architects, is that the minutes include a view graph (View Graph 5) which actually presents the instruction mixes which are exhibited by the different HFG computational methods. The reason why this is significant is that, from a computer system architecture viewpoint, this kind of detailed analysis is absurd. The reason why this level of accuracy is absurd, is that both the queuing nature of computer systems, as well as, the modularity with which these products are available, render any difference that is less than say 20 to 30% one way or the other, totally meaningless.

This together with similar other pieces of evidence indicate to us that the people who are involved in doing this kind of work, may know a lot about computers, but they know very little about system architecture. It looks to us more like a number of very skilled numerical analysts trying to become system architects. In fact, numerical analysts, as it is well known, invest a considerable amount of professional training

AD-A092 687

GENERAL SYSTEMS GROUP INC SALEM NH
REVIEW OF THE FIRE CONTROL SYSTEM.(U)

F/6 19/5

UNCLASSIFIED

NOV 80
GS60001Z

N00014-80-C-0104
NL

2 0 2
40
A. 9. 2000



END
DATE
FILMED
1 81
DTIC

in learning how to control the error accuracy of their algorithms to, sometimes, well less than 1% error. To carry over this kind of mentality to system architecture is, not only a waste of time, because a lot of work gets done that should not have been done, but it is actually harmful, because it causes the whole conceptual landscape of the problem to get cluttered with a lot of details which are totally irrelevant.

To substantiate our point in this specific case, one has only to refer to View Graph 5 to see that the differences between instruction mixes are absolutely negligible. In fact, the only significant differences are in the fixed point arithmetic and floating point arithmetic. In the case of fixed point arithmetic the point mass method appears to require three times the frequency of fixed point arithmetic instruction that either Stokes or Coating, on the other hand, the point mass method requires approximately 60% of the floating arithmetic instructions that either Stokes or Coating. Furthermore, Stokes and Coating, for all practical purposes, have exactly the same instruction mix. The other point is that, even these two significant differences, do not amount to much more than about 10% of the total instruction mix.

On View Graph 8-12, the computational times for the three methods are given for both the case of single missile

and for all 24 missiles. The computation times are given also for a six point spline and a five point spline. The key message that comes across from this slide, from a computer system viewpoint, is that the Stokes method requires approximately twice the computation of Coating which, in turn requires approximately 10 times the computation of the upward method. In other words, from a system architecture point-of-view, it is very clear from this data that the choice of the numerical algorithm affects the architecture in significant ways.

Next the meeting examined a throughput estimation study which presents a lot of numerical results in the form of performance curves. The study was done with a number of assumptions. The three key ones, from our point-of-view, being as follows:

- Workload to be the sum of the Trident I computational workload plus the Trident II unique, which is HFG.
- Similar software architecture to that of Trident I.
- HFG would be computed using the upward/point mass technique.

We would like to briefly summarize only a portion of the results because they are important to make a very specific point later on. The two sets of results we would like to summarize are the estimates of throughput (in Kips) for the case that the HFG

computation is done during the launch interval. For the purpose of illustration, we will assume that the launch interval is 20 seconds (end of the range of the curves). Similar results for the case in which the HFG computation is done during the transition interval, again, for illustration purposes, we assume that interval to be 20 minutes (end of curve's range). The numerical results, which were read by us from the curves in a very approximate way, are summarized in the Figures 8.1-1 through 8.1-4. The first thing that is apparent from these numbers is that, from an architectural point-of-view, one should not worry about I/O contention since it does not seem to make a significant difference. As one can see from the figures, the effect of varying from max to min I/O contention is less than 15%, which is insignificant in architectural terms. This conclusion that could have been arrived at in, as far back as November 1977, was actually eventually arrived at by the working group well over a year later.

The other point, that is obvious from these computations, is that since the present TDCC rated throughput, in Kips of Numerical Mix (N-Mix), is 182 Kips for the case of minimum I/O contention and 156 Kips for the case of maximum I/O contention, it is obvious that even the upward/point mass technique that is being used for this

8 GRAVITY I/O CONTENTION COMPENSATION	70	50	30
MAX I/O CONTENTION	550	400	280
MIN I/O CONTENTION	450	360	250

FIGURE 8.1-1
THROUGHPUT IN KIPS
LAUNCH INTERVAL = 20 SECS
VG - 8

# HIGH ACCURACY I/O CONTENTION MISSILES	24	12	6
MAX I/O CONTENTION	500	320	240
MIN I/O CONTENTION	480	280	180

FIGURE 8.1-2
THROUGHPUT IN KIPS
LAUNCH INTERVAL = 20 SECS
VG - 9

GRAVITY I/O COMPENSATION CONTENTION	70	50	30
MAX I/O CONTENTION	250	190	150
MIN I/O CONTENTION	200	160	120

FIGURE 8.1-3
THROUGHPUT IN KIPS
TRANSITION INTERVAL = 20 MINS
VG - 13

# HIGH ACCURACY MISSILES	24	12	6
MAX I/O CONTENTION	250	150	100
MIN I/O CONTENTION	200	140	90

FIGURE 8.1-4
THROUGHPUT IN KIPS
TRANSITION INTERVAL = 20 MINS
VG - 12

throughput estimation requires a computer which is significantly more powerful than the one presently on hand.

