NASA Technical Memorandum 87574

The SIFT Hardware/Software Systems - Volume I A Detailed Description

Daniel L. Palumbo

September 1985

Date for general release September 30, 1987

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

29-JUL-85 The SIFT Hardware/Software Systems - Volume I
A Detailed Description

1.0 Introduction 1
2.0 Hardware Configuration 3
2.1 The Datafile, Transaction File and Broadcasting. 3
2.2 The Real-Time Clock. 6
3.0 The Sift Operating System 6
3.1 The Local Executive 7
3.1.1 Local Executive Data Structures 7
3.1.1.1 Task Tables (TT array) 7
3.1.1.2 Task Schedules (SCHEDS array) 9
3.1.1.3 Buffer Information Table (BINF array) 10
3.1.1.4 Buffer Table (BT array) 11
3.1.1.5 Vote Schedule (SCHEDS array) 11
3.1.1.6 POSTVOTE Array 13
3.1.1.7 ERRORS Array 13
3.1.2 Local Executive Procedures and Functions 13
3.1.2.1 System Startup and Initialization. 13
3.1.2.1.1 PROCEDURE INITIALIZE. 14
3.1.2.1.2 PROCEDURE GPROCESSOR. 14
3.1.2.1.3 PROCEDURE DBADDRS. 14
3.1.2.1.4 PROCEDURE WORK. 14
3.1.2.1.5 PROCEDURE SYNCH. 14
3.1.2.1.6 Construct POWER2 Array. 14
3.1.2.1.7 PROCEDURE RECBUF(NW,P: PROCESSOR; S: SCHINDEX). 14
3.1.2.1.8 Resynchronizing. 15
3.1.2.1.9 Presetting Some Variables, the POSTVOTE and TT Arrays. .. 15
3.1.2.1.10 PROCEDURE BUILDTASK(TASKNAME: TASK). 15
3.1.2.1.11 Establish Initial Configuration. 15
3.1.2.1.12 Initializing Modules SIFTAP and SIFTIC. 16
3.1.2.2 Clock Interrupts and Scheduling. 16
3.1.2.2.1 FUNCTION SCHEDULER(CAUSE: SCHED_CALL; STATE: INTEGER):INTEGER. 16
3.1.2.2.2 PROCEDURE TSCHEDULE. 16
3.1.2.2.3 PROCEDURE VSCHEDULE. 17
3.1.2.2.4 Activating the Task. 17
3.1.2.3 Voting. 17
3.1.2.3.1 PROCEDURE VOTE(TK: TASK; DEFAULT: INTEGER). 17
3.1.2.3.2 FUNCTION VOTE3(DEFAULT: INTEGER): INTEGER. 18
3.1.2.3.3 FUNCTION VOTE5(DEFAULT: INTEGER): INTEGER. 18
3.1.2.3.4 PROCEDURE ERR(P: PROCESSOR). 18
3.1.2.3.5 PRCEDURE FAIL. 18
3.1.2.4 Inter-task Communication. 18
3.1.2.4.1 PROCEDURE STOBROADCAST(B: BUFFER; V: INTEGER). 19
3.1.2.4.2 PROCEDURE BROADCAST(B: BUFFER). 19
3.1.2.4.3 PROCEDURE WAITBROADCAST. 19
3.1.2.4.4 FUNCTION GETVOTE(B: BUFFER): INTEGER. 19
3.1.2.4.5 FUNCTION MEDIAN(B: BUFFER): INTEGER. 20
3.2 The Global Executive 20
3.2.1 GLOBAL FUNCTION CLKTASK. 20
3.2.1.1 Constant MAX_WINDOW. 20
3.2.1.2 Constants CLK_BUF and CLK_TRANS. 21
3.2.1.3 Constant COMMDELAY. 21
3.2.1.4 Constant OMEGA. 21
3.2.1.5 Array SKEW. 21
3.2.1.6 The Body of CLKTASK. 21
3.2.1.7 A Lesson in Disabling Interrupts and Malicious Liars. 22
3.2.2 The Interactive Consistency Tasks. 23

3.2.2.1 The Data Buffers of the Interactive Consistency Tasks. 23
3.2.2.1.1 The "A,B,C" Input Buffers. 23
3.2.2.1.2 The "O" Buffers. 23
3.2.2.1.3 Buffers EXPECTED and XRESET. 24
3.2.2.1.4 Buffers LOCK and NDR. 24
3.2.2.2 GLOBAL FUNCTION ICT1. 24
3.2.2.2.1 FUNCTION RANDOMIZE(SEED: INTEGR): INTEGER. 24
3.2.2.2.2 PROCEDURE COMUN1553A(ADR,N,SA,MODE,RT: INTEGER). 25
3.2.2.2.3 PROCEDURE GETNDR. 25
3.2.2.2.4 PROCEDURE GETREALDATA. 25
3.2.2.2.5 PROCEDURE GETRANDOMDATA. 26
3.2.2.2.6 PROCEDURE GETNEWDATA. 26
3.2.2.2.7 PROCEDURE DISTRIBUTE. 26
3.2.2.2.8 The Body of ICT1. 26
3.2.2.3 GLOBAL FUNCTION ICT2. 27
3.2.2.3.1 PROCEDURE REBROADCAST(VPX,P: PROCESSOR). 27
3.2.2.3.2 The Body of ICT2. 27
3.2.2.4 GLOBAL FUNCTION ICT3. 27
3.2.2.4.1 PROCEDURE GETIC2PROC(IC1P: PROCESSOR). 27
3.2.2.4.2 PROCEDURE VOTEDATA(DB: INTEGER). 28
3.2.2.4.3 PROCEDURE RESTORE. 28
3.2.2.4.4 The Body of ICT3 28
3.2.2.4.5 GLOBAL PROCEDURE ICINIT. 28
3.2.3 The Redundancy Management Tasks. 28
3.2.3.1 Data Structures and Buffers of the Redundancy Management Tasks. 29
3.2.3.1.1 The Constant THRESHOLD. 29
3.2.3.1.2 Buffers ERRERR, GEXECRECONF, and GEXECMEMORY. 29
3.2.3.1.3 PRESENTCONFIG. 29
3.2.3.1.4 NUMWORKING and NW. 29
3.2.3.1.5 WORKING. 30
3.2.3.1.6 The Processor Mappings VTOR, RTOV, and VTODF. 30
3.2.3.1.7 Schedule Pointers TPI and VPI. 30
3.2.3.2 GLOBAL FUNCTION ERRTASK. 30
3.2.3.3 GLOBAL FUNCTION FAULTISOLATIONTASK. 31
3.2.3.4 GLOBAL FUNCTION RECFTASK and FUNCTION XRECF(RECONF: BITMAP). 31

19-AUG-85 The SIFT Hardware/Software Systems - Volume I
A Detailed Description

1.0 Introduction

The Software Implemented Fault-Tolerance (SIFT) computer system was developed for NASA by SRI International as an experimental vehicle for fault-tolerant systems research. The SIFT effort began with broad, in-depth studies stating the reliability and processing requirements for digital computers which would, in the aircraft of the 1990's, control flight-critical functions. (See refs. 1 and 2.) Detailed design studies were made of fault-tolerant architectures which could meet the required reliability and processing requirements. (See ref. 3) Following these studies, SRI International and the Bendix Corporation designed and built the SIFT system which was delivered to NASA's AIRLAB facility in April 1982 (see ref. 4). The SIFT architecture consists of a fully distributed configuration of Bendix BDX930 processors with a point-to-point communication link between every pair of processors. (See fig. 1.) Although the design can accommodate up to eight processors, only six processors are in the current system; hardware reliability estimations have demonstrated that this is adequate to meet the stated reliability goals of a probability of failure less than 10^{-9} for a 10-hour flight.

Important distinctions between SIFT and other fault-tolerant computers are:

1. The functions supporting fault tolerance are primarily implemented in software (e.g. voting).
2. Different tasks can be supported at different replication levels (i.e. a non-critical task may be simplex whereas more critical tasks can be replicated 3-fold or 5-fold).
3. The unit of reconfiguration is a complete computer, i.e. processor, memory, and busses.
4. The design is not based on a special CPU or memory design.
5. The redundant computers are loosely synchronized.

The assignment of tasks to processors in SIFT is predetermined by a task schedule table which is constructed by the application designer. The SIFT scheduler periodically dispatches tasks according to this schedule. Random processor failures which occur during system operation create different sets of working processors, or configurations. The application designer must define a task schedule for each level of configuration the system may encounter. Reconfiguration in SIFT is accomplished by selecting the task schedule which corresponds to the current set of working processors. The decision to reconfigure is based on error information gathered when the data from replicate tasks is voted.

The synchronization of the computers is fundamental to the correct functioning of the communication system. Interprocessor communication is completely asynchronous. No handshake signals or rendezvous mechanisms are used. The validity of data is established by the precedence established in the task schedule and the synchronization of the processors.

