Webb T. Comfort

A Fault-Tolerant System Architecture for Navy Applications

This paper describes the architecture of a computer system, being designed and built for the U.S. Navy, that is expected to be the standard Navy shipboard computer for the next twenty years or so. It has a requirement for very high system reliability, which is addressed by a multiprocessor system configuration that can recover dynamically from hardware faults and support on-line repair of failed hardware elements. Successfully accomplishing this requires various types of redundant hardware elements and special system architecture features, as well as intelligent fault-recovery software. This also requires that the application programs be designed to participate fully in the recovery and reconfiguration process. This paper presents the overall system architecture and discusses a number of significant new features designed to support fault-tolerant operation, including a duplex control bus, a computer interconnection system, a technique for remote diagnostics, a single-button maintenance procedure, and special fault-handling software.

Introduction

In the late 1950s, the U.S. Navy began to introduce digital computers on board its ships to perform various real-time applications. The AN/UYK-7 was designated as the large shipboard standard computer. In the late 1970s, in response to increasing performance requirements and improving technologies, the Navy defined requirements for a new family of standard computers. The AN/UYK-43 (the system discussed in this paper) is the successor to the AN/UYK-7, and is aimed at achieving significant improvements in cost, performance, and functional capability while retaining upward program compatibility from the AN/UYK-7. Special emphasis has been placed on fault recovery and on-line maintenance, which are the topics of discussion here.

Figure 1 is a photograph of the IBM AN/UYK-43 computer. The enclosure has a cross section of approximately 20 × 22 in. (50 × 55 cm) and stands 6 ft (1.8 m) tall. This enclosure contains the following equipment:

- Two CPUs, each rated at approximately 2.2 MIPS (million instructions per second).
- Two programmable I/O Controllers (IOCs), each connected to 32 channels. The connectors for the channels are on the back of the cabinet.
- Up to ten memory modules, for a total capacity of 2.5M 32-bit words of memory.
- An integral display/control panel and an integral power/temperature panel for maintenance operations.
- A computer interconnection system to allow multiple enclosures to be connected into a single system.

The computer in the photograph is referred to as a "B" enclosure. An "A" enclosure is also being built which is smaller than the "B" enclosure and contains about half of its equipment.

• *Specification requirements*

As with most government contract work, the AN/UYK-43 is being built [1] in response to a Statement of Work issued by the Navy [2]. This document contains detailed specifications on exactly how the computer is to be built, its functional characteristics, its performance characteristics, etc. Table 1 summarizes the key requirements relative to fault tolerance and reliability. Although the terminology is defined in the notes of the table, a few clarifying comments are appropriate.

- The mean time between the occurrence of hardware fault conditions is MTTF, while MTBF is the mean time between system failures. For a non-fault-tolerant system, MTTF is the same as MTBF. In a fault-tolerant system MTBF is considerably larger than MTTF because it allows for the recovery from fault conditions and for

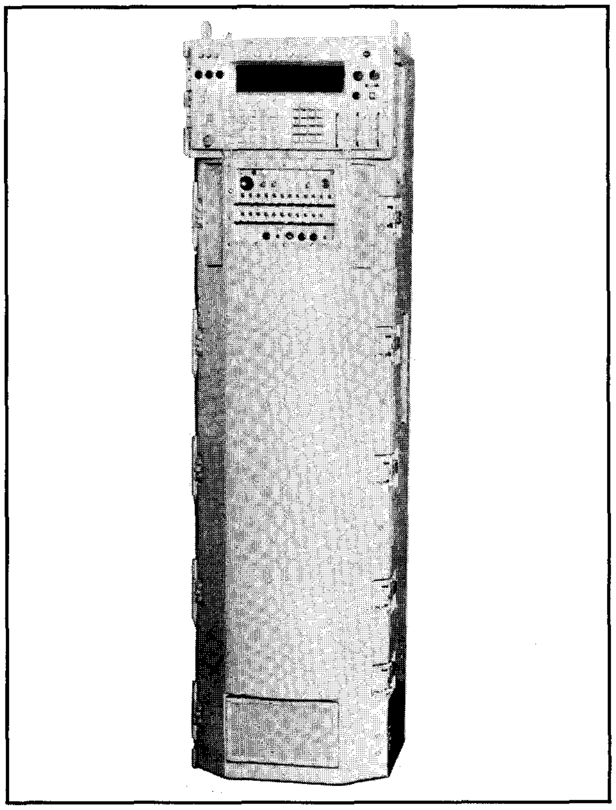


Figure 1 AN/UYK-43 "B" enclosure.

on-line repair (i.e., repair of the failed module while the system is operating). The Navy requirement is for a MTBF of at least 6000 hours; current projections show the MTBF for the "B" enclosure to be well in excess of 15 000 hours (based upon successful recovery from a fault condition, and with repair of the failed hardware module within eight hours).

- *Automatic isolation* is the process of running diagnostic programs in order to identify the specific replaceable unit (page) which, when replaced, will put the failed module back into operational status.
- The mean time to repair (MTTR) is based on the assumption that the faulty replaceable unit is successfully isolated and that the required replacement part is readily available.
- A *single-point computer fault* is a single fault condition which can cause the full "B" enclosure configuration to "crash," i.e., to bring the computer to a halted condition.
- The Navy "A" school graduate has attained at least a ninth-grade reading level, and has completed a 38-week Navy school as an Electronic Technician. In addition, he has completed a one-week maintenance course on the AN/UYK-43.

Table 1 Specification requirements for the AN/UYK-43.

MTTF $\geq$ 1050 hours (for "B" enclosure)
MTBF $\geq$ 6000 hours (with on-line repair)
Automatic isolation: 98% to three LRUs 95% to two LRUs 90% to one LRU
MTTR $\leq$ 15 minutes
MTTF of single-point computer faults $\geq$ 50$\times$ MTTF for total computer
99% of hardware faults repairable by Navy's "A" school graduate

MTTF = Mean time to fault condition
MTBF = Mean time between system failures
MTTR = Mean time to repair
LRU = Line replaceable unit

- *Similar work*

There are two existing systems which are very similar in philosophy and approach to the AN/UYK-43. One of these is the IBM 9020 computer developed for air-traffic control for the FAA [3]. This is a classical, tightly coupled multiprocessing system with three processors and three I/O controllers, all sharing a large modular main store. It has been in operation in various air-traffic control centers around the nation for about 15 years. The approach taken on the AN/UYK-43 for fault tolerance is similar in many respects to concepts developed for the 9020 system, particularly in the areas of on-line fault detection, fault recovery, dynamic fault analysis, dynamic system reconfiguration, and executive control.

The other system is a commercially available computing system known as the Tandem ®Non-Stop Computer [4–6]. The Tandem computer system has the same objectives as the AN/UYK-43, namely, being able to continue operation in the presence of failed hardware elements, and being able to perform on-line repair. The system architecture has a number of similarities. Up to sixteen processors can be connected together by means of a duplex bus, and it provides core memory with parity checking, and semiconductor memory with an Error-Correcting Code (ECC). Power switching is separate for each hardware module. However, there are also some differences. The Tandem design has explicitly avoided any shared memory, and has an I/O controller integrated into each CPU.

The IBM 3081 is one of IBM's newest high-performance large-scale computer systems. While it was not necessarily intended for high-availability applications, it has a number of characteristics similar to the AN/UYK-43 [7]. The 3081 contains two CPUs (called *dyadic* processors) with shared main store and shared I/O channels. Considerable emphasis was placed on fault detection and fault isolation, in order to achieve simplified maintenance for a complex machine. The 3081 is especially important from the point of view of fault tolerance because of its use of an embedded service processor to handle fault situations, and because of its implementation in LSI and the unique problems derived from that.

There are other fault-tolerant systems aimed at real-time control applications which have an entirely different approach from that described here. The objective for the AN/UYK-43 is to have a high availability for a relatively long mission time (on the order of 2000 hours). With these other systems, the objective is to have a high reliability for a relatively short mission time (a few hours); the emphasis there is on duplication and voting to mask out the effect of faults.

- The Space Shuttle avionics computer [8] contains four computers organized as a quad-redundant set. All four computers perform the same computations at the same time, and are synchronized by software, and their results are voted by software in order to determine when one computer has failed. In this way, the system is capable of retaining full computational capability after up to two faults. The system also has a fifth computer, with separate software, as a backup in case a third fault should occur, or in case a generic software design error should appear. This system is a premiere example of an operational multiple-redundant computer configuration.
- The Software Implemented Fault Tolerance (SIFT) system [9] is a multiple-processor system with private memories. Its development is sponsored by NASA, and the current experimental configuration contains ten processors. These processors can be grouped under software control into sets of three, where each set runs the identical computation and software votes on the results to determine when a failure occurs. It is more flexible than the Space Shuttle system because the allocation of processors to sets of three can be changed from time to time under software control. The emphasis is on the software structure and on designing the system to be able to prove the correctness of the fault-tolerant approach.
- The Fault-Tolerant Multiprocessor (FTMP) [10] is also sponsored by NASA. This system has multiple processors, but with shared main memory and redundant buses. The hardware modules can be grouped into triple sets (the same as with SIFT), with tight synchronization under program control.