The key point is that computing systems cannot be designed to utilize, on a steady state basis, 100% of the resources. The reason for this is that they are basically complex queueing systems. A queueing system will break down once the steady state rate of resource utilization exceeds the 70% level. By the way, this is the true explanation of the software stabilization factor that this project uses. If a throughput of about 550 Kips is required, this means that the machine should be capable of a rated throughput of at least 0.75 MIPS (Million Instruction Per Sec).

In view of the fact that at this very same meeting, the conclusion had been drawn that the upward method was the one that required the least amount of computation and that methods such as Coating and Stokes would require an order of magnitude more computation, it should have been very clear, way back in November 1977, that a completely redesigned computer was called for.

Furthermore, with the technology picture that we had back in 1977, it would have been clear to anybody competent in designing computer systems that a reimplementation of the TBOC to obtain a full order of magnitude improvement of throughput would have been possible.

The study itself draws the following conclusions (View Graph 14):

- Throughput consideration place gravity computation in the transition interval.
- Minimal impact on throughput due to I/O contention, and weighting (refinement, offloading).
- Significant impact due to the number of high accuracy missiles, and the percent of gravity compensation.

8.2 January 24, 1978 Throughput Working Group Meeting

This meeting was completely dominated by the HFG question, and it devoted most of its time to the presentation of a very large number of architecture alternatives for the Trident II FCS Computer System. The fact that HFG dominates, actually better said, swamps everything else, is illustrated very clearly by a computational load analysis that is presented in the minutes of this meeting. This analysis shows that the number of milliseconds of computation required out of each second of elapsed time is given by a formula as follows:

$$\text{COMP LOAD (TDCC)} = (332 + \frac{37550}{L}) \text{msecs/sec}$$

where: L=Launch interval

From this formula it is easy to see that the launch interval has to be at least 56 seconds to make the above equation come out to 1000 milliseconds/sec. What this means is that, assuming for instance that 20 seconds is the ideal launch interval, that the TDCC is at least three times less powerful than required. Actually, this is even worse because, as we stated before, the computational power of the BP should not be utilized at more than 70% steady state level, which means that a TDCC would turn out to be more than four times less powerful than desired.

What is really interesting about this computation is that, the dominant factor of the above formula is the factor 37550/L. This factor, in turn, decomposes as follows:

$37550 = 2750$ (transition computation) + 4800 (prepare tasks and disk waits) + 30000 (HFG)

In other words, of the 37,550 units, 30,000 are due to HFG, which means that for all reasonable launch intervals, the computational load associated with HFG will swamp anything else.

Another interesting thing to note, in the minutes of this meeting, is that values for the TDCC throughput are given for the following instruction mixes: N-Mix = 182, point mass = 170, layer = 162, upward = 163, Stokes = 157. The largest difference between these throughput numbers is that which exists between the N-Mix and the Stokes mix. This difference is $182 - 157 = 25$ which is only 14% of the value of the N-Mix to begin with. As we stated repeatedly before, a difference of the order of 15% in data processing requirements, is below the level of architectural significance.

Our Figure 8.2-1 summarizes the throughput figures, in Kips for three basic processors which are: the TDCC, the SBP 9900 with hardware floating point, and the AMD 2900 bit slice with hardware floating point. These throughput figures are given for various instruction mixes just as we mentioned above

INSTRUCTION PROCESSOR	MIX N-MIX	P, MASS	LAYER	UPWARD	STOKES
TDCC	182	170	162	163	157
SBP 9900 HW FLT POINT	35	27	31	27	26
AMD 2900 BIT SLICE WITH HW FLT POINT	309	235	270	233	224

FIGURE 8.2-1
THROUGHPUT IN KIPS

METHOD	B-18 DATA x $\frac{35 \text{ KIPS}}{309 \text{ KIPS}}$	B-27 DATA
STOKES	200	201.6
POINT MASS	13.71	13.92
LAYER	4.64	4.56
UPWARD	31.38	31.2

FIGURE 8.2-2
COMPUTATION TIME IN MINUTES

with regard to the TDCC. An analysis of these figures shows that for the SBP 9900, the maximum delta between throughputs for the different instruction mixes amounts to a 26% variance, and for the 2900 bit slice processor it amounts to 27.5%. That is, for these latter two processors, the differences begin to get into the threshold of significance but they are still very marginally significant.

View Graph B-18 shows that the SBP 9900 executing any of the four HFG algorithms (point mass, layer, upward, and Stokes) exceeds, for 24 missiles, the readiness constraint for the transition interval. Therefore, the conclusion is that this micro processor could not handle the HFG computation, that is, that the simple addition of such a micro processor is not the answer.