The SIFT operating system has two levels of authority. The Local Executive contains procedures which support scheduling, voting and communications. The Global Executive consists of tasks which cooperate to provide synchronization, redundancy management (fault isolation and reconfiguration) and interactive consistency. Since the delivery of SIFT, development and testing has continued at NASA Langley Research Center and several versions of the

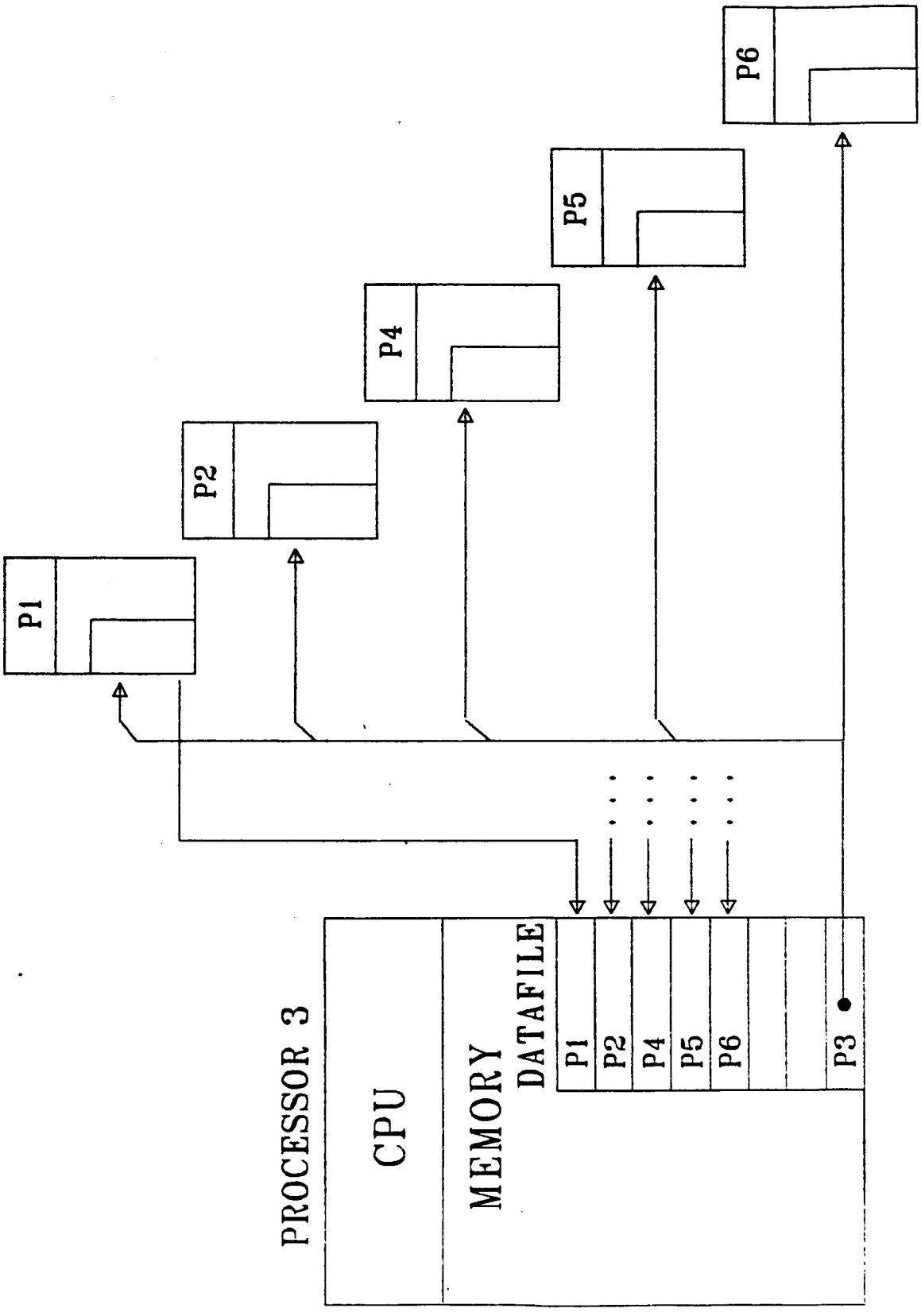


Figure 1. SIFT System Interconnect

operating system have evolved (see ref. 5). Each new version represents the different strategies employed to improve the performance of particular functions of the operating system.

Version B - The operating system as delivered.
Version R - Improved reconfiguration performance.
Version V - Improved vote performance.

In Version B, tasks which require extended execution time stretch their alloted time slot by disabling the clock interrupt. This is unacceptable because it results in varying delays in the execution of the periodic application tasks. The largest delays were encountered during reconfiguration. The operating system was redesigned to reduce this overhead. This new version is referred to as Version R. Version R is able to support reasonable task schedules without disabling the clock interrupts. Finally, the vote system was redesigned to improve the vote performance. This version is referred to as Version V.

The purpose of this document is to describe in detail Version V of the SIFT operating system. An explanation of the relationship between Version V and the two previous versions of the operating system is found in ref. 5. To fully understand the material presented in this document, Volume II of this report, "Software Listings", should be referenced when necessary. To facilitate discussion of the software design, a description of the hardware configuration precedes the software sections.

2.0 Hardware Configuration

The SIFT hardware consists of seven Bendix BDX930 avionics computers which communicate via a point-to-point broadcast network. Each computer in the system has a 16 bit CPU, 32K words of static RAM memory, 1K datafile memory, 1K transaction file memory, a broadcast controller, a 1553A controller, and a real-time clock (see figure 2). The CPU is constructed from AM2901 bit slice chips in a micro-programmed pipeline architecture and achieves a performance level of 1 MIPS. The 1553A controller provides a MIL STD 1553A bus interface for communication with external aircraft systems.

2.1 The Datafile, Transaction File and Broadcasting.

The datafile is a 1K memory block external to system memory which serves as a buffer area for the broadcast and 1553A controllers. The datafile begins at address 7400_{16} and is partitioned into eight 128 word "mailboxes" (see figure 1). Each input data stream from the broadcast network is hardwired to a mailbox, maintaining communication isolation. Included in the datafile address space are 8 locations (DATAFILE[1016 -> 1023]) which are memory mapped hardware registers (see figure 3). These registers provide access to the 1553A controller, the broadcast controller and the real-time clock. To prohibit an errant remote processor from gaining access to the I/O registers, the local processor must own the last mailbox (DATAFILE[896 -> 1023]). This requirement causes each processor's mailbox ordering to differ. For example, the mailbox ordering for processor 1 is 2-3-4-5-6-7-1. The ordering for processor 2 is 1-3-4-5-6-7-2, and so on. A processor uses its mailbox for an output buffer and pseudo input mailbox. A processor cannot actually broadcast to itself.

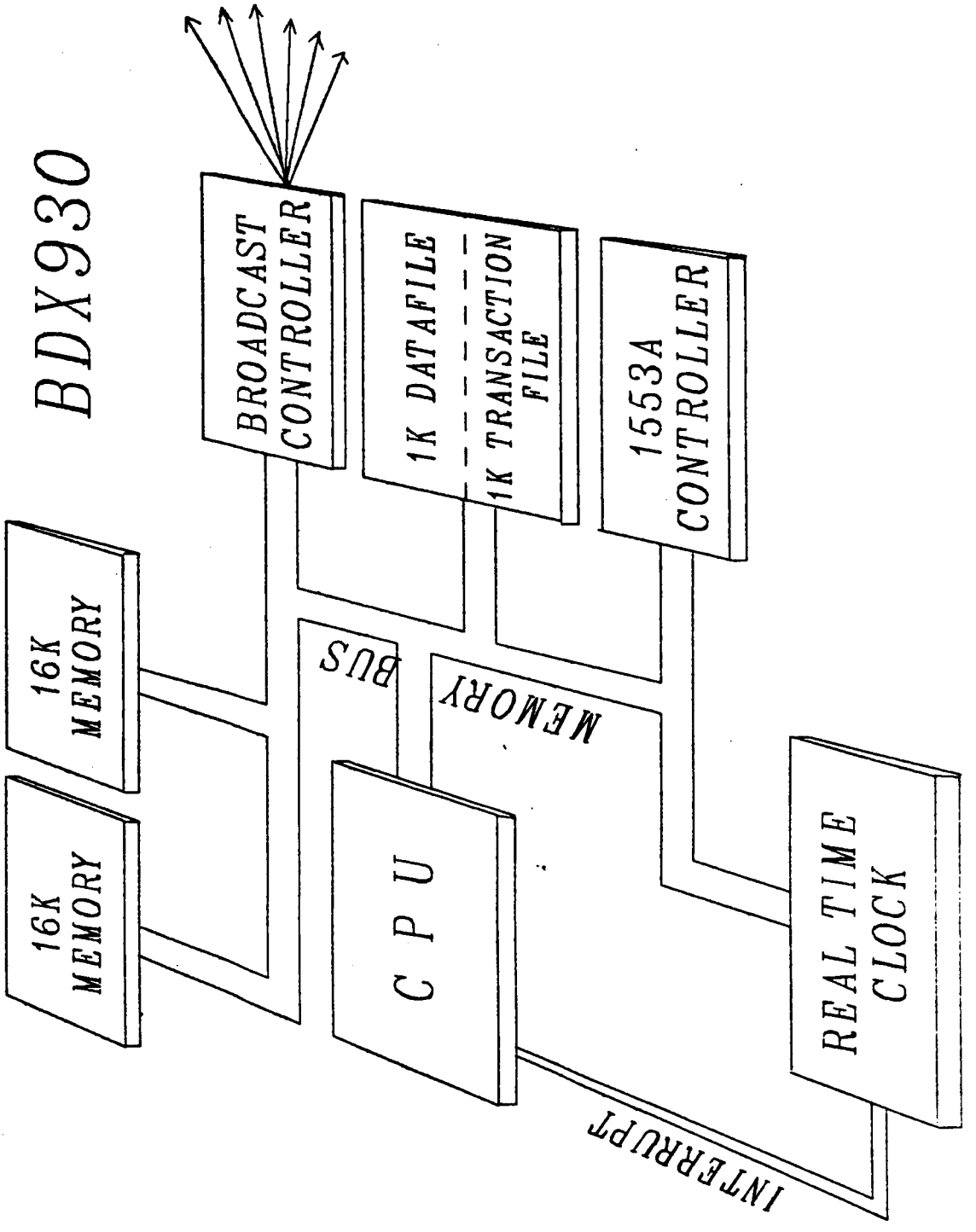


Figure 2. SIFT Processor Block Diagram

SYMBOL	LOCATION	DESCRIPTION
PIDEOF	77F8	PROCESSOR ID/ BROADCAST EOF BIT
TRANSPTR	77F9/W	WRITE TRANSACTION POINTER
ST1553A	77F9/R	READ 1553A STATUS REGISTER
	77FA	unused
CLOCK	77FB	SYSTEM CLOCK
	77FC	1553A TEST REGISTER
CMD1553A	77FD	1553A COMMAND REGISTER
	77FE	1553A T REGISTER
ADR1553A	77FF	1553A ADDRESS REGISTER

Figure 3. Memory Mapped hardware Registers

The transaction file, location 3400_{16}, is used in conjunction with the datafile to initiate a broadcast. Each data value in the datafile is associated with a destination address (mailbox offset) held in the transaction file. To broadcast a data value, the value is first stored in the datafile. The location of this value, DOFF, is represented as an offset from the start of the datafile. The desired destination mailbox offset, say MOFF, is stored in the transaction file. To start the broadcast, the value's DOFF is loaded into the transaction pointer TRANSPTR. The broadcast controller fetches the data value, appends its MOFF and transmits the result. Upon completion of the transmission, the broadcast receivers store the value at MOFF within the mailbox that is attached to that receive line. Ideally, the index to MOFF within the transaction file would equal the index, DOFF, of the data value within the datafile. Unfortunately, the hardware maps datafile address bits [9-8-7-6-5-4-3-2-1-0] into transaction file bits [8-7-6-5-4-3-2-1-0-9]. This associates datafile offsets 0 to 511 with all even transaction file offsets 0 to 1022. Datafile offsets 512 to 1023 map to odd offsets from 1 to 1023.