Key features for fault tolerance

The ability to react to faults and to recover from their effects has been a primary concern in the AN/UYK-43 program and is reflected in the design of the hardware and software. Redundancy at the hardware module level is, of course, required. Subsequent sections of the paper discuss various features, techniques, and strategies in more detail. The following list highlights the most important and unique of these features.

- A duplex control bus is the principal mechanism for communicating between a CPU and the various I/O channels. It is duplexed in order to operate in the presence of faults. It is also the means by which remote diagnostics are executed, configuration control is managed, and communications are carried out with other enclosures.
- A basic principle adopted is that when a CPU experiences a fault which makes it incapable of executing instructions, then it should not be depended upon to perform any active functions in the recovery process which follows. Consequently, when a CPU suffers a catastrophic failure it is simply stopped and some other processor is interrupted to take charge of the subsequent recovery procedure. This was judged to be a more cost-effective approach than providing a special built-in service processor to perform those functions. (See [11] for a description of one such service processor.)
- The functional hardware modules in the system are individually configurable; i.e., the interfaces with the buses can be controlled by software and the modules have individual power switches. This allows recovery software to prevent a failing module from contaminating the rest of the system, and also allows for repair of a faulty module while the rest of the system continues to operate.
- The computer interconnection system allows multiple enclosures to be connected in a variety of ways, based on the individual system requirements. They can be tightly coupled (and therefore share main store), or they can be loosely coupled and simply communicate by means of messages.
- The Fault-Tolerant and System Reconfiguration Module (FTRM) is the primary software module which is delivered with the system and which controls all fault-recovery actions. Its design is integrated closely with the various fault-tolerant features within the AN/UYK-43, and it interfaces with an executive program and with the user application programs to achieve fault tolerance.
- Isolation of a faulty *line replaceable unit* (LRU) is performed through the execution of diagnostic programs. On the AN/UYK-43, these programs are written in a special diagnostic language which is executable on either the IOC or the CPU.

These features are all described in more detail subsequently.

Organization of the paper

This introduction has summarized some background material and some key features of the AN/UYK-43 system. Subsequent sections of the paper cover the following:

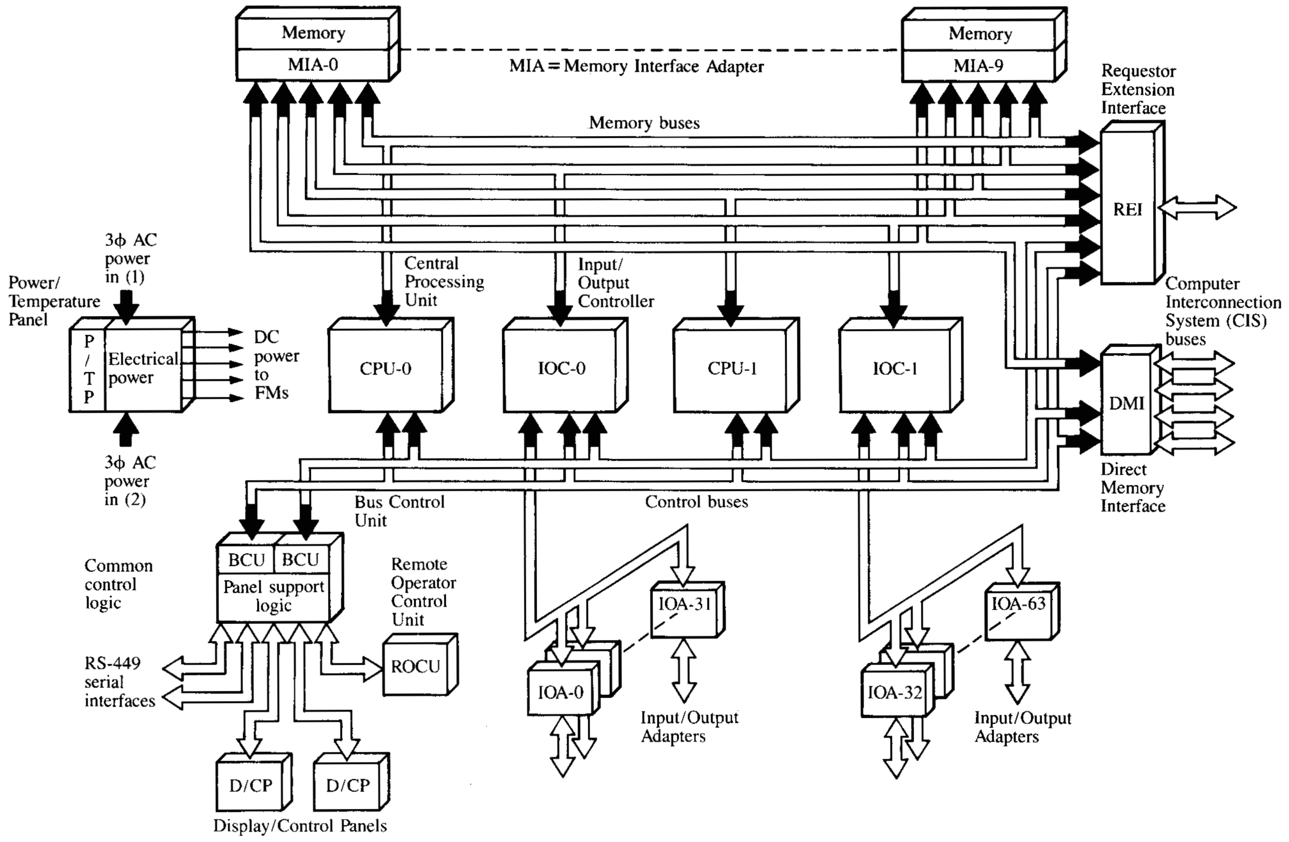


Figure 2 Computer set block diagram.

- A high-level view of the system architecture, and definition of some terminology used later in the paper.
- An overview of the casualty reaction process.
- A detailed discussion of the casualty reaction features of the AN/UYK-43, and how they relate to the system architecture and to the casualty reaction process.
- A summary and conclusions.

System architecture

A fundamental characteristic of a fault-tolerant system is that it must contain some form of redundancy, whether at the circuit level, the module level, the system level, or the software level. In order for a system to continue to operate in the presence of a fault, there must be redundant hardware somewhere either to correct the fault or to continue to operate correctly in place of the hardware affected by the fault. In the AN/UYK-43, this redundancy is introduced primarily (but not exclusively) at the level of the functional module, and therefore is evident in the system architecture.

Figure 2 is a system diagram of the AN/UYK-43 computer and reflects the hardware that is included within a "B" enclosure. The blocks in this diagram are generically referred to as functional modules (FMs), and the possible redundancies among the functional modules should be clear. Much of the terminology is relatively standard. It includes Central Processing Units (CPUs), Input/Output Controllers (IOCs), memories, Memory Interface Adapters (MIAs), and Bus Control Units (BCUs). An IOA is an Input/Output Adapter, the D/CP is a Display/Control Panel, the P/TP is a Power/Temperature Panel, and the ROCU is a Remote Operator Control Unit. The REI and DMI will be explained shortly. The top-level redundancy can be seen by inspection of the figure. There are two control buses, each of which can operate independently, and each of which serves as a backup for the other. There are separate memory buses for each processor (CPU and IOC), so that a bus failure will affect only one processor. Not shown in the figure is the fact that each IOC actually contains duplex processors. Provision for dual power sources is included.

A special Computer Interconnection System (CIS) has been designed to allow a number of enclosures to be connected together into larger systems. The CIS provides direct extensions of the memory bus and the control bus to additional enclosures. A special feature of this design is that the normal use of these bus extensions is transparent to the operational programmer; if a programmer wishes to access a main memory location in another enclosure, his program simply generates an address in the normal way. The address-decode mechanism then interprets the high-order bits of the address to decide whether the requested memory location is within this enclosure or another enclosure. Similarly, the principal function of the control bus is to communicate I/O requests from a CPU to an IOC controlling the proper channel. In order to request an I/O function from a channel attached to another enclosure, the CPU program simply requests the start of an I/O channel with an IOC number located in another enclosure.

The CIS is made up of a Requestor Extension Interface (REI) and a Direct Memory Interface (DMI). The REI in Enclosure 1 can be connected to DMIs in up to 16 other enclosures, and is the mechanism by which requests are sent from a processor within Enclosure 1 to any other enclosure. The DMI in Enclosure 1 is the mechanism by which other enclosures send requests to this enclosure. The DMI has four ports, each of which can be connected to one REI in another enclosure.

From a system point of view, the block diagram of Fig. 2 is that of a classical tightly coupled multiprocessing system where every processor is able to access all of main store. However, the actual use of memory is decided by the application software design, and can be constrained by means of software control. Each processor (as well as the CIS) contains an address translation table. This table converts logical addresses to physical addresses, and therefore the entries in the table control which processors have access to which memory modules. This allows a system operation which could include private storage for one or more of the processors, as well as defining how much memory (if any) would be shared by which processors.