View Graph B-19 shows the numbers for the case in which the HFG computation is shared between the TDCC and the SBP 9900. These numbers happen to fall within 10 to 15% from the numbers that would result with the TDCC alone, and again, in all, but the layer method, exceed the constraint for the transition interval. The conclusion that is drawn from the two View Graphs, B-18 and B-19, is that a single (referred to as monolithic) SBP 9900 micro-processor added to the TDCC will not help.

Having drawn this particular conclusion, the study proceeds to explore a number of distributed architectures. Basically the rationale being that if one micro processor can not do the job, one can explore the possibility of adding several micro processors

cooperating to give the computational power. The basic approaches that are considered are two, one is to missilize the micro processor, that is, dedicate a micro processor to each missile to do the HFG computation. The other approach is to partition the computation, that is, to partition the algorithm into phases that can be independently computed. If N independent phases can be partitioned out of the algorithm, then N micro processors could be kept busy increasing the throughput by almost the same factor.

In the event of the missilized micro processor, the computational time for 24 missiles is equal to that for one. That is, Stokes = 73.5 minutes, point mass = 5.03 minutes, layer = 1.7 minutes, upward = 11.54 minutes. Data access requirements are shown to be within the range of 80 kilowords to 120 kilowords, with a total data transfer delay amounting to somewhere between .3 to .5 minutes, which obviously is negligible. This last piece of information is interesting because it confirms that the HFG problem is strictly a compute bound problem.

These results for the SEP 9900 obviously indicate that all methods, except the Stokes integral method, are feasible, in the sense of not violating the constraint for the transition interval.

The second approach to a distributed architecture is really not carried very far, since information with regard to the partitioning of the algorithm is not on hand.

Next, the architecture study turns to explore the possibilities that could arise from the AMD 2900 bit slice. It basically predicates a 32 bit wide processor based on this bit slice component and in View Graph B-27, gives the corresponding data, on the assumption that all of the HFG computation would be done by this auxiliary processor.

Figure 8.2-2 shows the computation time in minutes for the four HFG algorithms. The first column shows some numbers we obtained from taking the data of View Graph B-18 and multiplying it by the simple N-LIX type ratio for the SBP 9900 (35 Kips) and that of the AMD 2900 bit slice processor (309 Kips). The second column shows the independently computed data given on View Graph B-27. As it can be seen, the agreement is extremely good. For all practical purposes all the work that is behind View Graph B-27 did not need to be done in the first place.

Figure 8.2-3 shows the same comparison for the case for the missilized version of AMD 2900. Again, the agreement between the computation time shown in View Graph B-29, which appears to be independently computed, and that which we computed from the data of B-18, is extremely good and for all practical purposes, it is the same data. In this figure, we, for the first time, show under the B-29 data, also, the delays due to the data transfers (the other set of numbers after the slash). In other words, it is only when one gets to the point of assuming

METHOD	B-18 DATA x $\frac{35}{309}$	B-29 DATA
STOKES	8.33	8.4/(*)
POINT MASS	.57	.58/.32
LAYER	.19	.19/.34
UPWARD	1.31	1.3/.46

FIGURE 8.2-3

COMPUTATION TIME IN MINUTES

/.xx = ADDITIONAL DELAY DUE TO DATA TRANSFERS

(*) VALUE OF DATA TRANSFER DELAY NOT KNOWN

a missile-dedicated processor (missilized 2900) which is twice as powerful as the current TDCC, that the data transfer delays become a significant portion of the computation's elapsed time.

This is another way to say that one does not need to worry about the delay in data transfer time, since it seems very unlikely that one would solve this problem by dedicating to each missile, a processor twice the power of the current Fire Control processor, in fact, there are better ways to solve this problem than this kind of overkill.

Another thing that one can read from the data presented in Figure 3.2-2 and 3.2-3 is that the Stokes method, at least as predicated in this particular study, is clearly the wrong approach to solve the problem. In fact, even after getting to the kind of overkill represented by Figure 3.2-3, the Stokes method still takes 3½ minutes.

The distributed architecture, using the AMD 2900 that would result by partitioning the HFG algorithms into phases is not further developed because, of course, no knowledge was present at the time, about potential phases of these algorithms.

A couple of additional cases involving the AMD 2900 are presented, both of them assume special instructions in micro code, resulting in about a thirty per cent improvement of throughput. One case is the monolithic auxiliary processor, and the other case is the missilized processor. These two cases are not even worth discussing because, as we stated repeatedly, thirty per cent is at the marginal significance level as far as architecture is concerned.

Finally the study winds up by suggesting (alluding to the possibility of using an auxiliary processor which would be a special purpose processor, namely an array type of processor, which of course, would have the intrinsic parallelism in three dimensional space, of performing simultaneously operations on the three coordinates and therefore, potentially being able to compute three times as fast as a conventional scalar oriented processor.