The broadcast transmitter signals completion through the most significant bit of register PIDEOF. Though a single transmission nominally completes in 8.6 μs, PIDEOF is signalled at 14.7 μs. This allows for worst case contention for the datafile at the receiver (see ref. 4).

2.2 The Real-Time Clock.

The real-time clock is a read/write register (referred to as CLOCK) which produces interrupts at 1.6 ms intervals. The clock consists of a 16 bit counter that is driven by the CPU's 16 Mhz crystal. The clock is therefore synchronized exactly to the fetch-execute cycle of the CPU. The least significant bit of the clock has a value of 1.6 μs.

3.0 The SIFT Operating System

The SIFT operating system is implemented in Pascal and performs the following major functions:

(1) task scheduling and dispatching
(2) data communication and voting
(3) redundancy management
(4) clock synchronization
(5) external I/O

These operating system functions fall into two categories: Local Executive and Global Executive. Items (1) and (2) are implemented in the Local Executive as procedures. The Global Executive is a set of tasks which assume the responsibility of items (3) through (5). The major distinction between the Local and Global Executives is that the Global Executive tasks exchange data and cooperate on a system wide basis, while the Local Executive procedures produce their results independently. The Local and Global Executive functions are presented in two separate sections. Each section contains a preliminary discussion of applicable data structures before the executable code is explained.

The Global Executive tasks communicate with each other and/or the Local Executive. A data structure will be considered to be part of the executive (Local or Global) that produces it. With this convention then, the ERRORS

array, which is updated in the VOTE procedure, is considered as part of the Local Executive even though it is the primary input to the redundancy management function of the Global Executive. Likewise, configuration variables such as NUMWORKING and WORKING, which may be used by Local Executive procedures, are considered to be part of the Global Executive.

3.1 The Local Executive

The Local Executive has two main responsibilities: (1) the scheduling of tasks and (2) the management of local data. The following data structures are used by the Local Executive:

(1) Task table
(2) Task schedule
(3) Buffer information table
(4) Buffer table
(5) Vote schedule
(6) POSTVOTE array
(7) ERRORS array

The data linkage between these structures is illustrated in figure 4. Only the POSTVOTE array and the buffer table are constructed completely by the operating system. The buffer table, however, is derived from information contained in the task table, task schedule and buffer information table. These structures require some, if not all of their information to be entered by the application designer. Since the SIFT operating system is driven by static data structures, an understanding of them is central to a description of the operating system.

3.1.1 Local Executive Data Structures

3.1.1.1 Task Tables (TT array)

The task table contains information specific to each task in the system. The following Pascal record defines its structure:

```
TT: ARRAY[TASKID] OF RECORD
                        CAUSE: (TASKTERM, CLOCKINT, SYSTEMSTART);
                        BUFS: INTEGER;
                        ERRORS: INTEGER;
                        STKPTR: INTEGER;
                        STATE: ARRAY[0..128] OF INTEGER;
                        END;
```

Most of these fields are initialized and managed by the operating system. Only the BUFS and STATE fields of the task table must be initialized by the applications programmer. The BUFS field points to a list of the buffer numbers in the buffer information table (described in detail below). This list defines the task's output variables. The initial value of STATE represents the task's stack as it would appear after an interrupt and holds the task starting address, terminating routine address, and initial register values. Since the task table is indexed by the variable TASKID, the order of entry into this table defines the task's TASKID. Other fields in the task table record are used by the scheduler. They contain the following information:

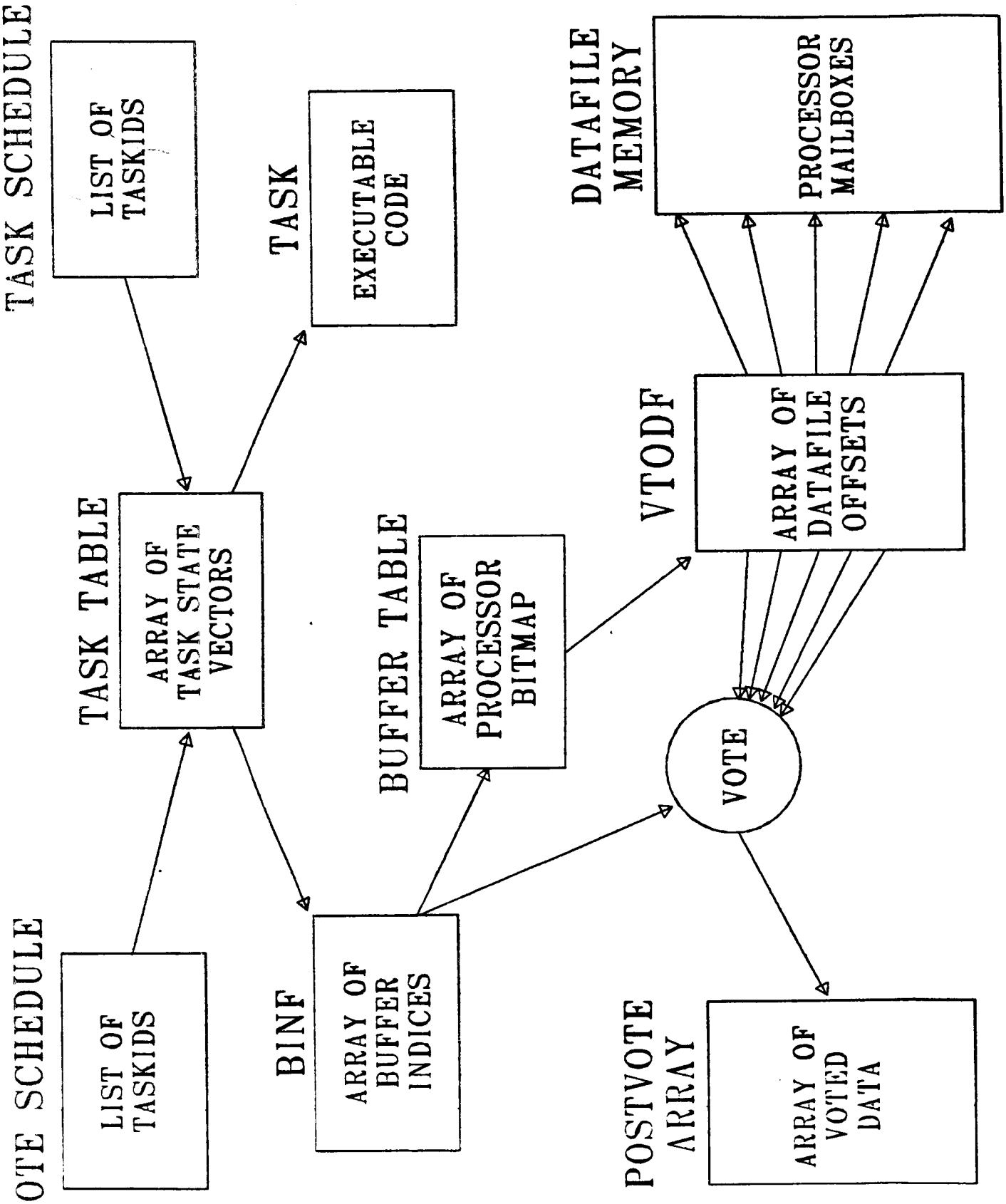


Figure 4. Linkages Between SIFT Data Structures

CAUSE - The reason for entry into the scheduler,

ERRORS - The number of times the task failed to complete,

STKPTR - A pointer to the top of the stack.

3.1.1.2 Task Schedules (SCHEDS array)

Task scheduling in SIFT is non-preemptive and is based on precalculated schedule tables. The schedule table defines the set of tasks which will be periodically dispatched. This period is called a major frame and is partitioned into 1.6 ms slots. Each task is statically allocated a subframe, which consists of a preset number of slots.

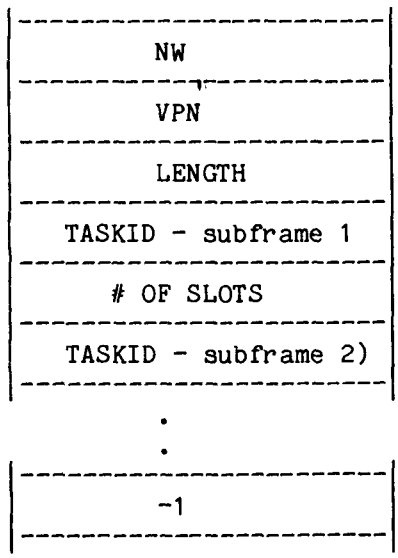
TASK SCHEDULE

SUBFRAME	SLOT	S61	S62	S63	S64	S65	S66
1	1	CLKTA	CLKTA	CLKTA	CLKTA	CLKTA	CLKTA
2	3	ICT1	ICT1	ICT1	NULLT	NULLT	NULLT
3	6	ICT2	NULLT	NULLT	ICT2	ICT2	ICT2
4	8	ICT3	ICT3	ICT3	ICT3	ICT3	ICT3
5	13	NULLT	MLS	MLS	MLS	MLS	MLS
6	15	GUIDA	GUIDA	GUIDA	GUIDA	GUIDA	NULLT
7	17	PITCH	PITCH	PITCH	PITCH	NULLT	PITCH
8	19	LATER	LATER	LATER	NULLT	LATER	LATER
9	21	ERRTA	ERRTA	ERRTA	ERRTA	ERRTA	ERRTA
10	23	NULLT	NULLT	NULLT	NULLT	NULLT	NULLT
11	25	ICT1	ICT1	ICT1	NULLT	NULLT	NULLT
12	28	ICT2	NULLT	NULLT	ICT2	ICT2	ICT2
13	30	ICT3	ICT3	ICT3	ICT3	ICT3	ICT3
14	35	NULLT	MLS	MLS	MLS	MLS	MLS
15	37	GUIDA	GUIDA	GUIDA	GUIDA	GUIDA	NULLT
16	39	PITCH	PITCH	PITCH	PITCH	NULLT	PITCH
17	41	LATER	LATER	LATER	NULLT	LATER	LATER
18	43	FAULT	FAULT	NULLT	FAULT	FAULT	FAULT
19	45	NULLT	NULLT	NULLT	NULLT	NULLT	NULLT
20	47	ICT1	ICT1	ICT1	NULLT	NULLT	NULLT
21	50	ICT2	NULLT	NULLT	ICT2	ICT2	ICT2
22	52	ICT3	ICT3	ICT3	ICT3	ICT3	ICT3
23	57	NULLT	MLS	MLS	MLS	MLS	MLS
24	59	GUIDA	GUIDA	GUIDA	GUIDA	GUIDA	NULLT
25	61	PITCH	PITCH	PITCH	PITCH	NULLT	PITCH
26	63	LATER	LATER	LATER	NULLT	LATER	LATER
27	65	RECFT	RECFT	RECFT	RECFT	RECFT	RECFT

Figure 5. Typical task assignment in SIFT

Figure 5 shows a typical task assignment for a 6 processor configuration. In this schedule, the major frame contains 66 slots in 27 subframes (the RECFT task takes two slots). The length of the major frame would then be 1.6 x 66 ms or 105.6 ms.