An important feature of the system architecture is the control bus. I/O operations are invoked by means of a message sent from a CPU to the proper IOC via the control bus. In addition, the control bus is used to transmit interprocessor interrupts, diagnostic commands, and certain other error-handling controls. There are two independent and identical control buses, each controlled by its own BCU. In order to use a control bus, a processor that wishes to send a message sends a signal to the common control logic which selects one of the two buses which is not busy (either one). After priority resolution, if any, the requesting processor is granted one of the control buses. When it is granted a control bus, the requesting processor then proceeds to send one or more control and data words to the proper destination, receiving a reply if necessary. That bus is then free to be allocated to some other processor. If a fault is detected in the transmission of the desired message, the BCU is notified and the requesting processor is automatically switched to the other bus in order to retransmit the message.

An IOC is a programmable processor in its own right, with its own instruction set, register set, memory protection, memory bus, and translation table. It is microprogrammed and, in fact, contains two physical processors. An IOC can have up to 32 devices (IOAs) connected to it, and its primary function is to essentially multiplex the operations of those 32 channels. In addition, the IOC takes charge during initial program loading and can also assume command during error-recovery situations when there are no operating CPUs left in the enclosure.

As shown in Fig. 2, a "B" enclosure can contain up to ten memory modules, each consisting of a Memory Interface Adapter (MIA) and one or more memory array pages. A memory module may contain either semiconductor memory or core memory. The memory word in the semiconductor memory is expanded to include an error-correcting code (single error correction, double error detection), which provides for automatic correction of transient faults in the memory array. A core memory page may also contain a monolithic front end, as is described later.

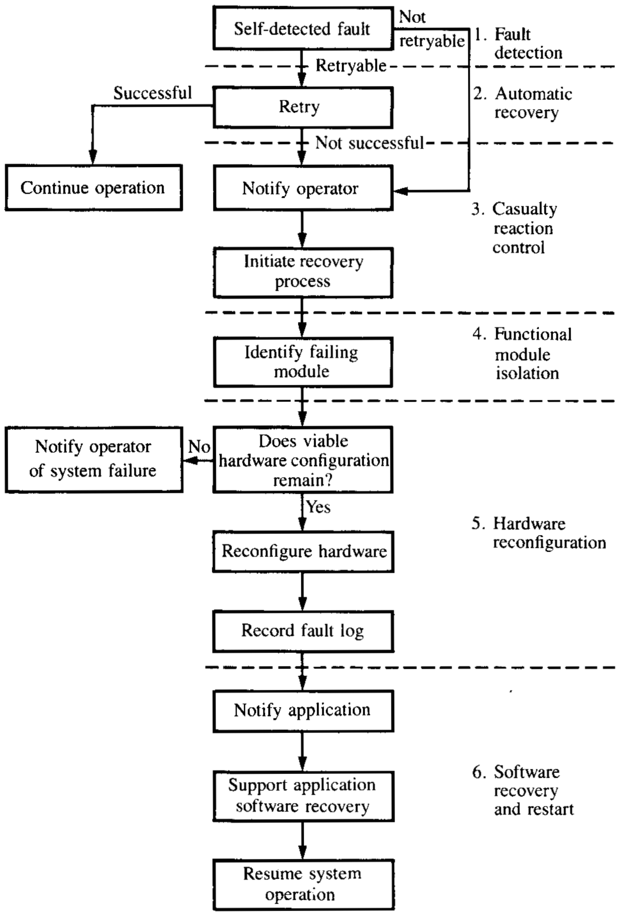
These are the principal characteristics of the system architecture for the AN/UYK-43, particularly as they relate to fault tolerance and casualty reaction. A more complete description can be found in [12].

Casualty reaction

Casualty reaction is a term used in the AN/UYK-43 program to cover all the processes, techniques, and features related to the handling of faults. The use of this term avoids some of the parochial definitions associated with such terms as *fault tolerance* or *high availability.* This section of the paper provides a top-level view of the major processes involved in casualty reaction.

A number of general requirements concerning functions to be provided for casualty reaction on the AN/UYK-43 system were provided in the Statement of Work. However, it required a significant system design effort to put in place a set of hardware and software capabilities to accomplish these general requirements. The principal objective of this paper is to describe how that has been accomplished. First, however, a few clarifying points are in order.

One of the most important requirements was the development of a casualty reaction design philosophy. This was accomplished very early in the program and represents an overall strategy of how to respond to fault conditions and how to accomplish on-line repair. This design philosophy was carefully documented and was the principal mechanism used to drive the hardware design and the software design to a coordinated solution. Most computer systems are developed without any particular motivation towards fault tolerance or



Self-detected fault
Not retryable
1. Fault detection
Retryable
Successful
Retry
2. Automatic recovery
Not successful
Continue operation
Notify operator
3. Casualty reaction control
Initiate recovery process
Identify failing module
4. Functional module isolation
Notify operator of system failure
No
Does viable hardware configuration remain?
Yes
Reconfigure hardware
5. Hardware reconfiguration
Record fault log
Notify application
Support application software recovery
6. Software recovery and restart
Resume system operation

Figure 3 Fault recovery phase.

casualty reaction. The result is that hardware capabilities such as fault detectors are left to the whims of the designer, and any software capabilities are added later as an afterthought. An effective fault-tolerant system simply cannot be designed this way. (A documented overview of the casualty reaction process for the AN/UYK-43 is contained in [13].)

Some effort was given to the consideration of software faults, i.e., residual bugs in the application programs. However, there does not appear to be at this time any effective, readily usable approach to this problem. There has been considerable research directed to this problem in the last few years in the areas of design diversity [14], recovery blocks [15], and formal proof of correctness of programs [16]. However, none of these approaches has reached the stage yet where it could readily be used in an arbitrary system application, and therefore does not provide applicable direction towards the design of a system like the AN/UYK-43.

Development of programming guidelines is an important part of the AN/UYK-43 program. Most existing fault-tolerant systems are designed to support a very specific application. This is an important advantage because the application software package can be designed in conjunction with the hardware system in order to provide a coherent approach to the handling of fault conditions. However, the requirement here is to develop the AN/UYK-43 system as a hardware/software package that can then be applied to various Navy shipboard applications. The objective of the programming guidelines is to provide advice to future application programmers on how they should design and organize their programs so as to effectively take advantage of, and work with, the facilities provided with the AN/UYK-43 computer.

The strategy for casualty reaction is divided into two principal phases. The first is *fault recovery*, which consists of the actions that take place when a fault is detected. The second is *fault repair*, which is the process involved with the on-line repair of a failed functional module. These two phases are treated separately because the processes involved are different, and because they may occur discontiguously in time. Fault recovery is an immediate reaction which gets the system operating again (perhaps in a degraded mode) while fault repair is a process that happens whenever the maintenance technician decides that it is appropriate.

• *Fault recovery phase*

The fault recovery phase begins the instant a fault is detected. The initial actions (when the fault is detected by hardware) are performed automatically by hardware. Then control is passed to the Fault-Tolerant and System Reconfiguration Module (FTRM) which performs another set of actions. Finally, notification is sent to the application program, which must then perform the final software recovery. The fault recovery phase completes when the system has resumed operation.

Figure 3 is a top-level flow chart of the steps that are executed to accomplish the fault recovery phase. These steps are discussed briefly here in order to give an overall picture of how casualty reaction works. Details of many of the features are described in a subsequent section.

1. *Fault detection*—This refers to a hardware-detected fault. The fault may have occurred in the hardware or it may be an error in the software, but in either case, a fault detector went off. A large number of real-time continuous hardware fault checkers are included in the AN/UYK-43 design. An interface is also provided for FTRM to respond to the software detection of faults.
2. *Automatic recovery*—The block in the flow chart uses the term "Retry," but this does not necessarily mean that an instruction or function is retried. Instead, it represents a set of facilities where in specific cases the

hardware has sufficient capability to automatically correct the fault, either by correcting, by retrying, or by disabling the failing function. However it is done, if the correction is successful the application program continues operation almost as if no fault had occurred.

3. *Casualty reaction control*—At this point it has been determined that a solid failure has occurred (or, more correctly, that a fault has occurred which is being treated as a solid failure). If the fault is catastrophic in the sense that a specific functional module is no longer able to perform its function, this fact is automatically displayed to the maintenance operator. Then an interrupt is sent to a processor in order to invoke the fault-handling software (FTRM). For a noncatastrophic fault, this interrupt goes to a predefined processor. For a catastrophic fault, the processor to be interrupted is defined by means of a register located in the BCU. The contents of this register are under software control, providing two advantages. One is that it allows the FTRM design strategy to determine which processor should be interrupted in the case of a particular type of catastrophic fault. The other is that it allows FTRM to change that decision after a fault has occurred. This is useful because it simplifies the handling of multiple faults, should they occur.
4. *Functional module isolation*—The hardware interrupt generated from the previous step invokes FTRM operation in a recovery processor. Note that FTRM can be executed on either CPU in the enclosure. FTRM does some preliminary handling of the interrupt, but its principal function is to identify the functional module which has failed. In the great majority of cases this is immediately obvious from the type of interrupt that occurred and from the interrupt status code. In the cases where it is not obvious, FTRM must execute an algorithm in an attempt to identify the specific functional module which has failed. These ambiguous cases are principally associated with the memory bus and the control bus.
5. *Hardware reconfiguration*—The reconfiguration of a functional module is the process of clamping its interfaces with the rest of the system so that, no matter what kind of erratic behavior it may exhibit, it cannot impact the normal operation of the other functional modules. The primary function of this step is to configure out the functional module which was identified as the one that failed. However, two other functions are also performed here. The first is to establish that there is an adequate set of hardware resources remaining operational to keep the system running. An obvious example is that if the last processor in the enclosure fails, there is no computational capability left in that enclosure and the only apparent option is to declare the system failed. In addition, a fault log is kept by the software, which records each fault that occurred and what reconfiguration action (if any) was taken.
6. *Software recovery and restart*—With the failed functional module now out of the system, the rest of the hardware should be able to operate in a normal way. However, it may be necessary to go back and "repair" the application program so that it can continue operation. A message is sent from FTRM to the application program describing exactly what went wrong and what functional module has failed. The application program must now take some kind of recovery action in order to restart the system. This may involve going back to a recent checkpoint, or switching in a hot spare hardware module, or perhaps loading into memory a substitute program which will operate in a degraded mode.