Quite frankly we are very puzzled by this entire performance, since the idea of complicating the system design by introducing N micro-processors, one for each phase of a partitioned algorithm, or worse yet, 24 processors, one for each missile, would introduce so many other complications in terms of either the physical, electrical connectivity of the system, or the system software to handle all of this. Such complexity seems to be totally unnecessary, because of the following reasons:

- a. It is not that difficult to design a processor which is software compatible to a previous one and has a performance

improvement of at least threefold and most likely, up to a full order magnitude. This is routinely done by computer manufacturers, as part of their design of product lines, and their evolution to track marketplace cost performance.

2. The wide spread in the computational times of the proposed algorithms would, in our opinion, be prima facie evidence that the root of the problem is the algorithm that is being proposed. That is that a good hard look at its complexity and whether it is really needed would probably be the best pay-off direction to go.

8.3 June 21, 1978 Throughput Working Group Meeting

The minutes start by stating the two fundamental questions as follows:

- a. Given the present processing capability of the Trident I FCS, how long will it take to perform the task on hand (i.e. existent Trident I tasks plus HFG)?
- b. Given the projected Trident II computational task, how much more processing capability (instruction throughput) will it be required to complete it in a launch interval which is x seconds long, or in a transition interval that is x minutes long?

The studies presented in this particular set of minutes finally use only the N-mix (i.e. no Stokes, layer, etc., specific mixes). Evidently, by this point in time, the project had realized that the difference between the mixes was not significant.

It is also stated explicitly that at a transfer rate of 500 kilowords per second, the I/O contention degradation of the TDCC N-mix is empirically established to be 15%. This confirms our previous statement that this problem of I/O contention is not architecturally significant, at least not for the HFG.

The minutes also have a lengthy discussion of the software stabilization factor that is used to increment the various estimates of throughput. The rationale for this factor is that empirical evidence shows that software development and maintenance costs grow very rapidly once computer resources are utilized at a level, quoting from the minutes, higher than 75 to 85%. This is indeed true and it is a wise move in making these estimates of processing capacity to utilize an augmentation factor. We would recommend an even more conservative approach that is justified by truly understanding the reasons for this phenomena. In fact, this phenomena of escalating costs is due to the nature of computing systems being systems of queueing servers. Queueing servers, if they are forced to operate under stochastic conditions with a steady resource utilization level in excess of 70%, saturate, causing all kinds of strong interactions, and system break-down conditions to occur. In light of this kind of theoretical understanding of the reason why the software costs go up, it is prudent to use a capacity augmentation factor of 30% or more. This, by the way, is the reason why we have repeatedly made the statement that 30% is kind of the border line of significance as far as architecture is concerned. The point being that capacities in a computing system must be viewed as "fuzzy" quantities which have to

have built-in a (fuzzy) slack interval of the order of 30% to accommodate the statistic/stochastic nature of the work load.

So we applaud the wisdom of the "software stabilization factor" but we are disappointed by its rationalization showing a lack of understanding of the underlying phenomena and resulting in a value for it which is too low.

The meeting was primarily concerned with the refinement of the throughput study. The latter considered only the HFG requirements. As we have seen previously, HFG requirements completely swamp everything else so that this restriction does not invalidate the study. Before commenting on the study *per se*, we would like to present the study conclusions.

- a. The computational techniques (Stokes, Coating, upward) is the most significant factor affecting the required throughput.
- b. Stokes and Coating require significantly more throughput than what is available in the Trident I FCS.
- c. The upward continuation method is feasible within the available computing power but no spare capacity would be left.

- d. A significant reduction of throughput is alleged in connection with the weighted case which reflects the impact of off-loading periodic tasks from the TDCC to some other processor. That is, a benefit is claimed for a distributed Fire Control Architecture.

Frankly, as we will comment below, this off-loading is not sufficiently explained to warrant any such rosy conclusion about distributed architectures.

- e. Less throughput will be required if HFG will be done during the transition interval as opposed to the launch or fire interval.
- f. The impact of I/O contention is not very significant.
- g. Some reduction of throughput will result by reducing the number of data points for the spline.
- h. The re-GI assumption has a minimal impact on the throughput requirements.

By and large, these are the first set of sensible conclusions we have seen in this entire set of studies, and the proof that could have been gotten much sooner is that when we studied the minutes of the working group, we read and commented on them in historical order, that is starting from the oldest one first and before reading the most recent ones, including this one, we had achieved exactly the same conclusions with an

infinitesimal fraction of the effort that probably went in doing all of this.

The study itself presents a very large number of graphs of the type that were used also in previous studies. All of the graphs presented assumed that the effect of the gravity field perturbation after the boost is neglected, and also that the computational method is based on the spline fitting through 6 points.

We will only mention a few numerical results from the basic case. They indicate, again reading the curves in a very approximate way, that for a 20 seconds launch/fire interval, (end of curve's range) the throughput required by the three methods under consideration, is as follows; Stokes 1250 Kips, Coating 700 Kips, upward 180 Kips. Similarly, the throughput required, if the HFG calculation is done during the transition period, and assuming a transition period of 20 minutes (end of curve's range) turns out to be Stokes 500 Kips, Coating 300 Kips and upward 120 Kips.