To support the different levels of configuration possible in the SIFT system, there must be schedules for 5, 4, 3, and 2 processor configurations (not shown). Although the Local Executive executable code and schedule table are identical on every processor, each executive uses a unique schedule contained in the schedule table. Each schedule is identified by the ordered pair (NW,VPN), where the NW field indicates the number of working processors and the VPN field is the virtual number of the processor which uses this schedule. Every physical processor has a virtual processor number assigned to it during reconfiguration. Since any processor may fail, the new virtual number cannot be pre-determined. Thus, each processor contains all the schedules. The schedules are initialized in BDX930 assembly code and have the following format:



The first two entries identify the schedule, i.e. contain the information from the (NW,VPN) ordered pair. The third field indicates the length of this section and is used for paging through the schedules. The subsequent fields contain a list of TASKIDs and the corresponding slot allocation. The end of a section is indicated by a -1. The schedule table is referenced as an array of integer:

```
SCHEDS: ARRAY[SCHINDEX] OF INTEGER;
```

The SCHEDS array also contains the Vote Schedule, which is described below.

3.1.1.3 Buffer Information Table (BINF array)

Before the buffer information table (BINF) can be constructed, the application designer enters, in BDX930 assembly code, a list of EQU instructions identifying each buffer name with a buffer number:

```
ERRER   EQU     33
GEREC   EQU     34
GEMEM   EQU     35
        .
        .
```

The BINF array is then filled with a list of buffer names to represent each task's output. A task's buffer name list is pointed to by the BUFS field of the task table. Each list is terminated by a zero field. The buffer information table is referenced as an array:

```
BINF: ARRAY[0..MAXBINF] OF BUFFER;
```

As an example, consider this portion of the BINF array.

```
        .
        .
        0
BUF1    ERRER
        0
BUF2    GEREC
        GEMEM
        0
        .
        .
```

If the Fault Isolation task's TASKID is FITID, and GEREC and GEMEM are its output buffers, then TT[FITID].BUFS is set to point to BUF2. The buffer information table is used by the SIFT operating system's initialization procedure to build the buffer table, described next.

3.1.1.4 Buffer Table (BT array)

The buffer table, BT, is the central data structure used by the system for redundancy management. This structure relates the number of working processors and TASKID to the set of processors that computes the associated task. The BT array is constructed during the SIFT operating system's initialization procedure from data in the schedule table.

The buffer table is referenced as a two dimensional array of type BITMAP:

```
BT: ARRAY[PROCESSOR,TASK] OF BITMAP;
```

The BT array is not declared as a SET OF PROCESSOR to avoid the increased overhead that would be involved in handling a set. The absence of bit manipulation instructions in the BDX930 assembly language forces the use of run time procedures to support sets in the SIFT Pascal cross compiler. The type BITMAP is defined 0..255. Bit 0 of BITMAP is associated with processor 1 and bit 7 with processor 8.

3.1.1.5 Vote Schedule (SCHEDS array)

The vote schedule's structure is similar to that of the task schedule. TASKIDs are entered in the subframe in which the task's data is to be voted. A vote schedule follows each group of task schedules associated with a

particular configuration (or number of working processors) and is thus contained in the SCHEDS array (see section on Task Schedule). The vote schedule is illustrated in figure 6.

VOTE SCHEDULE

SUBFRAME	SLOT	TASK : VARIABLES VOTED
1	1	
2	3	
3	6	ICT1 : EXPEX XRESE NDR
4	8	
5	13	ICT3 : LOCK
6	15	MLS : QX QZ QY
7	17	GUIDA: PSIN PHIN RN QDELY QLATM TIMER
8	19	PITCH: CMDEL QDELZ CMDTH QPITM
9	21	LATER: CMDAI CMDRN
10	23	ERRTA:
11	25	
12	28	ICT1 : EXPEX XRESE NDR
13	30	
14	35	ICT3 : LOCK
15	37	MLS : QX QZ QY
16	39	GUIDA: PSIN PHIN RN QDELY QLATM TIMER
17	41	PITCH: CMDEL QDELZ CMDTH QPITM
18	43	LATER: CMDAI CMDRN
19	45	FAULT: GEREC GEMEM
20	47	
21	50	ICT1 : EXPEX XRESE NDR
22	52	
23	57	ICT3 : LOCK
24	59	MLS : QX QZ QY
25	61	GUIDA: PSIN PHIN RN QDELY QLATM TIMER
26	63	PITCH: CMDEL QDELZ CMDTH QPITM
27	65	LATER: CMDAI CMDRN

Figure 6. Typical SIFT Vote Schedule.

As stated above, there is one vote schedule for each level of configuration (i.e. all processors in a configuration use the same vote schedule). The schedule assigns 0, 1 or more TASKIDs to a subframe. The buffers produced by these tasks, as defined by the BINF array, are voted before the task scheduled for that subframe is executed. The result of this vote is placed in the POSTVOTE array. The restriction of allowing only one vote schedule for each configuration level guarantees that all good processors contain exactly the same data in the POSTVOTE array -- even if its schedule does not execute tasks which use all the data. Although this appears wasteful, it protects the fault isolation process. By having all processors vote all data, the processors' error reports will more accurately reflect the system's status. Also, since all data is available on every processor during reconfiguration, it is not

necessary to transfer data to a processor when its new schedule contains a task it previously had not executed.

3.1.1.6 POSTVOTE Array

The POSTVOTE array contains the result of the vote procedure.

```
POSTVOTE: ARRAY[BUFFER] OF INTEGER;
```

All processors must vote every task's data in order to maintain an accurate POSTVOTE array. This restriction simplifies the reconfiguration process as explained in section 3.1.1.5 Vote Schedule. An additional design guideline requires that a task use only temporary variables during execution. If any static storage is necessary, the task should define the data as output. The data is then retrieved during the next iteration through the POSTVOTE array.

3.1.1.7 ERRORS Array

The ERRORS array is the Local Executive's link to the Global Executive.

```
ERRORS: ARRAY[PROCESSOR] OF INTEGER;
```

The Local Executive maintains a count of processor errors in the ERRORS array. The Global Executive task, ERRTASK, uses this data to produce its error report.

3.1.2 Local Executive Procedures and Functions

A listing of the Local Executive data structures, procedures and functions is included in module Pascal SIFTOP.MCP. In addition to this Pascal code, assembly code found in modules SIFTIH.SR and SCHEDULE.SR is used to support Local Executive functions. The following sections will describe the procedures and functions invoked during system startup and initialization, clock interrupts and scheduling, voting, and inter-task communication.

3.1.2.1 System Startup and Initialization.

The first section of module SIFTIH.SR is set to absolute addressing mode to allow definition of the starting address, 100_{16}, and interrupt vectors. There are two active interrupts in the SIFT system: the power fail interrupt at 400_{16} and the clock interrupt at 401_{16}. The only other functions performed in absolute addressing mode in SIFT is the initialization of the transaction file at 3400_{16}, and the datafile at 7400_{16}.

Beginning at 100_{16}, the CONT instruction disables interrupts and sets FLAG 1. FLAG 1 is significant because it controls whether the BDX930 performs saturated or unsaturated arithmetic. SIFT requires unsaturated, i.e. unsigned, arithmetic and therefore FLAG 1 is set. After the CONT instruction, an indirect branch is taken to the relative addressing portion of the assembly code where the stack is defined in R15 as starting at 5000_{16}. The remaining registers are cleared before the INITIALIZE routine is called. Upon return from INITIALIZE, interrupts are enabled and a wait loop is entered.

3.1.2.1.1 PROCEDURE INITIALIZE.

Procedure INITIALIZE is located in module SIFTOP.MCP. In this procedure, the POWER, BT and TT arrays are initialized, the initial configuration is set and initial synchronization is obtained. These functions shall be described in the order of their execution.

3.1.2.1.2 PROCEDURE GPROCESSOR.

Procedure GPROCESSOR reads the processor's physical id number from PIDEOF. The processor's id is hardwired into the most significant 4 bits of this register. The value in PIDEOF must then be shifted to the right 12 bits to obtain the PID.

3.1.2.1.3 PROCEDURE DBADDRS.

Procedure DBADDRS constructs array DBAD. Array DBAD maps processor PIDs into indices to their mailboxes within DATAFILE.

3.1.2.1.4 PROCEDURE WORK.

Procedure WORK determines which processors are in the initial configuration. This is indicated by the receipt of the processor PIDs in buffer R_0. Array element WORKING[P] is set to true if processor P PID is received. (See section on Global Executive data structures for a description of WORKING.)

3.1.2.1.5 PROCEDURE SYNCH.

Procedure SYNCH establishes initial system synchronization. The processors wait for the highest numbered working processor to broadcast a synchronization value. This procedure was necessary when the SIFT was hosted on the ECLIPSE 250 computer. The current host system starts the processors simultaneously, thereby eliminating the need for this procedure.

3.1.2.1.6 Construct POWER2 Array.

Due to the lack of an EXP function in the cross compiler and the need to set certain bits in the BT array, the POWER2 array is constructed, where

```
POWER2[I] := 2 EXP I;
```

3.1.2.1.7 PROCEDURE RECBUF(NW,P: PROCESSOR; S: SCHINDEX).

For a given level of configuration and TASKID, the BT array will return a value indicating which processors computed the task. This information is reduced from the task schedules by procedure RECBUF. To accomplish a complete initialization of BT, RECBUF must be called for every schedule as defined by the ordered pair (NW,P) and the index into SCHEDS, S.