It cannot be emphasized too strongly that the ultimate success of this type of a recovery scheme depends upon the application software being able to continue operation even though one or more specific hardware modules have failed. There are a number of techniques available which can be used to aid in this process, but these must be designed into the application program from the start. Other forms of redundancy which would not impact the software operation, such as triple modular redundancy [17, 18], might be implemented in hardware. However, these are considerably more expensive in terms of hardware.

While the details of the various steps may be different, this fault-recovery process applies no matter which functional module has failed. The process also applies for multiple faults, as long as the second fault does not occur until the recovery process is complete for the first fault. Since the recovery process is expected to take only a few seconds at most, the probability of that occurrence is quite small. (It is important to distinguish here between the occurrence of a second independent fault and the occurrence of a second indication of the same fault.)

- *Fault repair phase*

When the fault recovery phase is complete, the maintenance panel (D/CP in Fig. 2) contains a display showing which functional module has failed. When the maintenance technician observes this and is prepared to institute a repair action, a button on the panel is depressed to initiate the fault repair phase. The fault repair process involves the identification of the specific pluggable element (LRU) which has failed, replacing it, checking that the module is now working properly, and returning the module to full operation within the system. Figure 4 is the high-level flow chart of this process, which consists of the following steps:

7. *LRU isolation*—The application program has veto power over the request by the maintenance technician to begin on-line repair (since the application may be

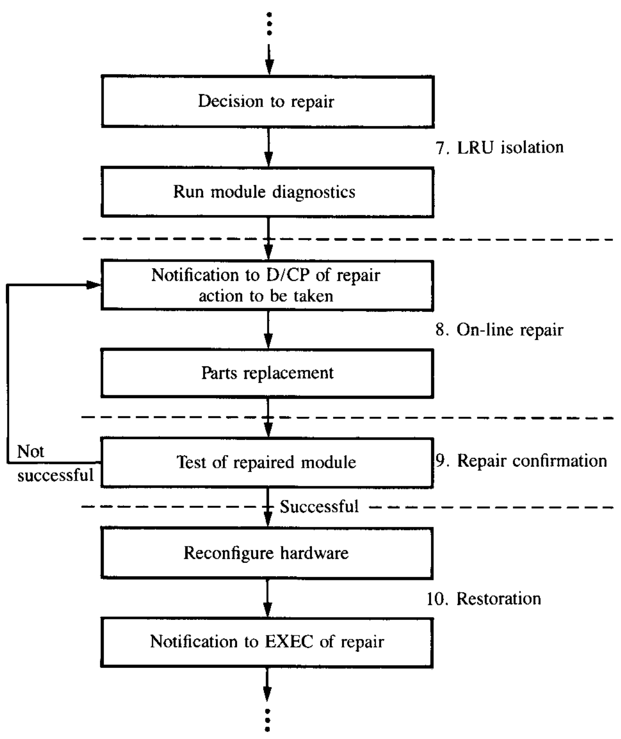


Figure 4 Fault repair phase.

Table 2 Casualty reaction responsibilities.

Process step	*Responsibility*	*Support*
Fault recovery		
1. Fault detection	Hardware, FTRM, application	—
2. Automatic recovery	Hardware	FTRM
3. Casualty reaction control	Hardware	FTRM
4. Functional module isolation	FTRM	—
5. Hardware reconfiguration	FTRM	Hardware
6. Software recovery and restart	Application	FTRM
Fault repair		
7. LRU isolation	FTRM	Application
8. On-line repair	Maintenance technician	FTRM, hardware
9. Repair confirmation	FTRM	Application
10. Restoration	Application	FTRM

involved in some critical process and may not wish to devote system resources to the repair action at this time). If the application agrees to the repair process, the principal function to be carried out is to run a set of diagnostics against the failed functional module to determine the specific LRU which needs to be replaced. The application program makes available a block of 32K words of memory, and the proper set of diagnostic programs is loaded into that memory from an external device. This set of programs is then scheduled as a task with the executive program so that it may run in a background mode with the existing application program. In this way, the application program can control the amount of CPU time spent running the diagnostics. When the diagnostic programs are complete, they have identified the most likely set of LRUs which should be replaced to repair the functional module.

8. *On-line repair*—A special display is presented on the D/CP of the LRUs to be replaced. (This is illustrated subsequently.) The rest of this step is a manual process. The repair technician must switch off the power to the failed module (the power for each functional module can be switched on and off separately), open the cabinet, remove the faulty LRU and replace it with a spare, and close the cabinet and turn the power to that module back on. A button is then depressed on the D/CP requesting that the functional module be tested again to see if the repair was effective.
9. *Repair confirmation*—The next step is to retest the functional module to see if it is now operating properly. This is accomplished by running the same set of diagnostic programs against that module. If no failures are found, the D/CP display will show that the previously failed functional module is now in standby mode, which indicates to the technician that the repair has been successful.
10. *Restoration*—Since the module has now been repaired, it must be reconfigured back into the active hardware system; that is, the interfaces which were effectively disconnected during the recovery phase can now be reconnected. This means that the functional module is now fully capable of operating with the rest of the modules in the system. A message is then sent to the executive (or application) program indicating that the identified functional module is now ready for use. It is up to the application program to decide when and how to begin using this hardware resource again.

Note that this entire fault repair phase is executed on line; that is, the diagnostics are run and the repairs made by the maintenance technician while the rest of the system continues to operate and perform its application function.

• *Casualty reaction responsibilities*

The ten steps just described represent the overall strategy for casualty reaction on the AN/UYK-43. As was indicated earlier, it represents a coordinated and integrated design of hardware, fault-tolerant software, and application software. In order to clarify this responsibility, Table 2 lists each of the ten steps, shows where the primary responsibility lies for executing each step, and also shows those cases where support is required from other areas.

The distinction between the fault-handling software (FTRM) and the application software is especially important. FTRM cannot and does not attempt to provide a complete fault-recovery function for an arbitrary application. That application software must be designed to be able to handle those cases where individual hardware modules have failed. The primary objective of having the fault-tolerant design philosophy just described is to make sure that the people responsible for each of the areas of the system know specifically what their responsibilities are and where their interfaces are with the rest of the system.

Fault-tolerant features

The previous two sections have given a general understanding of the hardware configuration and how casualty reaction works on the AN/UYK-43. This provides a context within which to discuss specific fault-tolerant capabilities that are provided in hardware and/or software to implement the system. The following discussion is divided into subsections, each of which covers a particular capability. In each case, the more important or more interesting features are described in some detail.

- *Fault detection*

The casualty reaction process does not start until a fault is detected, which can be accomplished by either hardware or software. In general, hardware fault detection is preferable to software fault detection because it occurs continuously and in real time. Here, "continuous" means that every instance of a function is checked, and "real time" means that the fault is detected within a very few machine cycles of when it actually occurs. Software fault detectors normally only apply when they are explicitly invoked by the macro program, and will usually not detect a fault until many (hundreds of) instruction execution times after the fault actually occurs. Continuous real-time detection is much more effective because there is a lower probability that the fault will be propagated to some other part of the system, because it is much easier to know precisely where the fault occurred, and because the detector will also find transient faults. But whatever fault-detection techniques are used, successfully achieving the potential high reliability of a fault-tolerant system is dependent upon detecting a very high percentage of the faults that occur.

A large number of hardware fault detectors have been included in the AN/UYK-43 design. Table 3 summarizes the number of individual fault detectors designed into each type of functional module, classified by the general types of detectors involved. In general, when a fault is detected it causes an interrupt into one of the CPUs. This results in a context switch to an interrupt-handling program, the masking of other interrupts of the same type, and the storing of an interrupt status code containing information defining the specific type of fault that occurred. Certain faults (primarily in the CPU) are considered catastrophic whenever the module is no longer able to execute any reasonable function. In such a case, the failed CPU is halted and an interrupt is sent to the other CPU. Faults detected in some other functional module will result in an interrupt into one of the CPUs so that the FTRM program can be invoked and casualty reaction can proceed.

Table 3 Number of hardware fault detectors, by type.

Type	*CPU*	*Memory*	*IOC*	*BCU/ PSL*	*CIS*	*Panels*
Parity check	31	5	11	2	17	3
Timeout	8	3	7	12	6	8
Decode check	6	1	5		3	1
Limit check	4		3		1	
Interface check	2		3	2		6
Others	10	1	8	3		9

Note: See Fig. 2 for definitions of functional modules.