In addition to the basic case, the study presents a weighted case which uses an off-loading factor (off-loading periodic tasks from the TDCC to some other processor). This off-loading factor is not sufficiently explained. Additional cases are generated by considering an enhancement factor (refinement of

certain existing computational tasks). This factor, too, is not sufficiently explained or defined. Finally, additional cases result from introducing the so called software stablization factor. Having introduced all of these different corrective factors, the study proceeds to generate graphs for all possible permutations.

8.4 April 6, 1979 Throughput Working Group Meeting

The minutes of this particular meeting show that the next working group meeting was tentatively scheduled for May/June 1979. We presume that this meeting never came off since we were not given the minutes of any other meeting beyond the April 6, 1979.

Also these minutes are the first time when the Trident I Upgrade Program is ever mentioned as a possible way station to the Trident II. We read both of these two elements to mean that back into the spring of 1979 the commitment to the Trident II Program was beginning to weaken. So, probably, there have been no more working group meetings. In fact, the eventual cancellation of the Trident II Program and replacement of it with a C4 Upgrade Program would remove the pressure from the implementation of the HFG feature.

In connection with the discussion of the Trident I Upgrade as a way station to the Trident II Program, is stated that the Fire Control System could begin the evolution towards a distributed architecture in the Trident I Upgrade.

The basic point seems to be that the results of the analyses presented at the June 21, 1978 meeting, subsequent work, and discussions must have convinced people more and more that a distributed architecture is the way to go.

It is hard to be completely sure of this because we are only dealing from these minutes. If this is true, we believe this to be very unfortunate because, as we have stated above, the data and the way it has been arrived at is hardly a convincing base of evidence for the necessity of a distributed architecture with all its complexities.

The key point that we are making is that, nowhere in these particular minutes, or any of the preceding meetings, we see any indication that somebody has done a study of as simple an issue as, for instance, what would be the potential TDCC throughput improvement resulting from the introduction of a cache memory. The way we look at it, a 102 Kips computer is taking on the average 5 microseconds per instruction. This would correspond to a memory cycle somewhere between 500 nsec and 1 usec with a cache memory is completely possible to get down to 70/100 nano-seconds. Plenty of studies in the computer industry have shown that one does not have to have a very large cache size to get to hit ratios of the order of 90%. Such a modification would be totally software invisible, and even from a hardware engineering point-of-view, is not a big deal. In fact, it has been done a number of times by many many manufacturers who have rejuvenated old designs that way. The point is that this is, comparatively speaking,

a much easier thing to do instead of all of these complicated analyses on architectural possibilities. The fact that it has not been done, or at least not documented, removes a lot of credibility from the studies that are being presented in these minutes.

The minutes indicate that the Burst Height Compensator (BHC, Smart Fuse) will not significantly increase the Fire Control trajectory processing throughput. So this is additional evidence that many of the new requirements of Trident II do not contribute in any significant way to the throughput issue, and HFC remains the crucial question.

The bulk of the minutes are dedicated to a discussion of the correlation method. The method is essentially based on the idea that the gravity field variation effects over the trajectory correlate very strongly with the gravity field anomaly at the point of launch. This method had been previously considered, however, what has been presented at this meeting is a new implementation called the Release Method which is shown to have the ability to achieve an accuracy of double that of the C4. The most promising aspect of this method is that only a limited number of co-efficients (24) must be computed. The claim is also made that this method requires, in terms of processing

capacity (storage and Kips), about the same as that required by the computation of no more than 24 Stokes vectors, that is an order of magnitude reduction. The 24 vectors in question correspond to one initial vector (at launch point) per missile. Thus, this is, potentially, a dramatic saving from the implementation methods using a spline approximation which require 18 vectors per missile as opposed to the 100 vectors that were thought to be needed to begin with. (See November 10, 1977 minutes.)

Another key point that comes out of the documentation is that although this Release method appears very promising in reducing the required throughput, it requires extensive validation, which is presently underway, before it can be committed to.

Finally, the point is made, that to achieve an accuracy 3 times as high as C4, a four to six spline method must be considered. As we understand it, if the accuracy is approximately twice C4, then this single vector approach can work. If the requirement for accuracy goes beyond that, then one could not consider the simpler method and one is forced

to compute a 4 to 5 point spline fit, which would throw one back to the kinds of throughputs that we have seen mentioned in the Section 8.1, 8.2, and 8.3 above.

The minutes also mention the interesting point that no estimates of execution time, or memory requirements, have been developed for the various possibilities for improving the contributions to the W-Matrix errors. This is interesting mostly from the point-of-view that by the time this meeting met, we are about two years down the road from the meeting that was discussed in Section 8.1.

The basic conclusions of the meetings seem to be as follows:

- Accuracy equal .7C4 Release Method
- Accuracy equal .5C4 Release Method
- Accuracy equal .3C4 Stokes Method (4/5 spline)
Doing HPG computation during the transition phase rather than the firing phase, while improving the throughput, raises some concerns because the data would not be recent (data staleness issue). A study of the effects of data staleness is called for by the minutes.
- No requirements have been identified anywhere that would suggest that memory in excess of 262 kilowords might be required by the projected applications.