In procedure RECBUF, each TASKID, T, within schedule (NW,P) is extracted and used as an index to the BT array. The bit corresponding to the virtual processor which uses this schedule, POWER2[P], is then set by a "boolean or" operation (BOR) into BT[NW,T]. The reasons for excluding the null task, NULLT, from this operation probably comes from some requirement in a previous version and is now unnecessary.

3.1.2.1.8 Resynchronizing.

Writing the BT array is expected to take a relatively long time, i.e. greater than 25 ms. The SYNCH procedure is called once again to insure that the processors don't drift too far apart, i.e. greater than 200 μs.

3.1.2.1.9 Presetting Some Variables, the POSTVOTE and TT Arrays.

Variables PRESENTCONFIG and RECONF which are used during reconfiguration are cleared. The global frame, major frame, and subframe counters, GFRAME, FRAMECOUNT, and SFCOUNT respectively, are initialized. SFCOUNT is set to MAXSUBFRAME to insure that the first time the SCHEDULER is executed, the task schedule pointers are reset to the beginning of the schedule.

The POSTVOTE array is cleared and the TT array is initialized for all tasks.

3.1.2.1.10 PROCEDURE BUILDTASK(TASKNAME: TASK).

Procedure BUILDTASK calls assembly procedure REINIT, to initialize the STATE and STKPTR fields of the TT record.

3.1.2.1.10.1 PROCEDURE REINIT(VAR STACK: INTEGER; VAR V: STATEVECTOR).

Procedure REINIT is found in module SIFTIH.SR. The STATEVECTOR, which is the memory pool within the task table for the STACK, had its first entry set to the task start address in module SCHEDULE.SR. The remaining entries are now cleared. REINIT sets the STATEVECTOR (stack) to look as if the task had been called by termination routine TTERM and then interrupted, with the exception that the task resume address is instead the task start address. With this structure, the SCHEDULER can treat the scheduling of each task as a resume operation. When a task exits, the termination routine, TTERM, restores the STATEVECTOR to this format. Registers R14 and R15 are not saved because they are the heap and stack pointer, respectively. As of this writing, SIFT does not use the heap. REINIT returns STACK set to point to the location 18 within the STATEVECTOR. This will be the top of the stack when the task is started.

0	Task start address:	supplied in module SCHEDULE.SR
1	Termination address:	supplied by REINIT as TTERM
2	R0	
3	FLAGS	
4-16	R1 - R13	
17	Task resume address:	supplied by REINIT as start address
18	<Top of Stack>	

3.1.2.1.11 Establish Initial Configuration.

The ERRORS array is cleared and a temporary variable, RECONF, is constructed based on the WORKING array to reflect the starting configuration. The POSTVOTE value representing buffer GEXECMEMORY is set to equal RECONF. Buffer GEXECMEMORY is used by the Global Executive to protect against a transient disturbance that could cause an otherwise good processor to be reconfigured out of the system (see section 3.2.3.3 FAULTISOLATIONTASK). Function XRECF is called to bring the new configuration into effect. XRECF is a Global Executive function and is explained in detail in section 3.2.3.4.

3.1.2.1.12 Initializing Modules SIFTAP and SIFTIC.

Procedures APPINIT and ICINIT are called to perform initialization procedures in module SIFTAP.MCP and SIFTIC.MCP respectively. SIFTIC contains the interactive consistency tasks which are discussed as part of the Global Executive in section 3.2.2.

3.1.2.2 Clock Interrupts and Scheduling.

Clock interrupts occur every 1.6 ms and are vectored through location 401_{16}. An indirect jump is made through location ACINT to the clock interrupt handler. To accommodate the restriction of one word instructions for interrupt vectors, the jump must be through page zero. Location ACINT is therefore defined by a DEFPZ (define page zero) pseudo-instruction. Because the DEFPZ instruction requires the address of the interrupt handler as an argument, this instruction must occur after the interrupt handler (CINT). The effect of the JMA0 instruction is to push R0 onto the stack and save the resume PC in R0.

The clock interrupt handler checks for the end of a subframe by incrementing the repeat counter, RPCNT, and testing for zero. The current task is resumed if RPCNT is non-zero. Otherwise the remaining registers and resume address are saved and the stack is switched to the executive stack, 5000_{16}. The scheduler is called with the interrupted task's stack pointer and a value (=1) indicating a clock interrupt has occurred.

3.1.2.2.1 FUNCTION SCHEDULER(CAUSE: SCHED_CALL; STATE: INTEGER): INTEGER.

The scheduler is called to either terminate a task (CAUSE = TASKTERMINATION) or schedule a new task (CAUSE = CLOCKINTERRUPT). Although CAUSE has a third value, SYSTEMSTARTUP, it is never used. At system start, the SCHEDULER is entered for the first time with CAUSE = TASKTERMINATION and TASKID = 0, i.e. zero task. When a task terminates, the SCHEDULER substitutes the null task, NULLT, for the active task for the remainder of the subframe.

To service a clock interrupt, the SCHEDULER must first check to see if the current task is interruptable. In the current version of SIFT only the null task and zero task should be interrupted. If any other task is interrupted, that task's TT.ERRORS field is incremented and its STATEVECTOR rebuilt. As an aid to debug, procedure PAUSE is called with the hexadecimal argument BADn, where n is the TASKID. PAUSE places this "error message" in register R1 and halts the processor.

Continuing with the processing of the clock interrupt, the frame counters, SFCOUNT, FRAMECOUNT and GFRAME, are serviced and the schedule pointers, VP and TP, are reset if necessary. Global variables TPI and VPI are defined during the reconfiguration process and represent the beginning of this processor's task and vote schedules, respectively, for the current configuration.

3.1.2.2.2 PROCEDURE TSCHEDULE.

Procedure TSCHEDULE loads the new TASKID and repeat counter, RPCNT, from the task schedule. The repeat counter controls the number of 1.6 ms slots in the next subframe and therefore, limits the amount of CPU time the new task can

take. Global variable TP is maintained pointing to the next task schedule entry. If the end of the task schedule is reached in fewer than MAXSUBFRAMES, procedure TSCHEDULE schedules the null task for 3.2 ms.

3.1.2.2.3 PROCEDURE VSCHEDULE.

Procedure VSCHEDULE initiates the voting of task output data. At index VP within the vote schedule will be either a list of TASKIDs or a -1 indicating the end of the vote schedule. Procedure VOTE is called to perform the vote function. A default value of -1 is placed in the POSTVOTE array for each buffer which did not have a majority value.

3.1.2.2.4 Activating the Task.

On completion of TSCHEDULE and VSCHEDULE, a new task has been selected (TASKID updated according to the task schedule) and all data voted (according to the vote schedule). The final function SCHEDULER performs is to retrieve the new task's stack pointer and return this value to the assembly code routine. Upon return from function SCHEDULER, R12 holds the new task's stack pointer value. The switch is made to the task's stack, its registers and flags are restored, and the task is resumed.

3.1.2.3 Voting.

The vote schedule contains a list of TASKIDs for each subframe. Procedure VSCHEDULE removes one TASKID at a time from the list and passes this value to procedure VOTE. Procedure VOTE calls either VOTE3 or VOTE5 (depending on the replication level of the task) to vote the output data buffers associated with the task. If any errors are detected, procedure ERR is called to increment the offending processor error count. If a majority value is not found, procedure FAIL is called to increment all the processor error counts. A default value is substituted for the majority value in such cases. The default value is -1 for all data.

3.1.2.3.1 PROCEDURE VOTE(TK: TASK; DEFAULT: INTEGER).

Procedure VOTE has two parameters, the id of the task to be voted, TK, and the default value to be used if a vote fails, DEFAULT. Procedure VOTE must first determine on which processors the task was executed before the task's data can be retrieved and voted. Variable K is loaded with the the processor bitmap contained in the BT array. Variable I is kept as a virtual processor number. The status of the least significant bit of K is tested by function ODD, a 1 indicating that processor I computed task TK. Variable J accumulates the replication count for task TK. Because the ERRORS array must be associated with physical processor numbers, I is translated to a physical processor number and stored in PREAL. Depending on the replication level, J, P'J' is load with PREAL and D'J' with the datafile mailbox offset. Processor bitmap, K, is shifted right by a divide by 2 and the next virtual processor is tested for task TK.

After the replication level, J, of TK has been determined and all the P'J' and D'J' loaded, the output data buffers of task TK are voted and the result stored in the POSTVOTE array. Variable LBUFS is set to point to the task's list of data buffers found in BINF, i.e. LBUFS = TT[TK].BUFS. Variable B is

set to the first buffer number. The replication level determines the method of voting, i.e.

```
If J < 0      - DEFAULT is used
If 0 < J < 3  - Processor P1's value is used
If 3 ≤ J < 5  - Procedure VOTE3
If J ≥ 5      - Procedure VOTE5
```

Global variables V1..V5 are loaded with the replicate data. Global variables are used to reduce the procedure call overhead.

After a data buffer is retrieved, voted and stored in the POSTVOTE array, the local processor replaces its representation of the data, found in the last mailbox, with the voted value. The only known reason for this procedure is that interactive consistency task 1 (ICT1) outputs data directly from the DATAFILE without first copying the voted values from the POSTVOTE array.

Finally, pointer LBUFS is incremented and buffer number B updated. The vote continues until no buffers remain, i.e. B = 0.

3.1.2.3.2 FUNCTION VOTE3(DEFAULT: INTEGER): INTEGER.

In function VOTE3 a three-way vote is performed on the data found in variables V1, V2 and V3. A majority value is found when two values agree, therefore one fault can be tolerated. If a value miscompares with the majority value, procedure ERR is called with the associated processor number, P1..P3. If a majority value cannot be found, the default value is substituted and the error counts of all three processors are incremented.

3.1.2.3.3 FUNCTION VOTE5(DEFAULT: INTEGER): INTEGER.

Function VOTE5 performs a five-way vote on global variables V1..V5. A majority is found when three values agree, therefore two faults can be tolerated. If a value miscompares with the majority value, the associated processor's error count is incremented. If a majority value cannot be found, procedure FAIL is called to increment the error count of all processors, i.e. P1..P5.