Software fault detection occurs when some macro-level program designed to look for erroneous conditions finds one. Three kinds of software fault detection are supported in the AN/UYK-43.

- Part of the FTRM software package is a set of macro-level self-test programs. At system initialization time, FTRM requests the executive to schedule these programs as a background task and to execute them periodically. These programs are specifically designed to detect faults in areas of the hardware not covered by the hardware fault detectors. Whenever any of these self-test programs detect a fault, they send a message to FTRM indicating the fault condition.
- Two special instructions are provided in the AN/UYK-43 architecture: CPU confidence test and IOC confidence test. The execution of these instructions invoke microprogrammed procedures which perform a basic check on the processor function and its ability to execute instructions. Detection of a fault results in an interrupt to the CPU.
- Application programmers are encouraged to include fault detection in the design of their software modules. These might include techniques such as checking parameters to be within limits, using check sums on blocks of data, or using timeouts. FTRM provides an interface whereby, if such a fault is detected, a message can be sent to FTRM, whereupon it makes a check of its own on the affected functional modules.

Table 4 Automatic recovery techniques.

Recovery technique	*Generic method*	*System notification*	*Impact on system*
1. Cache disable	Correct and avoid	Interrupt	Loss of CPU performance
2. Disable monolithic front end		Count	Loss of memory bandwidth
3. Disable IOC engine		Interrupt	Loss of I/O bandwidth
4. Retry main memory	Correct transient faults	Count	Temporary delay
5. Retry IOA read		Status	Probably none
6. Alternate control bus	Repeatedly correct	Interrupt	Loss of bus bandwidth
7. ECC on semiconductor memory		Count	None
8. Disable memory priority	Avoid	Status	Loss of priority

• *Automatic recovery*

Automatic recovery, as discussed here, is characterized by three principal features:

- The detected fault is fixed immediately, that is, within just a few machine cycles.
- There is no lost or damaged information. The fault is corrected in real time, and the damage is not propagated into the system in any way.
- The fault condition is transparent to the software; that is, applications programs do not even know that the fault has occurred.

Within the limits of this definition, there are a number of automatic recovery techniques which have been included in the design of the AN/UYK-43. The principal ones are listed and classified in Table 4. Brief descriptions of each are given in the following:

1. Each CPU contains a cache memory in order to improve the average memory access time. A fault in the cache memory or its associated circuitry causes the cache to be disabled; thereafter, memory accesses go directly to main memory without using the cache.
2. A special memory has been developed for the AN/UYK-43 computer which is called a mono-fronted core memory. This memory module contains 64K words of core memory plus 64K words of monolithic memory, and there is a one-to-one mapping between the contents of the two memories. Each store into the memory module first stores into the monolithic memory and then stores the same data into the corresponding word in the core memory. Reads from the memory simply come from the monolithic memory. Such a memory module provides the access time of a monolithic memory while also providing the retention of data in the core memory array during power outages. The automatic recovery technique occurs when a fault is detected in the monolithic front end, whereupon the monolithic memory is disabled and all future accesses simply come from the core memory.
3. The IOC actually contains two identical processors. Whenever one of these fails, it is disabled and the other engine takes over the full load for the IOC. (However, one processor can only support one-half of the full I/O data rate.)
4. In both the CPU and the IOC, whenever a fault is detected on the memory bus (as distinguished from a solid fault detected in the memory module itself), the processor will retry the access one time in order to correct for a possible transient error on the memory bus. If the access on the retry also detects the fault, it is treated as a solid fault.
5. When an IOC is reading data from a device through an IOA, if a fault is detected, the IOC will try to reread the data. Note that the IOC cannot retry I/O data which are being written to the device because the detection of that fault must occur out in the device itself.
6. The duplex control bus has been described previously. Whenever a processor tries to use the control bus and detects a fault in the bus it is using, it will automatically switch to the alternate bus and attempt to get the message through there.
7. Extra bits are included in the semiconductor memory modules to provide correction of single-bit faults. Note that if a solid single-bit fault occurs either in a single word or in all words in the module, that fault will be corrected on every memory access.
8. The memory interface adapter for each memory module contains a special chip which implements the desired priority control for that memory module. The logic also includes a special high-speed bypass invoked whenever there is only a single request outstanding against that memory module. (If there is only one requestor at a time, there is no need to resolve priorities). This high-speed bypass circuit is then also used to provide automatic recovery. If a fault is detected in the memory priority circuit, the circuit is switched out of operation and the high-speed bypass is applied to every access. In this case, the normal priority algorithm is replaced by a simple first-in, first-out resolution of memory contention accesses.

Three other types of information are included in Table 4 (second through fourth columns). One is the generic method for handling the fault. In the first three cases, the fault which was detected is corrected and further occurrences are avoided because the faulty hardware is no longer being used. The next two cases are simply the correction of a transient fault. The next two cases will correct for transient faults but will also repeatedly correct the presence of a solid fault. The final case does not do any correction; it simply avoids the fault conditions by disabling the faulty hardware.

As previously discussed, one of the characteristics of this type of automatic recovery is that the application software does not know that a fault has occurred. However, it is still important that the fault not be ignored completely; that is, there should be a record of the fact that the fault has occurred. As indicated in Table 4, system notification can be accomplished in three different ways. In some cases an interrupt is generated anyway, even though the fault has been corrected. Such an interrupt is handled directly by FTRM in order to update the status of the functional module which is experiencing the fault. In some cases, the hardware provides a counter which is incremented every time the particular fault occurs. If the count should ever overflow, that will cause an interrupt. The third technique is to simply record, in an internal status word, the fact that a particular fault has occurred. FTRM periodically samples these counters and status words, and thus becomes aware when these faults have occurred.

While there is no direct impact on the application software, the existence of a fault condition will in general have some kind of an impact on the operation of the hardware. The fourth column of Table 4 indicates, for each recovery technique, what the nature of the impact is expected to be.

• *Hardware reconfiguration*

Hardware reconfiguration is the process of logically disconnecting a functional module from the rest of the system. (The term also applies when a repaired functional module is reconnected to the system.) The form of the disconnection should not be absolute disconnection; rather, the level of disconnection is determined by the following objectives:

- The most important requirement is to ensure that, if a module has failed, no matter what it might do it should not impact any of the rest of the active system. This implies that all normal outgoing operational interfaces must be disabled.
- If a functional module in the operational part of the system should attempt to make a normal operational access to the failed module, this should be detected and indicated to the operational system.
- A processor in the operational system should be able to get into the failed module in order to execute diagnostics on that module. This allows a recovery processor to perform fault isolation on the failed module and then to verify complete operational capability of the repaired module before configuring the module back into the system.
- The reconfiguration mechanisms should be under software control.
- The actual hardware controls should be located external to the failed module, i.e., should be part of a good module.
- Once the failed module has been configured out, it should not be able, by itself, to change the reconfiguration controls.

The basic strategy for casualty reaction is reflected in this set of objectives. Rather than expecting a failed module to diagnose itself, the strategy is for a good processor to cause the diagnostics to be run on the failed unit. This requirement legislates against a reconfiguration mechanism which completely disconnects the failed module from the system.

Reconfiguration control has to occur at the interfaces between an individual functional module and the rest of the system. As can be seen in Fig. 2, those primary interfaces are at the memory bus and the control bus. Separate configuration control mechanisms are provided for each of these.

There are two separate control mechanisms provided at the memory interface. The first can be used to prevent a failed processor from accessing a good memory, while the second is used to prevent a good processor from accessing a failed memory. The first mechanism involves the priority circuits in each memory module. Each Memory Interface Adapter (MIA) contains a relatively sophisticated programmable priority mechanism. Basically, the priority circuit contains a table with ten slots in it, where each processor can be allocated by software to one (or more) of the ten slots. The important point is that the memory module does not even recognize a request from a processor unless that processor's identifier has been inserted into the priority table somewhere. Therefore, for example, if CPU 0 is failing, FTRM can remove the identifier of that CPU from the priority table in each memory module; this prevents the failed processor from reading or writing into any memory module. (Note that this mechanism can be used selectively to allow the failed processor to access one memory module for diagnostic purposes, if that should be desired.) The second mechanism for controlling the memory interface is the address translation table. Each CPU and each IOC (as well as the CIS) contains its own address translation table, which maps logical memory addresses onto physical memory modules. If Memory Module 2 has failed, FTRM can ensure that no processor will try to access that module by modifying all the address

Table 5 Fault-Tolerant and System Reconfiguration Module (FTRM) characteristics.

Software module
Runs under existing executive program
Handles hardware fault interrupts
Isolates faulty hardware module
Makes system failure decision*
Reconfigures hardware, as required*
Maintains status of hardware configuration
Notifies application program of changes in hardware status
Maintains error log
Controls on-line repair*
Schedules periodic self-test programs*
Controls multiple-enclosure fault recovery

*With guidance from the application program.

Table 6 Principal FTRM components.

Program modules:
FTRM initialization
Interrupt handler
FM isolation
Application services
On-line repair
Data structures:
System resource table
Casualty error data table
Casualty reaction log

translation tables so that no logical address will map into the physical addresses associated with that module. (Close coordination between FTRM and the application program is required to do this in an intelligent way.)