View Graph A-29 makes a further pitch for the approach of using micro-processors as opposed to the budget assumption, which is to augment throughput by additional Basic Processors. The arguments given against the use of additional Basic Processors are the following:

- MCC Space
- Poor Hardware Cost Effectiveness Dollars/Kips
- Limited Throughput Growth Increment

Both of these approaches seem to us to be poor ones and so we are somewhat surprised in seeing this particular view graph and this particular comparison. The fundamental comparisons that should be made is between a, so called, distributed architecture, and a software transparent redesign of the TDC using state-of-the-art technology.

3.7 October 11, 1979 Memory Study

This paper was produced by Hughes Aircraft Corporation and appears to be an informal/white paper examining a number of alternatives to expand the addressability of the memory controller beyond the current 256 kilowords. The key aspects of this study are that the redesign of the memory controller has been authorized and committed, however, under a strong directive that no other system components have to be modified. Furthermore, the memory control is being redesigned for one megaword addressability. Plans seem to be underway to develop an actual physical memory of 512 kilowords, using a 64 K-bit chip.

Due to the characteristic of the chip being utilized, the access time is being reduced from 7 ticks to 2 ticks that is down to 520 nano-seconds. It is further reported that the reduction of the access to 520 nano-seconds will improve the M-mix throughput of the processor by 30%. We are, frankly surprised, by the rather modest level of throughput improvement resulting from a $3\frac{1}{2}$ fold reduction of the access time to the memory. This must be due to the stated constraint of confining changes only to the memory controller, thus the CPU cannot take full advantage of the increased access speed of the memory.

The paper presents a number of possible schemes for increasing addressing. We will briefly summarize all the schemes

presented and their evaluation.

Register Extension

This scheme consists in widening all of the controller paths from 13 bits (256 K-words) to 20 bits (1 megawords). This scheme definitely violates the directive, namely, to modify only the MCLR. For this reason the scheme is considered only to some extent. The study indicates that this scheme is characterized by the following:

- It is the most attractive of all the schemes presented.
- It has a straightforward implementation.
- The hardware/software architecture of the system will remain the same.

The basic problem is that all 32 bits of the word have been used. Consequently, to implement this scheme a redefinition of the instruction format must be done with impact on the software, of course. We also happen to agree with HAC that this is the best scheme and we know, through several similar experiences, that the software impact of these kinds of redefinitions does not necessarily need to be catastrophic. That is, the key is the proper definition of the new instruction format. Sometimes it is possible to define a revised instruction format so that the impact on

software can be contained.

As we see it, the first thing to do would be to remove the restriction to confine the modifications only to the MCLR. This restriction has to be removed in any event, because with such a restriction it becomes essentially impossible to get any expansion of addressing that has any true architectural and application value. Once the restriction is removed, it would be important to commission a study for a redefinition of a super-set architecture that would be, to the maximum extent possible, compatible with the present TDC instruction format, while gaining the two extra bits of addressability in the instruction format. Only if such a study would show that there is no possibility of doing this, this particular alternative should be dropped.

Automatic Memory Duplication

This scheme would envision two banks of memory, 256 kilowords each, where automatically every store would be doubled in both banks. The scheme ends up using the full 512 kilowords of memory and it does not have any addressing problem. However, it does not extract any usefulness out of the extra 256 kilowords other than extra redundancy and, consequently, reliability. What is wrong with using redundancy this way is that not all the program and data

structures in the system need full redundancy. Thus, it is a pseudo argument, and a specious one to argue that one would get some benefit out of 512 kilowords of memory by this kind of scheme, since the same reliability effects can be obtained, probably, by duplicating only about 10% of the involved information structures and, consequently, not wasting all kinds of memory.

Main Memory Banking

This scheme logically widens the 18 bit address to 20 bits by concatenating two bank select bits (256 kilowords banks). The bits to be concatenated would be provided by Bank Select Registers (BSR). The idea would be to provide one BSR for each MCCLR interface i.e., 4 BSRs.

this scheme, again, violates the directive of confining changes only to the MCCLR. The study reports the following advantages for this scheme:

- BSR's concept
- All changes are limited to the OS, and more specifically to the monitor, since the monitor would have responsibility for the BSR switching and the memory assignment activities.
- No impact on application tasks and on the software factory (the collection of tools for developing software). The reason for this is that the banking activity would be handled strictly at run-time.

On the second page of the discussion of the main memory banking scheme, the study gives under Paragraphs 1., 2., 3., 4., and 5. a number of very good reasons why the directive imposed on this redesigned memory system is totally unacceptable.

Dynamic Memory Relocation (DMR)

This scheme consists in adding another level of address translation so that one would go from the 16 bit program address to the 18 bit absolute address (the present scheme) with then the 18 bit absolute address being fed in the second stage of address translation to produce a 20 bit physical address.