3.1.2.3.4 PROCEDURE ERR(P: PROCESSOR).

Procedure ERR increments the error count of processor P. The error count is kept in array ERRORS.

3.1.2.3.5 PRCEDURE FAIL.

Procedure FAIL is used in VOTE5 to increment the error counts of all five processors.

3.1.2.4 Inter-task Communication.

Inter-task communication in SIFT involves the use of Local Executive procedures to physically broadcast the data and then, during some other task, to retrieve it via the vote subsystem. The entire process depends on the correctness of the preset tables, schedules and arrays. Data transfer is done

for other processors performing ICT1 to acknowledge communication with the simulation.

3.2.1.2 Constants CLK_BUF and CLK_TRANS.

CLKTASK performs its own broadcast. To broadcast its clock value, a processor uses buffer R_0. The DATAFILE offset is 7*128 or 896 = TPBASE. The TRANSFILE index associated with this buffer is 2*896 - 1023 = 769. Constant CLK_TRANS is set to this value. To set up the broadcast TRANSFILE[CLK_TRANS] must be loaded with the mailbox offset of R_0, = 0, marked with the EOFBIT, 8000_{16}. Constant CLK_BUF is set to 8000_{16}.

3.2.1.3 Constant COMMDELAY.

Constant COMMDELAY represents the average delay of a clock transmission, 24_{16} clock ticks, or 38.4 μs.

3.2.1.4 Constant OMEGA.

Constant OMEGA is the skew limiting factor. The optimum value for OMEGA has been shown to be 134_{16} clock ticks, or 209 μs (ref. 7).

3.2.1.5 Array SKEW.

Array SKEW is used to hold the clock skews calculated during each processor's window. SKEW is declared globally to avoid compiler errors which result when too many local variables are used.

```
SKEW: ARRAY[PROCESSOR] OF INTEGER
```

3.2.1.6 The Body of CLKTASK.

CLKTASK begins by calling procedure DISABLE to disable the clock interrupts. This is necessary to preclude clock interrupts during the exchange of clock values. If clock interrupts were allowed, greater variance would be added to the communications delay and, therefore, greater error to the clock reading. Since there are 8 windows of 250 μs each, a clock interrupt would definitely occur during one of the windows.

Next all R_0 buffers are cleared in preparation for the clock exchange. This is probably unnecessary. The transaction file is initialized. This need only be done once.

The clock exchange is then performed. Beginning each window, the corresponding processor's SKEW is set to zero and the window start time is loaded into variable WINDOW. Global variables are needed when referencing the clock to prevent the compiler from optimizing away the desired calculation. If the current window is the local processor's broadcast window, the processor enters a tight loop where the clock is repeatedly read and broadcast.

The accuracy of the broadcasted clock value depends on how quickly the receiving processor recognizes the arrival of the new clock value. The receiving processor reads the initial value of the broadcasting processor's R_0 buffer (the broadcaster may be ahead of the receiver and already have broadcast clocks) and then loops until a different value arrives. The initial

clock value is kept in PCLOCK, the new clock value in CCLOCK, and the processor's own clock at that time in ACLOCK. When a new clock value arrives, the SKEW, including the COMMDELAY, is calculated. The granularity of the loop within the windows can insert additional skew between the processors. To reduce this effect, once a clock value has been received, the processor drops into a tighter loop waiting for the end of the window.

After all processor clocks have been exchanged, the clock correction is calculated. For all working processors, their associated terms in SKEW are summed. If the SKEW value is greater than or less than OMEGA its SKEW is taken as zero. The clock correction, DELTA, is calculated as the average of the SKEW terms. The clock correction, DELTA, is then applied to the clock.

3.2.1.7 A Lesson in Disabling Interrupts and Malicious Liars.

Before exiting CLKTASK, the interrupts are enabled. There are two problems with disabling the interrupts during a task. First, a task may take longer than indicated by the number of slots allocated to it in the schedule. This would occur, for example, if the interrupts are disabled for longer than 3.2 ms., i.e. more than 2 slots. When the interrupts are re-enabled one, not two, interrupts would result. The second problem arises when a task has a variable execution time and enables the interrupts close to the occurrence of an interrupt. Take, for example, a task that normally takes just under 3.2 ms to execute. After the interrupts are enabled, the first interrupt is counted immediately. The second interrupt is counted shortly thereafter. Now suppose that due to the variance in the task's execution, it takes just over 3.2 ms to execute. When the interrupts are enabled, only one interrupt is counted. The second interrupt registers 1.6 ms later. The task actually takes 4.8 ms.

While this deviation from design specification is a problem, another effect is more insidious. If the variance in task execution is dependent on output from another task, a "malicious" processor can cause one processor to stretch its subframe the extra 1.6 ms, while not affecting the others. This skew injected into an otherwise "good" processor will eventually cause the "good" processor to be reconfigured out of the system. This scenario is not far fetched. Every processor's execution time is dependent on the other processors' data due to the vote error processing. If a processor sends bad data to only one processor, only that processor will record the error. The extra time spent recording the error will delay the start of the next task on that one processor.

This behavior was actually observed on the SIFT system. The baseline operating system originally disabled the interrupts during scheduling, and therefore during voting. In one subframe in which six values were voted, interrupts were enabled just before the 3.2 ms interrupt. One processor developed an intermittent fault which affected the broadcast transmission to only two other processors. During normal system operation, the fault in the one processor would cause the other two processors to be configured out of the system! The clock task is especially susceptible to this failure mode due to the dependence of the receive window length on whether or not the transmitting processor's clock is seen. BEWARE!.

3.2.2 The Interactive Consistency Tasks.

Interactive consistency, as implemented in SIFT, takes place in three discrete steps, performed by three tasks: ICT1, ICT2 and ICT3. First, in task ICT1, external data is input and distributed. Next, in task ICT2, the data is redistributed so that all processors have at least three copies of the data. Finally, in task ICT3, the three copies are voted to produce one consistent value for the data throughout the system. A significant feature of the interactive consistency algorithm is that the processor that produces the data originally, i.e. runs ICT1, should not participate in the redistribution of the data, i.e. run ICT2. This is necessary to protect the system from the effects of a "malicious" liar (see ref. 8). In SIFT, three redundant channels are sampled by three processors executing task ICT1. Four processors execute task ICT2. Task ICT2 is designed in such a way that if a processor is scheduled to run both ICT1 and ICT2, the processor will not redistribute data it produced when running ICT1. This rule is relaxed if there are three or less processors in the configuration. All processors execute task ICT3 to vote the data.

Data is input and distributed during ICT1 to input the "A", "B" or "C" buffers (see section 3.2.2.1.1), depending on which redundant channel the processor is sampling. For example, processor 1 executes ICT1 and, since it samples channel 1, will broadcast its data to the A buffers. Processors 2 and 3 will use the B and C buffers. The simulation expects ICT1 to execute on the first three working processors. During ICT2, processor 1 would rebroadcast the B and C buffers, processor 2 A and C buffers, processor 3 the A and B buffers and processor 4 the A, B and C buffers. All processors should at this point contain three copies of the A, B and C buffers. It remains for task ICT3, which runs on all processors, to vote the three copies of the A, B, and C buffers. The voted results are kept in the POSTVOTE array. To retrieve a data value, an application tasks invokes the MEDIAN function (see section 3.1.2.4.5). The MEDIAN function will return the mid-value of the A, B, and C buffers found in the POSTVOTE array.

3.2.2.1 The Data Buffers of the Interactive Consistency Tasks.

The interactive consistency tasks have no prominent data structures. There are, however, several data buffers used during interactive consistency.

3.2.2.1.1 The "A,B,C" Input Buffers.

The interactive consistency tasks are designed to communicate with triply redundant external devices. The current simulation provides 21 input variables: ALPHA, BETA, ..., YCNTR. The three sets of data are kept in buffers AALPHA, ABETA, ..., AYNCTR; BALPHA, BBETA, ... BYNCTR; and CALPHA, CBETA, ..., CYCNTR (see module SIFTDEC.CON). The "A" series use buffer numbers 40 thru 60. The "B" series use 61 thru 81, and the "C" series use 82 thru 102. This data is eventually processed as input to the application tasks.

3.2.2.1.2 The "O" Buffers.

The application tasks' output resides in buffers OCMDAIL, OCMDELE, ..., OPITMO, buffers 103 thru 110. A data value representing the current configuration is output in buffer ORECONF, 111, to keep the simulation

informed of which SIFT 1553 channels can provide valid data. The application tasks are iterative and must be kept synchronized to the correct iteration in the simulation. Buffer OSYNC, 112, is used for this purpose. SIFT sends an iteration count in OSYNC with new data. SIFT then expects the simulation to return that value incremented by 1, indicating that input data for the next iteration is ready. The simulation expects SIFT to return the iteration count with the control response.

3.2.2.1.3 Buffers EXPECTED and XRESET.

If the simulation's next iteration of input data is not ready at the time SIFT requires it, SIFT continues processing with random data. The synchronization word expected from the simulation is saved in buffer EXPECTED to comply with the design rule that tasks do not use permanent storage. The expected synchronization word is voted and retrieved by the next iteration of the interactive consistency tasks.

The application tasks perform an auto-land function. Buffer XRESET is used to reset the application tasks to the proper mode whenever the simulation signals that the auto-land procedure is to begin.

3.2.2.1.4 Buffers LOCK and NDR.

As mentioned above, it is entirely possible that a new set of input data will not be available from the simulation when SIFT requires it. Since SIFT cannot conveniently wait for the data (this would disrupt measurements of the system's performance), a provision has been made to provide random data to the application tasks whenever "real" data is not available. When random data is used, the tasks' current set of output must be saved since some of the data will be used to continue the integration functions found in the control algorithms during the next iteration. Also, the resulting random output must not be sent to the simulation.

Buffer NDR is used to communicate to all processors the status of the input data, i.e. NDR = 1 if real data is available. During interactive consistency task 3, if real data is not ready, the tasks' output data is saved in a temporary array (TEMPVOTE). Buffer LOCK signifies that random data has been processed and that the output function should not be done. These functions are only necessary when interfacing to the simulation and would not be required in an actual aircraft system.