Use of the control bus is regulated by the Bus Control Unit (BCU). Any functional module which wants to send a message over the control bus must first send a request line to the BCU, which then legislates which functional module will be allocated the bus for a period of time. The BCU contains a mask register with one bit associated with each functional module. If this mask bit is on, the associated module is not allowed to use the control bus. This mask register can be loaded by FTRM, which allows it to control which modules can use the bus and which ones cannot. Therefore, if CPU 0 has failed, FTRM (running in CPU 1) can set the mask register in the BCU to prevent CPU 0 from accessing the control bus. Note that CPU 0 can still receive messages over the control bus from some other processor; it is just not allowed to send its own messages over the control bus. Consequently, a failed processor cannot pollute the system by usurping the control bus; but, at the same time, a good processor can execute diagnostics by sending them over the control bus to the failed processor.

In addition to controlling the interfaces, certain operational controls are also available. Diagnostic commands can be sent over the control bus to a processor to control its actions, such as Stop, Start, Single Step, and Single Micro Step.

The configuration controls that have just been described apply primarily within a single enclosure. When several enclosures are connected together to form a complex system, there are features available within the CIS to control those interfaces also. These include the ability to enable or disable the CIS interface; an interface priority circuit (identical to the one in the memory module) which can be used to limit accesses; and an address translation table and a set of protection registers, both of which can be used to control memory accesses from outside the enclosure.

• *FTRM program*

Previous sections have described hardware features provided with the AN/UYK-43 for the handling of faults. The Fault-Tolerant and System Reconfiguration Module (FTRM) is the program which is responsible for overall control of casualty reaction in the system. It uses all of the hardware information available, implements the overall casualty reaction strategy, and interfaces with the executive and application programs. Table 5 is a list of the characteristics of FTRM and the functions which it must provide. Table 6 is a list of the principal components of FTRM, both program modules and data structures. Each of these is described briefly.

FTRM initialization

Part of the load package which is read into memory at initialization time is FTRM, and it is the first program to be executed. It has two main functions. One is to initialize the hardware configuration. There are a number of tables and registers which control the operation of the hardware configuration, including the address translation tables, the priority registers of the memory modules, and the various registers that control communications among the hardware modules. While the hardware provides default values in these areas, FTRM must replace them with values that fit with the operational system. The other principal function is to create and initialize a system resource table so that it reflects the current operational configuration of hardware.

Interrupt handler

Under normal circumstances, while the machine is executing, FTRM never executes at all. It is invoked only when a fault occurs, which is normally indicated by an interrupt. The AN/UYK-43 has three classes of interrupts, one for hardware faults, one for program faults, and a third for I/O. Hardware faults are sent directly to FTRM and its interrupt handler. (The other two classes of interrupts go to the executive, which may decide that the interrupt really represents a fault, in which case it is passed on to FTRM.) On receiving the interrupt, FTRM creates an entry in the casualty error data table. The interrupt status information and the relevant software context information are saved in this table. The interrupt handler normally operates with interrupts masked off. Its work is kept to a minimum in order to reduce the period of time during which interrupts cannot be accepted. When it is completed, the interrupts are unmasked and control is passed to the FM isolation module.

One of the complicating problems here is that certain fault conditions (such as memory faults) can be detected more than once, by separate processors. FTRM must accept these multiple fault indications and sort them out appropriately.

Functional Module (FM) isolation
This module first performs a careful analysis of the interrupt status information in order to verify that a hardware fault has occurred. The program must then determine which functional module has experienced the fault. This is done by analyzing the interrupt status information. In most cases this is a straightforward process, and it can readily be determined which FM is at fault. However, there are some cases where this is not true. For example, if the interrupt indicates that a fault was detected on the control bus, the location of the actual failed component could be in one of four places: it could be the sender's interface to the control bus, it could be the receiver's interface to the control bus, it could be in the BCU controller, or it could be in the control bus itself. In such a case, FTRM must exercise a special algorithm to determine where the fault really occurred. Once the failed FM has been identified, the system resource table is updated to reflect this, and a message is sent to the application notifying it of the change in hardware status.

Application services
FTRM provides a number of special services related to fault handling for the application programs. These are provided by means of a standard message interface, where the application program sends a message to FTRM indicating the service it requires. Typical services include the following:

1. FTRM provides a facility to save vital data. It keeps its own critical data in two copies stored in two different memory modules. This service allows the application to send its vital data to FTRM, where it is saved in the two different stores. The data are not directly addressable by the application but must be accessed via FTRM.
2. FTRM schedules the execution of a set of periodic hardware self-test programs. The application may add additional software tests to that set or may delete from it.
3. If the application decides to change the configuration by adding or deleting one or more functional modules, it sends a request to FTRM, which then performs the necessary reconfiguration of the interfaces.
4. FTRM maintains a number of data structures containing information describing the system and the events that have occurred. (Some of these are discussed subsequently.) The application can request access to these data from FTRM.

On-line repair
On-line repair is invoked when the maintenance technician presses a button on the D/CP, indicating that he is ready to begin the repair phase. FTRM responds to this request and proceeds to schedule the execution of the appropriate diagnostic programs as a background task with the executive. In this way, the rest of the system continues operating while the diagnostics are being carried out. Upon completion of the diagnostics, the most probable list of LRUs to be replaced is known. FTRM then controls the dialog with the D/CP to assist the maintenance technician in making the repair. When the repair is successfully completed, it sends a message back to the application program, indicating that the repaired functional module is now available for use.

System resource table
This table contains one entry for each functional module in the system. It is created at system initialization time and is maintained by FTRM as long as the system is in operation, indicating when modules have been removed (due to failure or any other reason) and when they are returned. At any time, the current status of each FM can be found in this table (whether it is operational, halted, powered off, operating in a degraded mode, etc.). The application can access any of this information by means of the application services.

Casualty error data table
This table is essentially a buffer area for the processing of current interrupts; it contains one entry for each error type for each CPU. This is required because a high-priority interrupt may interrupt the handling of another interrupt before that interrupt is complete. Since there are two CPUs sharing the same copy of FTRM, there must be entries separately for each CPU. When all the data are collected for a specific interrupt, that information is recorded in the casualty reaction log.

Casualty reaction log
This table contains one entry for each fault indication that occurs in the system. The entry is created for a particular interrupt with the set of status information that was put together in the casualty error data table. The log entry is then expanded to include indications of what happened as a result of the fault (what reconfiguration occurred, the LRU callout that was made, and if repair was successful). This log is maintained in a fixed area of main store; when this area becomes full, the oldest entries are dropped off as new ones are added. If the application program desires to retain this information, it should periodically collect the information from FTRM and record it on a tape or some external device.

Table 7 Application Recovery Module (ARM) responsibilities.

React to fault notifications from FTRM
Decide what software was damaged
Decide and perform software recovery:
Reload and restart
Switch to degraded mode
Switch in spare hardware modules
Return to last checkpoint
Keep track of recoverable/nonrecoverable I/O
Coordinate multiple-enclosure recovery
Handle restoring of repaired hardware modules

- *Executive program interface*

A number of the FTRM functions (such as interrupt handling and the control of hardware resources) are very closely related to the traditional functions associated with executive programs. One of the complexities of the AN/UYK-43 program is that FTRM must be designed to operate with a number of different and unknown executive programs. Consequently, it has been necessary to draw the line between functions to be performed by FTRM and functions to be performed by the executive. The general approach has been to exclude from FTRM whenever possible those functions that are normally handled by an executive and that do not directly relate to fault handling. Clear well-defined interfaces are provided between FTRM and the executive. These interfaces have been identified and the principal ones are described here.

- As previously noted, FTRM is the first program to be invoked during system initialization. When this is completed, FTRM sends a message to the executive and expects it to do its own initialization and whatever initialization is required for the application program.
- The executive is expected to handle two of the three classes of interrupts, namely, the program fault conditions and the I/O conditions. If the executive decides that any of these cases represent potential hardware fault conditions, it is expected to pass these back to FTRM for error processing.
- Some of the functions of FTRM are to be performed as tasks in a multi-programming mode with other application programs. The two prime examples of this are the diagnostic programs and the periodic software self-test programs. When FTRM decides that these need to be executed, a request is sent to the executive to have these tasks scheduled in a normal way.
- The normal mechanism for passing status information or requests between FTRM and application tasks is by means of some form of message protocol. Most executives provide some general form of task-to-task communication which can be used for this purpose.
- FTRM will notify the executive whenever there is a change in hardware status. For example, if a fault interrupt has occurred and FTRM determines that Memory Module 7 has failed, FTRM will send a message to the executive indicating this. The executive is then free to take whatever action it wishes based on this information.

- *Application program responsibilities*

The ultimate effectiveness of fault recovery depends upon the application program being designed to respond to, and work around, failed hardware modules. After FTRM has done everything that it can to handle a fault, it finally sends a message to the application saying, in effect, "CPU-0 has failed. What do you want to do about it?"