This particular scheme would cause the physical address space to be divided into 64 16 kiloword segments. By the way, the 16 bit address is referred throughout the TDCC documentation as being a virtual address. Actually, it would be preferable to refer to it as being a name space address as opposed to virtual. In fact, virtual memory has acquired, in practice, the connotation that the program name space is greater than physical memory and in this case, we have exactly the reverse. Both for the current TDCC, as well as, the TDCC with the DMR, the program name space is considerably smaller than the physical memory.

A scheme where the program name space is smaller than physical memory is well justified in real-time applications where one would like to have a number of program name spaces stored directly in real memory so that they can be commuted for execution at a very fast rate.

Returning to DMR, the paper states that the introduction of the extra address translation stage will increase the access time by one tick and namely, it will bring the access time to 780 nano-seconds resulting in an N-mix throughput improvement of only 24% as opposed to the 30% with two ticks. Frankly, if the loss is only 6% of the throughput, we would not consider this a fundamental argument against this particular scheme.

Since in implementing this scheme, one would have to utilize BSAs, the same arguments that were given for the main memory banking apply, namely, that as long as one is under the restriction to confine the changes to just the MCCLR then this scheme will not work.

Appending TID and MID

TID stands for Task Identification Number, and MID for Module Identification Number. The idea of this scheme is again to use the same BSPs that would be used in the DMR scheme, but append

two bytes, one to contain the TID, and the other to contain the MID. All this would do is add some additional protection capability. It can be validly argued that such protection capabilities would be redundant with what already exists in the computer anyway. The paper, therefore, recommends that this particular scheme be dismissed. We totally concur with it.

The final conclusion of the paper is that the only scheme that is consistent with the directives to confine changes to MOCLR is the memory duplication. The paper itself winds up recommending the DMR scheme. We, believe, that first of all, the restriction should not have been imposed in the first place, and secondly, that actually one should consider more carefully, the register extension scheme. In particular, with the idea of coupling it with the widening of the address space of the individual user task since the obvious trend is for user processes to use very large name spaces especially for the handling of data.

3.6 June 10/11, 1980 Dahlgren Program Review

At the program review held in Dahlgren on June 10/11, 1980, Mr. Carlton Duke indicated that the whole issue of the future design of the FCS system has lost most of its criticality because of some major program changes that have occurred since the CSO present effort was started. Basically, as we understand it from other sources, the major program changes are that the Trident II program has been discontinued and replaced by a more modest program of up-grade of the C4 (C4U).

Mr. Duke indicated that the three major improvement areas High Frequency Gravity Compensation, W-Matrix, and at Sea calibration are presently considered part of the modernization program. Such a program will have its initial flight tests in the 86/87 time frame. Mr. Duke further added that the project intentions is to study all of these different computational problems for the next two or three years so that more precise ideas would be jelled on how to handle them. This extended engineering phase, of the next two to three years, should, then result into a much more precise information on whether the FCS needs more CPU power, Mass Storage, etc., etc..

Mr. Duke confirmed that the HFG is, indeed, the big item and, furthermore, there is a commitment to have at least

a baseline scheme for handling it by the end of 1981. Apparently the project has made a commitment to actually implement various HFC algorithms on the present machines (both the CDC 6700 and the BP). These experimental implementations would be aimed at finding out the answers to questions such:

- How much additional compute power is required.
- How much additional memory.
- How much additional mass memory etc.

Mr. Luke also indicated that the new MK/6 inertial guidance system is very different from the past one and it may include on/line calibration (at sea calibration).

He also stated that it is the view of the project that the new guidance system could pose a great burden on the existing computing hardware and this is one of the primary reasons why micro processors are being considered. No decisions have yet been made with regard to:

- Does the accuracy gain justify the added complexity, risk, etc.
- If the accuracy gain justifies the new approach will it be done by Guidance or Fire Control?

Besides conveying an impression of lack of urgency, because of the programmatic changes (Cancellation of Trident II Program), Mr. Duke also gave us a very strong impression that the entire issue of HFG was no longer to be considered as important as it had been in the past. The importance of HFG has been extensively documented in this section in the process of reviewing the minutes of the working group on throughput. It rests on the fact that, just about any approach to HFG computation, seems to lead to a requirement for a computer with many times the power of the TDCC, at least in terms of Kips.

The way Mr. Duke justified this much more relaxed attitude on his part, during our program review, was that the early studies were based on the viewpoints of theoretical geophysicists, which while being correct, were excessively complex and cumbersome. Since then, according to him, the approach has been simplified considerably cutting down, consequently, on the amount of computation that is to be performed.

We specifically asked Mr. Duke to give us some examples of this type of simplification and he indicated that, for instance, considerable simplification comes from realizing that most of the High Frequency Gravity effect occurs during the boost phase of the trajectory. Another simplification came from realizing that instead of computing a Stokes double integral every second, one could very easily cut it down to computing a Stokes integral every 10 seconds and then interpolating the gravity vectors every one second. Since

the boost phase is approximately 200 seconds, this reduces the number of gravity vectors that have to be computed (Stokes integrals) to 20 or so. He also referred to the five spline interpolation method, which will result in about the same accuracy, but will result into an order of magnitude reduction of computation. Finally, he referred to a method which allows computing a single gravity vector at the launch point. He indicated that this yields approximately about 60% of the accuracy of the five spline method mentioned above. Mr. Duke must have been referring to the Release method that was discussed in the last of the working group meeting reports that we have.