3.2.2.2 GLOBAL FUNCTION ICT1.

Task ICT1 performs the input/output to the external devices on the 1553 bus. This function is complicated by the fact ICT1 must synchronize with the simulation. ICT1 outputs data according to the state of buffer LOCK. It determines if new data from the simulation is ready and supplies random data if not. The status of the data, random or real, is communicated to ICT3 thru buffer NDR. The following procedures are used during task ICT1.

3.2.2.2.1 FUNCTION RANDOMIZE(SEED: INTEGR): INTEGER.

Function RANDOMIZE computes a pseudo-random number based on the value of SEED. The function used is standard for a 16 bit machine.

3.2.2.2.2 PROCEDURE COMUN1553A(ADR,N,SA,MODE,RT: INTEGER).

Procedure COMUN1553A sends and receives data on the 1553 bus. The parameters of COMUN1553A are used to construct command words for the 1553 controller. The parameters are:

ADR: The starting address of the data. Must be within the DATAFILE, i.e. 7400_{16} to $77FF_{16}$. Typically I/O is done from the last mailbox, 7780_{16} to $77FF_{16}$.

N: The number of words to transfer. Up to 32 words may be transferred.

SA: The sub_address of the device. Each remote terminal can have 32 sub-addresses. The sub-address is inserted in bits 5 thru 9 of the command word. To address sub_address 1, SA = 20_{16}. I/O is sent to sub-address 0; synchronization data to sub-address 1.

ME: Indicates whether data is to be transmitted or received. MODE = 0 for transmit, 400_{16} for receive.

RT: The remote terminal address. Up to 32 remote terminals can be addressed on the 1553 bus. The remote terminal address is inserted into bits 11 thru 14. Remote terminal 1 is then addressed as 800_{16}.

COMUN1553A begins by constructing the command word (= N + SA + MODE + RT). The address (ADR) and command word are then loaded into their respective controller registers (ADR1553A and CND1553A, see figure 3). Procedure WAIT1553A is called to wait for the controller to signal completion. If an error occurred, the operation is retried. If no error, COMUN1553A waits an amount of time equal to a retry.

3.2.2.2.3 PROCEDURE GETNDR.

Procedure GETNDR communicates with the other processors running ICT1 to determine if the simulation's next iteration of data is ready to be transferred to SIFT, i.e. if new data is ready. The processors running ICT1 use the R_0 buffer to indicate whether or not they received the expected synch word. GETNDR begins by clearing the R_0 buffers, a 1 being a positive indication. The synch word is then input over the 1553 bus. If it is equal to the expected value or RESET then the processor's R_0 buffer is set to 1. Buffer R_0 is then broadcast. GETNDR must then wait for a time greater than the maximum skew that can be expected between the processors before the R_0 buffers of the other processors can be checked. This time is equivalent to the MAX_WINDOW time used by the clock task, see section 3.2.1.1. Before entering the time-out loop, the processor's input buffer area is chosen, i.e. the A, B, or C buffers. Variable INDEX, which is set to point to the chosen buffer area, is declared globally to avoid the overhead of passing the value as a parameter.

3.2.2.2.4 PROCEDURE GETREALDATA.

Procedure GETREALDATA is called once a consensus has been reached that new input data is available from the simulation. The synchronization word is input again since this processor may not have received a valid copy, i.e. the two other processors may have agreed that the new data was ready. If the synchronization word indicates a reset operation, buffer XRESET is marked to

alert the application tasks and the expected synchronization word, EXPNDR, is set to the reset value, -1. The new data is then input. Global variable INDEX which is used in the call to COMUN1553A has been previously set in procedure GETNDR to point to either the A, B or C buffer. Buffer NDR is set to a 1 to indicate that new data has been loaded.

3.2.2.2.5 PROCEDURE GETRANDOMDATA.

If synchronization with the simulation was not attained, random data is substituted so processing can continue. Buffer XRESET is set to zero. A reset is unnecessary while processing random data. Variable EXPNDR is restored to its previous value (see section 3.2.2.2.8). The seed for the function RANDOMIZE is calculated as the total subframe count. The input buffers are then filled with random data. Finally buffer NDR is set to 0, indicating new data is not ready.

3.2.2.2.6 PROCEDURE GETNEWDATA.

Procedure GETNEWDATA is called after GETNDR. Sufficient time has therefore elapsed since the "new data ready" query began, and the results can now be tested. GETNEWDATA tests the R_0 buffers of all working processors. If buffer R_0 is equal to 1 then variable READY is incremented. New data is available if READY $\geq$ 2, i.e. 2 of the 3 processors executing ICT1 agree that new data is ready. If NUMWORKING = 1 the local processor's indication is sufficient. Procedure GETREALDATA is called if new data is ready, GETRANDOMDATA if not.

3.2.2.2.7 PROCEDURE DISTRIBUTE.

Procedure DISTRIBUTE utilizes the broadcast bus's ability to transmit more than one data value at a time. The procedure is similar to broadcasting a single value except that the EOFBIT is set on the last element only. DISTRIBUTE begins by loading the transaction file with the destination offsets for all input buffers. The offsets are left-shifted three places, i.e. multiplied by eight, as required by the hardware. A call to WAITBROADCAST insures the broadcast bus is not busy. The EOFBIT of the last element of the transaction file is then marked. Setting PIDEOF to zero enables multiple transmissions. The TRANSPTR register is then loaded with a value equivalent to INDEX to begin the broadcast. DISTRIBUTE waits until the broadcast is complete.

3.2.2.2.8 The Body of ICT1.

Task ICT1 begins by loading variable EXPNDR with the expected synchronization value. Buffer EXPECTED is initialized in procedure ICINIT. If the outputs are unlocked, i.e. LOCK = 0, then new output data is sent to the simulation along with the synchronization value. EXPNDR is set to the next expected value. If new data is not ready this iteration, the former value of EXPNDR must be restored. Variable OLDEXPECTED holds the original value of EXPNDR. If necessary, EXPNDR is restored in GETRANDOMDATA.

Procedure GETNDR establishes the value of the simulation's synchronization word. Procedures GETNEWDATA and DISTRIBUTE input new data and broadcast it to the other processors. The next iteration's synchronization value is saved in buffer EXPECTED.

3.2.2.3 GLOBAL FUNCTION ICT2.

Interactive consistency task 2, ICT2, rebroadcasts the data that was input and distributed by interactive consistency task ICT1. Task ICT2 performs a key function of the interactive consistency algorithm, i.e. it determines the source of the input data replicates (i.e. which processors ran ICT1) and rebroadcasts only that data which the host processor did not produce. Given 3 ICT1 replicates, at least four processors will be needed to do the rebroadcast correctly. Once ICT2 completes, each processor in the system should have 9 copies of the input data, each data value from the 3 input channels replicated 3 times.

3.2.2.3.1 PROCEDURE REBROADCAST(VPX,P: PROCESSOR).

Procedure REBROADCAST broadcasts the 1553 data pointed to by parameters VPX and P. VPX has a value 0, 1, or 2 depending on whether the A, B, or C buffers are to be broadcast. The buffers are located in processor P's mailbox. REBROADCAST contains a straightforward loop which transfers the data from processor P's mailbox to the host processor's output mailbox. The corresponding transaction file locations are loaded during the transfer. Once the broadcast bus becomes available, the EOFBIT is marked in the last data value and the broadcast is initiated.

3.2.2.3.2 The Body of ICT2.

Task ICT2 begins by fetching the bitmap of those processors which executed task ICT1. Variable IC1V contains the processor bitmap. Variable IC1P maintains the virtual processor count as IC1V is interrogated. Variable VPX will assume a value of 0, 1, or 2 depending on whether the A, B, or C, buffers are to be broadcast. Buffers A, B, and C were produced by virtual processors 1, 2, and 3 respectively. If IC1V is odd, then the virtual processor indicated by IC1P executed task ICT1. If less then 3 sets of buffers have been rebroadcast, virtual processor IC1P's real processor number is compared with the host processor's PID. If they are not equal, i.e. the host processor did not produce the data as indicated by VPX, then procedure REBROADCAST is called to broadcast the 1553 data located in processor P's mailbox. This process continues until all working processors are tested.

3.2.2.4 GLOBAL FUNCTION ICT3.

Task ICT3 votes the data replicates produced by task ICT2. If 3 or more processors are in the configuration, there should be 3 replicates of each data item in the A, B, and C buffers. To locate this data ICT3 first determines which virtual processors executed ICT1. The A, B, and C buffers were produced by virtual processors 1, 2, and 3 respectively. Next, the processors which executed the ICT2 task which rebroadcast the ICT1 data must be found. Valid replicates are found in these processor mailboxes. Task ICT3 is also responsible for saving and restoring data that would be corrupted while outputs are locked.

3.2.2.4.1 PROCEDURE GETIC2PROC(IC1P: PROCESSOR).

Procedure GETIC2PROC finds the 3 processors which rebroadcast data produced by processor IC1P when it executed ICT1. This set of processors is kept in

global array VP. Variable IC2V contains the bitmap of processors which produced ICT2. Variable IC2P maintains the virtual processor number and variable REP the number of replicates found. GETIC2PROC will search for 3 replicates. Beginning with virtual processor 1, IC2V is tested until the next processor which produced ICT2 is found. This processor is made part of the set of processors kept in array VP if IC2P is not equal to IC1P or there are just 3 processors in the configuration. Although processor IC2P did not rebroadcast data it produced, the data is in the correct location and can be used if there are only 3 processors in the configuration.

3.2.2.4.2 PROCEDURE VOTEDATA(DB: INTEGER).

Procedure VOTEDATA retrieves and votes replicated 1553 data. Parameter DB indicates which of buffer set (A, B, or C) is to be voted. Global array VP contains the virtual processor numbers associated with the mailboxes which contain the data replicates. Variable BASE is an offset to the buffers to be voted within the mailbox. Indexing through the data set, variable NB becomes the offset to the particular data item. A deeply nested array access is needed to keep the compiler honest when the pair (VP[i],NB) is translated into the corresponding data values V1, V2, and V3. A straightforward vote follows. The majority value is stored in the POSTVOTE array. If a majority is not reached, the processor halts with a value $C3_{16}$ in register R1.

3.2.2.4.3 PROCEDURE RESTORE.

Procedure RESTORE manages the storage and retrieval of the temporary data. Procedure RESTORE has 2 states. If new data is ready and outputs were locked, then outputs are unlocked and the temporary data is restored. If new data is not ready and outputs were unlocked, then outputs are locked and the temporary data is stored.