The FTRM design assumes the existence of a generic application program known as the Application Recovery Module (ARM). This is the module which is responsible for the software recovery of the application and its interface with FTRM. This ARM might be a single module that controls recovery for the whole system; there might be a set of ARMs, each associated with a particular part of the system, or the ARM might be simply the executive. However it is implemented, this ARM is expected to perform certain functions, as outlined in Table 7. The ARM will receive a message from FTRM indicating which hardware module has failed, and based on its knowledge of the organization and structure of the total application, it must determine where and how the software has been damaged and what kind of recovery to implement. Some possible options are listed in the table. The question of how these various techniques should work is beyond the scope of this paper. (Unfortunately, there is little in the technical literature that discusses these various techniques.) The important point to recognize is that none of these techniques will work unless the application system is designed ahead of time to make them work.

In addition to providing one or more ARM programs, FTRM expects guidance from the application in the form of two tables. The first is called the Software State Mapping Table. It contains one entry for each hardware functional module, and its primary function is to provide a program module identifier, that is, the identifier of the proper ARM which should be notified when that functional module fails. Then if CPU-0 fails, for example, FTRM looks in the table to determine which ARM to send the message to, indicating that the failure has occurred.

The other table is called the Application Options Table. This table is used for the application program to provide guidance to FTRM on how to handle certain situations. It includes the following kinds of information:

- Certain of the FTRM application services are considered to be of a privileged nature. The Application Options

Table identifies which ARMs are allowed to request these privileged services. For example, one of the FTRM services allows the application program to request that an individual functional module be configured out of the system. The application program might wish to delete Memory Module 7 from the system (for some reason) and to request FTRM to do that. However, that is a very powerful capability, and it is probably not a good idea to let any arbitrary application program come in and make that kind of request. Hence, only privileged ARMs are allowed to make that specific request.

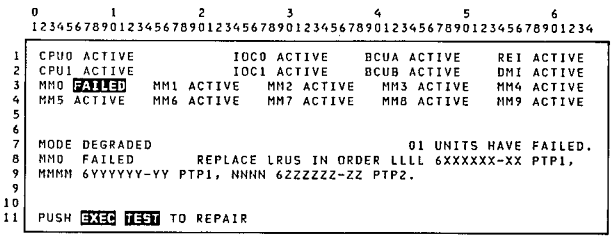
- In certain kinds of catastrophic fault conditions, the hardware or FTRM will automatically configure out a failed hardware module. However, the application program may be executing some very critical tasks and may feel that it is better to continue operation attempting to use that failed module. The Application Options Table provides a way for the application to instruct FTRM to do this.
- In order to execute the on-line diagnostic programs, FTRM must request the use of certain other system resources. The Application Options Table provides the identifier of the program module which is in charge of those resources.

These tables are expected to be designed into the overall application software package and included in the application load block. At initialization time, they would be sent to FTRM for its use during system operation. If the application elects not to provide these two tables, FTRM will use some simple default cases to handle each situation.

The application programmer is also encouraged to include software fault detectors where this is feasible.

- *Single-button maintenance*

Considerable emphasis has been placed in the AN/UYK-43 program on simplifying the maintenance procedures. This is important to the Navy for three reasons. First, maintenance is expected to be performed by technicians with very limited technical skills. Such technicians are simply not qualified to single-step through programs at a console or to use a scope to probe signal levels on logic pages. Second, many of these computers will be installed on ships and will be at sea for extended periods of time. Educated engineering personnel will not be available to perform maintenance on the machines. Third, with an MTTF of over 1000 hours, on the average a fault will occur only once every six weeks or so. In that situation, it is difficult for any technician to retain experience and expertise on how to make repairs effectively. As a result, considerable effort is being spent to provide complete diagnostic programs to perform the LRU isolation process and to provide a simple maintenance protocol at the D/CP. This protocol has been called *single-button maintenance*.



```
   0         1         2         3         4         5         6
   1234567890123456789012345678901234567890123456789012345678901234
 1  CPU0 ACTIVE            IOC0 ACTIVE    BCUA ACTIVE     REI ACTIVE
 2  CPU1 ACTIVE            IOC1 ACTIVE    BCUB ACTIVE     DMI ACTIVE
 3  MM0 FAILED   MM1 ACTIVE   MM2 ACTIVE   MM3 ACTIVE     MM4 ACTIVE
 4  MM5 ACTIVE   MM6 ACTIVE   MM7 ACTIVE   MM8 ACTIVE     MM9 ACTIVE
 5
 6
 7  MODE DEGRADED                               01 UNITS HAVE FAILED.
 8  MM0  FAILED       REPLACE LRUS IN ORDER LLLL 6XXXXXX-XX PTP1,
 9  MMMM 6YYYYYY-YY PTP1, NNNN 6ZZZZZZ-ZZ PTP2.
 9
10
11  PUSH EXEC TEST TO REPAIR
```

Figure 5 D/CP display.

This procedure follows the overall flow chart of Fig. 4, and is carried out as follows:

- The display on the D/CP indicates which functional module has failed (or more than one, if that is the case). The D/CP contains a display element with 11 lines and 64 characters per line, as illustrated in Fig. 5. The top four lines are continuously updated to show the operating state of each of the hardware functional modules in the enclosure. In the figure, it is indicated that Memory Module 0 has failed. Line 11 is the direction to the technician as to how to institute the repair action.
- The technician depresses the EXECUTE TEST button to indicate to FTRM that he is ready to institute a repair action. FTRM checks with the application for approval to do the repair operation at this time. If this is acceptable, FTRM proceeds to load the memory diagnostics into main store and to cause them to be executed. The diagnostic program runs multiprogrammed with the application, and when it is complete, has the proper LRU callout.
- FTRM displays the LRU callout on lines 8 and 9 of the display. Three LRUs are displayed, with the most probable one being displayed first. This display is illustrated in the figure.
- The technician must now make the physical repair. He powers down the failed memory module, opens the cabinet, and replaces one or more of the LRUs indicated on the display. He then closes the cabinet and turns the power to the failed module back on.
- As indicated on line 11 of the display, he pushes the EXECUTE TEST button again to cause the diagnostics to be rerun, in order to see if the repair has been effective.
- FTRM re-executes the memory diagnostics. If the diagnostics still detect a fault, the display will continue to show an LRU callout list (either the same one, or perhaps a different list). If the diagnostics no longer detect a fault, the display is changed. The technician now has the option either to rerun the diagnostic test or to push the BYPASS TEST button to notify FTRM to put the now-repaired memory module back into the system.

Table 8 Sample diagnostic commands.

CPU Macro Stop
Read CPU Scan Register
Load CPU Control Store
Start CPU Execution at Address K
CPU Continue
Read MIA Status Register
Read CIS Nested Status Register
Wrap Data
Wrap Bad Parity
Read DMI Priority Status
Start at Micro-Address K
Stop at IOC ENDOP
Read IOC Status
Microstep Sequence A
Microstep Sequence B
Read IOC Local Store (Address K)
Write IOC Local Store (Address K)
Reset Alternate BCU
Reset Error Status History
Change Bus Priority

- When he pushes BYPASS TEST, FTRM updates the System Resource Table to indicate that Memory Module 0 is now in standby mode (and changes the display accordingly). A message is sent to the proper ARM, which can then put the module back into active status.

This repair procedure can be aborted at any point by depressing the BYPASS TEST button. The technician might want to do this if, for example, he does not have the right spare parts in stock, or if the replacement of the called LRUs does not repair the problem.

As indicated previously, this maintenance procedure requires no knowledge of programs or registers or words in memory, etc. Still, it is expected to be effective in repairing the system about 99% of the time. For the remainder of the cases where the LRU callout turns out to be incorrect, or where the fault is in an area of the system not subject to diagnostics, a more complicated procedure is available and is supported by software in the machine. This includes traditional console functions, such as manual loading of registers, display of memory locations and register contents, single-stepping of the program, etc.

- *Remote diagnostics*

The basic diagnostic philosophy with the AN/UYK-43 is that it is better not to let a failing hardware module attempt to diagnose itself; rather, the failed module is put into a stopped state, and a good processor will come in over the control bus and execute the necessary diagnostic sequences to determine what is wrong with the failed module. This is referred to as "remote diagnostics" [19].

The package of diagnostics for a functional module is made up of a series of individual tests, where each test is a sequence of individual diagnostic commands to be sent over the control bus. Thus, for example, a specific test might include a sequence of commands which would load several CPU registers, load the macro-instruction counter, load the micro-instruction counter, single-step the microprogram 13 times, read out the contents of two internal registers, and compare the actual results with the predicted results. Such a test is executed by sending each of these commands sequentially over the control bus to the failed CPU. A list of some of the diagnostic commands that can be used in creating these test sequences is given in Table 8.

This use of the packages of diagnostic programs for the repair phase of casualty reaction is known as On-Line System Diagnostics (OLSD). The same set of diagnostic packages is also available for use in an off-line mode when the system is not operational. This is known as Stand-Alone System Diagnostics (SASD). For this situation, the identical set of diagnostic packages is used, but a diagnostic supervisor is added. The function of the diagnostic supervisor is to sequentially run diagnostics against every functional module in the system to determine which ones are working and which ones are not. Whenever a fault is identified in one of the functional modules, the same type of single-button maintenance scenario is provided at the D/CP.