He also indicated that on top of all these simplifications one can simplify the actual Stokes method (upward continuation) resulting in a greatly simplified computation of the individual gravity vectors. Finally, he stated another approach is to precompute the gravity vector at several points in the space around the earth, and use lots of mass memory to store these precomputed numbers, to get, consequently, a very fast throughput.

After our careful study of the minutes of the working group on throughput, we believe that all of the above alleged simplifications have been discussed extensively by that working group and that the only alternative that was

not presented at those meetings was the notion of precomputing the gravity vectors and, consequently, trading off storage against computational power. So except for this alternative, we do not see that any of the information given to us at Dahlgren changes any of the conclusions that we reach, as far as computing system architecture is concerned, from the study of the documentation available to us.

We believe, in particular, that it is most unfortunate that none of the documentation made available to us, gives us any information whatever on the potential trade-off between the use of mass storage memory (resort to precomputation of the gravity vectors) versus greater computational power. In fact, as we stated several times before, our overall reaction to the throughput studies is that the computational algorithms for HFG are making an excessive demand for computing power. What we are saying is that from our experience of having designed many computing systems, those requirements were coming across as too lopsided in the direction of computer power. Consequently, it would have been very interesting for us to look at alternatives where storage could have been traded-off against computing power.

8.7 GSG Conclusions

In this section we will briefly describe the direction GSG would recommend for the evolution of the FCS. The evolution being required by the additional and expanded applications that have been documented to us. Our recommendations are aimed at selecting a direction of technical evolution of the FCS computer system that would minimize the overall technical risks of the project. Technical risks are equated by us with risks associated primarily with the development of systems software.

The most important conclusion we can arrive at, after the review of the documentation regarding additional application requirements, is that the High Frequency Gravity compensation represents the predominant factor in the documented workload for the modernization program. An ancillary conclusion is that the most important factor in determining the computing workload generated by HFG is the selection of the HFG algorithm. In light of the above, we offer the following recommendation:

Recommendation #1:

Every effort possible should be performed to formulate an algorithm that, while consistent with the accuracy requirements, imposes the very minimum computational load on the Basic Processor.

In other words, what we are saying is that efforts spent on the algorithm will pay a handsome dividend with regard to insuring a non-lopsided architecture for the computing system.

The evidence provided to us points in a clear and convincing way to the fact that, regardless of what algorithm is ultimately chosen for HFG, a processor of considerable greater power than the present Basic Processor is required by the program. We would also conclude, from the evidence, that the new processor would have to have a throughput capability of at least 1 MIP.

Our review of the architecture of the TDCC and of the Operating System RTOS shows that a solution based on a processor with an architecture that is upward compatible from that of the present TDCC should not have unacceptable disadvantages.

Our view of upward compatibility implies the careful examination of the possibility of modifying the instruction format so to allow increasing the process addressability to at least 18 bits (256 kilowords) as opposed to the present 16 bits (64 kilowords). We realize that improving addressability by 2 bits requires reformatting the instruction format to obtain the extra bits. We believe, however, that this is very important, in view of the overall trend for user or application programs to use, especially for their data, segments greater than 64 kilowords.

In light of the above comments, we would like to make the following recommendations:

Recommendation #2:

Commission the design of a 32 bit, high speed, scientific processor of at least 1 MIP. Such a processor should be strictly upward compatible^(*) with the TDCC, and it should have a segment size of at least 256 kilowords (18 bit application program addressability, or 18 bit virtual addresses). Such a study should result in a design that is sufficiently complete to determine such factors as: Cost, volume, implementation time, power consumption, reliability, etc..

Recommendation #3:

Do not consider distributed architectures until either the study referred in Recommendation #2 results into a very strongly negative indication (we do not expect that such a negative indication will result from the study), or requirements, other than those stated in the documentation reviewed by us, give a very strong indication that a multiprocessor architecture is truly required.

(*) Strictly upward compatible: allows importing any TDCC software via a simple recompilation process.

We believe that, if the definition of the upward compatible architecture of the new processor is carefully done, the Monitor and the execs, which are written in assembly language, will transport to the new architecture with essentially no problems. In any event, even if there were to be problems importing the Monitor and the exec, we expect that, if the architecture is truly upward compatible, the modifications that would be required would be local in nature and greatly aided by the existing PDLs.

Our review of the RTOS design leads to the conclusion that, even though there are some features of that design which can be questioned, in balance it is a good design. Furthermore, the additional application requirements that have been documented to us, do not constitute evidence that the present architecture of RTOS is obsolete. This is an important reason for our recommendation that a strong effort be done in the direction of enhancing the FCS System via an upward compatible higher power processor.

FILMED
- 8