3.2.2.4.4 The Body of ICT3

Task ICT3 begins by loading variable IC1V with the bitmap of processors which executed ICT1. Variable IC1P is the virtual number associated with the processor that computed ICT1. Variable DB takes on values 0, 1, and 2 for 1553 buffers A, B, and C respectively. If there are 3 or more processors in the configuration, the virtual number of the processor which produced the data associated with the current value of DB is extracted from IC1V and held in IC1P. Procedure GETIC2PROC is then called to load global array VP with the virtual processor numbers of the processors that rebroadcast IC1P's data. If there are less than 3 processors in the configuration, processor 1 is assumed to have produced valid data. Procedure VOTEDATA is called to vote the data set. When all the data has been voted, procedure RESTORE is called to maintain the temporary storage area.

3.2.2.4.5 GLOBAL PROCEDURE ICINIT.

Procedure ICINIT loads initial values for the EXPECTED, LOCK, OLATMO and OPITMO buffers. It also clears the temporary storage areas.

3.2.3 The Redundancy Management Tasks.

The redundancy management tasks are responsible for interpreting the error data accumulated during the vote process, locating faulty processors, and

establishing a new task schedule based on the number of remaining good processors. These functions are carried out by 3 tasks; the ERRTASK, the FAULTISOLATIONTASK and the RECFTASK.

3.2.3.1 Data Structures and Buffers of the Redundancy Management Tasks.

It is the responsibility of these redundancy management tasks to construct the data structures which accurately reflect the system's current configuration. The following sections describe the data structures and buffers used to carry out this function.

3.2.3.1.1 The Constant THRESHOLD.

The Local Executive's ERROR array is an input to the redundancy management tasks. A processor's error count is deemed significant if it exceeds THRESHOLD. THRESHOLD's current value is 3 and is defined in the error task.

3.2.3.1.2 Buffers ERRERR, GEXECRECONF, and GEXECMEMORY.

Buffers ERRERR, GEXECRECONF, and GEXECMEMORY all refer to sets of processors, i.e. bits 0 thru 7 represent processors 1 thru 8. A bit marked in these buffers indicates that the respective processor has failed. The redundancy management tasks first interpret a processor's local error data, the ERROR array, to construct the error report for that processor. The reports are then broadcast to all processors in buffer ERRERR where they are examined and compared to determine if any processor has in fact failed. Buffer GEXECRECONF is used to distribute the configuration report. To prevent good processors from being eliminated by transient errors, a processor must produce errors for two consecutive frames (a frame is about 100 ms). Buffer GEXECMEMORY is used to hold the previous frame's actual configuration result. This will be logically anded with the current iteration's configuration result to produce the configuration report, GEXECRECONF. Thus, a processor has to be failed for two consecutive frames before a reconfiguration can take place.

3.2.3.1.3 PRESENTCONFIG.

PRESENTCONFIG holds the configuration report that was used to derive the processor's current configuration state. PRESENTCONFIG at one time was used to determine if a reconfiguration was necessary, i.e. if PRESENTCONFIG was not equal to the configuration report. Since the Version R modification, the reconfiguration function is efficient enough to be performed every frame.

```
PRESENTCONFIG: BITMAP;
```

3.2.3.1.4 NUMWORKING and NW.

NUMWORKING and NW are two manifestations of the same entity, the number of good working processors in the present configuration. Apart from the fact that NUMWORKING is specified as a global variable and NW is not, there remains no need for both these variables.

```
NUMWORKING AT NUMLOC: PROCESSOR;  (* NUMLOC = 6800_16 *)
NW: PROCESSOR;
```

In the original version of the operating system, NUMWORKING was used by those procedures which counted from 1, NW by those which counted from 0.

3.2.3.1.5 WORKING.

The WORKING array is constructed from the configuration report during reconfiguration.

```
WORKING: ARRAY[PROCESSOR] OF BOOLEAN;
```

An element WORKING[P] is true if processor P is not marked as failed in the configuration report.

3.2.3.1.6 The Processor Mappings VTOR, RTOV, and VTODF.

One of the major changes made in the operating system in going from the baseline version to Version R was the use of virtual processor numbers to reduce reconfiguration overhead. A processor's physical number is defined by where it is located in the system rack. This also defines which datafile mailbox the processor will transmit to. A processor's virtual number is defined by the ordering of the working processors and indicates which task schedule the processor is using. The three arrays VTOR, RTOV, and VTODF were created to implement this modification.

```
VTOR:  ARRAY[PROCESSOR] OF PROCESSOR;
RTOV:  ARRAY[PROCESSOR] OF PROCESSOR;
VTODF: ARRAY[PROCESSOR] OF DFINDEX;
```

VTOR translates virtual processor numbers into physical (or real) processor numbers. RTOV performs the inverse transformation. VTODF translates a processor's virtual number into its corresponding datafile mailbox address.

3.2.3.1.7 Schedule Pointers TPI and VPI.

If the ERROR array is considered to be the Local Executive's input to the Global Executive, then schedule pointers TPI and VPI are the Global Executive's output. TPI is an index into the SCHEDS array and points to the beginning of the task schedule the processor should execute. Similarly, VPI points to the beginning of the vote schedule.

```
TPI,VPI: SCHINDEX;      (* SCHINDEX = 0..6FF16 *)
```

3.2.3.2 GLOBAL FUNCTION ERRTASK.

Task ERRTASK constructs an error report from the WORKING and ERRORS arrays. The error report takes the form of a bitmap of processors. A marked bit indicates that the corresponding processor is considered failed by the reporting processor. A processor has to accumulate more than THRESHOLD errors before the host processor will report the subject processor failed. The current value of THRESHOLD is 3. The completed error report is broadcast into buffer ERRERR.

3.2.3.3 GLOBAL FUNCTION FAULTISOLATIONTASK.

The FAULTISOLATIONTASK uses the error reports of all working processors to arrive at a new system configuration of processors. For a processor to be considered faulty, at least two different processors must report the subject processor failed. All processors, both working and not working, are tested. To preclude the possibility that a transient error might cause the removal of a good processor from the system, a processor must be faulty for two consecutive major frames.

The FAULTISOLATIONTASK begins by loading all the error reports into array ERRPT. The error report is a bitmap of processors where a marked bit indicates that the processor is failed. The FAULTISOLATIONTASK uses WORKING and ERRPT to construct the new configuration, RECONF. RECONF is a bitmap of processors where a marked bit also indicates a failed processor. A single bit is marked in variable BITEST corresponding to the processor being examined, i.e. BITEST = 1 when processor 1 is being tested, BITEST = 2 for processor 2, BITEST = 4 for processor 3, etc. For all processors then, all working processor error reports are tested, excluding the subject processor. If 2 or more error reports have BITEST marked, BITEST is set in RECONF. The configuration report as represented by the "boolean and" (BAND) of RECONF with GEXECMEMORY is broadcast into buffer GEXECRECONF. GEXECMEMORY is the preceding frame's actual configuration result, i.e. RECONF. A processor would have to be failed for 2 consecutive frames before it would be marked as failed in GEXECRECONF. The current frames configuration value, RECONF, is then broadcast into GEXECMEMORY for use next frame.

3.2.3.4 GLOBAL FUNCTION RECFTASK and FUNCTION XRECF(RECONF: BITMAP).

Task RECFTASK calls function XRECF with parameter GEXECRECONF. Function XRECF is also used by procedure INITIALIZE to establish the initial configuration.

Function XRECF computes the configuration-dependent data structures. This computation is done whether the configuration has changed or not. The time elapsed during RECFTASK is therefore constant from frame to frame. Using the reconfiguration word, RECONF, XRECF constructs the WORKING, VTOR, RTOV and VTODF arrays. The working processor count, NW, is also set. Variable PRESENTCONFIG is set to RECONF. PRESENTCONFIG has no real purpose other than for testing and data acquisition. The reconfiguration word is also stored in buffer ORECONF for output to the simulation. XRECF then scans the schedule tables for the section built for the current number of working processors, NW. Within each section is a schedule table for each virtual processor and the vote table. XRECF locates the task schedule assigned the host processor and sets the initial task schedule pointer, TPI, to the first entry in the schedule. If a task schedule cannot be found for the host processor, the host halts with a value of $F00B_{16}$ in register R1, indicating it has been configured out of the system. The remaining processors establish the initial vote schedule pointer, VPI, and global variable NUMWORKING.

References

1. Wensley,J.H., et al., "Design of a Fault-Tolerant Airborne Digital Computer. Volume I-Architecture," NASA CR-132252, 1973.

2. Ratner, R.S., et al., "Design of a Fault-Tolerant Airborne Digital Computer. Volume II - Computational Requirements and Technology," NASA CR-132253, 1973.

3. Wensley, et al., " Design Study of Software Implemented Fault-Tolerance (SIFT) Computer," NASA CR-3011, 1982.

4. Goldberg J., et al., "Development and Analysis of the Software Implemented Fault-Tolerance (SIFT) Computer," NASA CR-172146, June 1984.

5. Palumbo,D.L. and Butler.R.W., "Measurement of SIFT Operating System Overhead," NASA TM 86322, April 1985.

6. Green,D.F., Palumbo,D.L., and Baltrus,D.W., "Software Implemented Fault-Tolerant (SIFT) User's Guide," NASA TM 86289, Aug. 1984.

7. Butler, R.W., Palumbo, D.L., and Johnson, S.C., "Application of a Clock Synchronization Validation Methodology to the SIFT Computer System," IEEE 15th International Symposium on Fault-Tolerant Computing," June 1985.

8. Pease, M., Shostak, R., and Lamport, L., "Reaching Agreement in the Presence of Faults," Journal of the ACM, Vol. 27, No. 2,Apr. 1980,pp. 228-234.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-87574		

4. Title and Subtitle	5. Report Date
The SIFT Hardware/Software Systems - Volume I A Detailed Description	September 1985
	6. Performing Organization Code 505-34-13-32

7. Author(s)	8. Performing Organization Report No.
Daniel L. Palumbo	
	10. Work Unit No.
9. Performing Organization Name and Address NASA Langley Research Center Hampton, Virginia 23665	11. Contract or Grant No.
	13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This report contains a detailed description of the software implemented fault-tolerant computer's operating system and hardware subsystems.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Fault tolerance Operating system Broadcast bus	[redacted] until September 30, 1987 Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	35	