SASD is invoked automatically under two circumstances. First, SASD is an integral part of the initialization process. When power is turned on, one of the first things that happens is that SASD is loaded, and it proceeds to check out the whole set of available hardware. (One of the byproducts of this first execution of SASD is the initial state of the system resource table, which is then passed to FTRM.) The second case where SASD is used is when the enclosure crashes. On-line repair is no longer possible, and the SASD initialization test is invoked.

The bulk of the code (the diagnostic tests themselves) is identical between OLSD and SASD. The only difference is how the tests are invoked and how they are sequenced. The diagnostics under OLSD are always executed on the CPU, whereas the diagnostics under SASD must sometimes be executed by an IOC. Since the IOC and the CPU are not architecturally compatible, the diagnostic programs have been written in a special diagnostic language which maps directly into control commands on the control bus. A separate interpreter is then provided, one for the CPU and one for the IOC, to cause the execution of the diagnostic commands over the control bus.

Conclusion

This paper has described how casualty reaction is being accomplished on the AN/UYK-43 computer. The overall philosophy of handling faults is not particularly novel, as similar things have been done on multiple-computer systems in the past. However, many of the details of the hardware and software implementation are new and interesting, and are expected to be quite effective.

• *Summary and evaluation*

Many concepts have been discussed in this paper, some at a detailed level and some at a more conceptual level. It would be inappropriate to try to summarize all of them. However, the following key concepts deserve comment and emphasis.

- Achieving a high system MTBF depends upon successfully recovering from faults, which in turn depends upon detecting the faults in the first place. Significant design effort and hardware circuit cost have been expended in the AN/UYK-43 to provide hardware fault detection. Ideally, one would like to be able to detect 100% of the faults continuously in real time by hardware. However, in each case, the decision of whether or not to include a particular hardware fault detector is a tradeoff of a number of factors, including the cost of the hardware checker as compared with the cost of the circuit being checked, the unreliability added to the system as a result of the addition of the checking circuit, and the possible performance degradation that might result from including the checker circuit.
- The single-button maintenance concept is very important, especially in the Navy context. However, its effectiveness depends directly upon the diagnostic programs being able to indicate the proper LRU callout.
- The overall casualty reaction strategy is a fairly general process. It should be applicable to a large class of multiple-CPU configurations. However, the design for a particular computing system must strike a balance between what is "right" for fault tolerance and what (sometimes skeptical) users are willing to put up with. Some users still object to providing programs with absolute control over what is done with faulty hardware. Detailed design is also a tradeoff between general solutions and detailed solutions. "Generality breeds inefficiency," and also leads to more complexity. Successful fault tolerance requires immediate response to fault conditions, and users often object to the allocation of significant amounts of system resources to handle these fault problems.
- This casualty reaction system is designed to recover from any single fault in a "B" enclosure (with two CPUs and two IOCs). However, it should also recover from many fault conditions in an A enclosure (with only one CPU and one IOC). Fault detection, software recovery, and single-button maintenance should apply in either case.
- The use of remote diagnostics (as previously described) appears to be a matter of philosophy. Many commercial computing systems (which are not necessarily emphasizing fault tolerance) allow a failed CPU to diagnose itself. In the AN/UYK-43 approach described here, it is felt that self-diagnosis could result in many more opportunities for polluting the system. Consequently, diagnostic control is given to a processor which is known to be operating properly.
- Again, the application program must be responsible for the ultimate software recovery, and generally this cannot be patched onto the application programs after the fact; it must be designed in from the beginning. This will be a nontrivial problem in trying to transport AN/UYK-7 programs to the AN/UYK-43 while at the same time trying to take advantage of the AN/UYK-43 casualty reaction capability.

• *Program status*

The Navy has awarded two parallel engineering development contracts for the AN/UYK-43 computer, one to IBM and one to another computer manufacturer. The instruction set architecture of the AN/UYK-43 computer was specified in detail in the Statement of Work, and therefore AN/UYK-43 programs should execute on machines from either manufacturer. However, the approach to fault tolerance and casualty reaction was only specified in general terms. Consequently, the approaches of the two companies are probably quite different. This paper has described IBM's approach to casualty reaction.

The engineering models of the AN/UYK-43 computer are scheduled to be delivered in March of 1983. Acceptance testing of the hardware includes the demonstration of the casualty reaction capabilities as described here. The Navy will then select one contractor for production. The AN/UYK-43 is expected to be operational in the fleet by 1986, and will provide a standard shipboard computer for the Navy for at least 20 years.

Acknowledgments

There are over 100 people at IBM currently working on the development of the AN/UYK-43 computer, so it is simply not possible to recognize all the people who have contributed to the design and implementation of this fault-tolerance strategy. However, three people should particularly be recognized. R. W. Bergeman was the manager of System Engineering on the AN/UYK-43 program at the time this fault-tolerant design philosophy was developed and put in place. While he provided guidance in a number of areas, he gave the author considerable latitude in laying out the casualty reaction process in the way the author saw best. R. E. Miles is the principal hardware systems engineer on the program and is responsible for the definition of a number of the key hardware features that are used here. Finally, M. C. Intrieri, who took charge of the design and development of the FTRM software, is a key individual involved in making the fault-tolerant system work.

References and notes

1. The work reported herein was performed, in part, under U.S. Navy Contract N00024-80-C-7135.
2. *RFP-N00024-80-R-7135, Attachment A, Statement of Work for Full-Scale Engineering Development of Computer Set AN/UYK-43 (XN-1)(V),* Department of the Navy, Naval Sea Systems Command (March 1980).
3. "An Application-Oriented Multiprocessing System," *IBM Syst. J.* **6,** No. 2 (1967). (This issue contains a series of six papers describing the 9020 system.)
4. J. A. Katzman, "A Fault-Tolerant Computing System," *11th Hawaii Conf. System Sciences* **3,** 85–102 (Jan. 1978).
5. ®Non-Stop is a registered trademark of Tandem Computers, Inc., Cupertino, CA.
6. J. F. Bartlett, "A 'Non-Stop' Operating System," *11th Hawaii Conf. System Sciences* **3,** 103–117 (Jan. 1978).
7. "System Development and Technology Aspects of the IBM 3081 Processor Complex," *IBM J. Res. Develop.* **26,** No. 1 (Jan. 1982). (This issue contains a set of papers describing the IBM 3081 and various aspects of its design and development.)
8. J. R. Sklaroff, "Redundancy Management Technique for Space Shuttle Computers," *IBM J. Res. Develop.* **20,** 20–28 (Jan. 1976).
9. J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," *Proc. IEEE* **66,** 1240–1255 (Oct. 1978).
10. A. L. Hopkins, Jr., T. B. Smith III, and J. H. Lala, "FTMP—A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," *Proc. IEEE* **66,** 1221–1239 (Oct. 1978).
11. J. Reilly, A. Sutton, R. Nasser, and R. Griscom, "Processor Controller for the IBM 3081," *IBM J. Res. Develop.* **26,** 22–29 (Jan. 1982).
12. "AN/UYK-43 (XN-1)(V) Computer Set, Technical Description," *Report No. 82-D91-004,* IBM Corporation, Owego, NY, May 1982.
13. W. T. Comfort, "Casualty Reaction on the AN/UYK-43: Scope and Overview," *Report No. 81-D91-002,* IBM Corporation, Owego, NY, Apr. 1981.
14. L. Chen and A. Avizienis, "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," *Digest FTCS-8, 8th Annual International Conference on Fault-Tolerant Computing,* June 1978, pp. 3–9.
15. B. Randell, "System Structure for Software Fault Tolerance," *IEEE Trans. Software Engineering* **SE-1,** 220–232 (June 1975).
16. R. C. Linger, H. Mills, and B. Witt, *Structured Programming Theory and Practice,* Addison-Wesley Publishing Co., Reading, MA, 1979.
17. R. E. Kuehn, "Computer Redundancy: Design, Performance, and Future," *IEEE Trans. Reliability* **R-18,** 3–11 (Feb. 1969).
18. A. Avizienis, "Fault-Tolerance: The Survival Attribute of Digital Systems," *Proc. IEEE* **66,** 1109–1125 (Oct. 1978).
19. In some commercial systems, the term "remote diagnostics" refers to diagnostics performed via a long-distance communications line.

Received September 29, 1982; revised December 12, 1982

Webb T. Comfort *IBM Federal Systems Division, Owego, New York 13827.* Mr. Comfort received a B.A. in mathematics from Pennsylvania State University and an M.S. in computer science from the University of Michigan. He joined IBM in 1956 and his early work was in programming and programming systems for real-time computers. In 1963, he transferred to Poughkeepsie, New York, where he worked on computer architecture for high-performance systems and multiple-computer systems. He was also responsible for the design of the operating system for the IBM System/360 Model 67, which was IBM's first commercial computer with virtual memory and dynamic paging. In 1973, he transferred back to the Federal Systems Division in Owego, where he has been doing computer architecture definition and evaluation for real-time control computers. Mr. Comfort is a senior programmer and for the last two years has been the lead systems engineer specifying and designing hardware and software fault-tolerance capabilities for the AN/UYK-43 computer. He has been an Association for Computing Machinery National Lecturer and spent one year as Visiting Professor in Computer Science at the University of Massachusetts. Mr. Comfort is a member of the Association for Computing Machinery, Phi Theta Kappa, and Sigma Xi.