simplex processor to 10^{-10} per hour for a multiprocessor that uses parallel hybrid redundancy. For those functions requiring fault masking a triplex level of redundancy is provided. For low criticality functions or noncritical functions, the GPCs may be duplex or simplex. Parallel hybrid redundancy is used for extremely high levels of fault tolerance and/or for longevity (long mission durations). GPCs can also be made damage tolerant by physically dispersing redundant GPC elements and providing secure and damage tolerant communications between these elements. Within AIPS, computers of varying levels of fault tolerance can coexist such that less reliable computers are not a detriment to higher reliability computers.

The overall framework in which AIPS operates can be characterized as a limited form of a fully distributed multicomputer system. A fully distributed fault and damage tolerant system must satisfy several requirements. The following subsections describe these requirements and characterize the AIPS architecture in the context of these requirements.

2.2 FUNCTION MIGRATION

A fully distributed system must have a multiplicity of resources which are freely assignable to functions on a short-term basis [3]. AIPS has multiple processing sites; however, they are not freely assigned to functions on a short-term basis. During routine operations the General Purpose Computers at various processing sites are assigned to perform a fixed set of functions, each computer doing a unique set of tasks. However, in response to some internal or external stimulus, the computers can be reassigned to a different set of functions. This results in some functions migrating from one processing site to another site in the system. Under certain conditions, it may also result in some functions being suspended entirely for a brief time period or for the remainder of the mission. In AIPS this form of limited distributed processing is called semi-dynamic function migration.

The internal stimuli that result in function migration may consist of detection of a fault in the system, a change in the system load due to a change in mission phase, etc. An example of an external stimulus is a crew initiated reconfiguration of the system.

2.3 RESOURCE TRANSPARENCY

Another characteristic of a fully distributed system is that the multiplicity of resources should be transparent to the user. To a large extent, this is true in the AIPS. Function migration is transparent to the function and the person implementing that function in software. Interfunction communication is handled by the operating system such that the location of the two communicating functions is also transparent to both. The two functions could be collocated in a GPC or they may be executing in different GPCs. Indeed, at one time they may be collocated, while at a later time one of them may have been migrated to another site. This transparency is achieved through a layered approach to interfunction communication. One of these layers determines the current processing site of the function to which one wishes to communicate. If it is another GPC, another layer in the communication hierarchy is invoked that takes care of appropriate IC bus message formatting and interface to the bus transmitters and

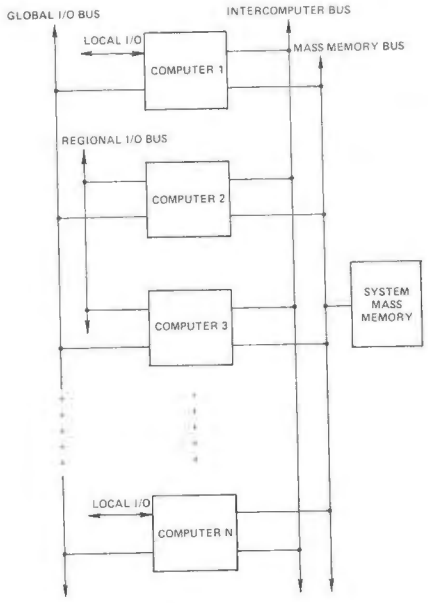


Figure 1. AIPS Architecture: A Software View

receivers, that is, the physical layer. This layered approach is responsible for hiding the existence of multiple computers from the applications programmer.

2.4 SYSTEM CONTROL

Another characteristic of a totally distributed system is that the system control is through multiple cooperating autonomous operating systems. The AIPS operational philosophy differs considerably in this regard. The overall AIPS system management and control authority is vested in one GPC at any given time. This GPC is called the Global Computer. All other GPCs are subservient to this GPC as far as system level functions are concerned. However, all the local functions are handled quite independently by each computer. This philosophy is more akin to a hybrid of hierarchical and federated systems. This is explained in the following.

Under normal circumstances each GPC operates fairly autonomously of other computers. Each GPC has a Local Operating System that performs all the functions necessary to keep that processing site operating in the desired fashion. The local operating system is responsible for an orderly start and initialization of the GPC, scheduling and dispatching of tasks, input/output services, task synchronization and communication services, and resource management. It also is responsible for maintaining the overall integrity of the processing site in the presence of faults. This involves fault detection, isolation, and reconfiguration (FDIR). The local operating system performs all of the redundancy management functions including FDIR, background self tests, transient and hard fault analysis, and fault logging.

The services provided by local operating systems at various processing sites are similar although they may differ in implementation. For example, the multiprocessor version of the operating system must take into account the multiplicity of processors

for task scheduling. Similarly, it must also consider the more complex task of redundancy management and cycling of spare units. The uniprocessor operating system can also have different variations depending upon the level of redundancy and the I/O configuration.

The Local Operating System in each computer interfaces with the Network Operating System. The Network Operating System is responsible for system level functions. These include an orderly start and initialization of various buses and networks, communication between processes executing in different computers, system level resource management, and system level redundancy management. System level resources are the GPCs, the I/O, IC, and the MM buses, and the shared data and programs stored in the mass memory or in some other commonly accessible location. System redundancy management includes FDIR in the I/O and IC node networks, correlation of faults in GPCs (both transient and hard faults), reassignment of computers to functions (function migration), and graceful degradation in case of a loss of a processing site.

Some of the functions of the Network Operating System are centralized the Global Computer. The Global Computer is responsible for system start, resource management, redundancy management, and function migration. It needs status knowledge of all processing sites and it must be able to command other GPCs to perform specific functions. This communication is accomplished via the Network Operating System, a portion of which is resident in each computer. The Global Computer does not participate in every system level transaction. Some of the system level functions performed by the Network Operating System may involve only a pair of nonglobal GPCs.

One of the GPCs is designated to be the Global Computer at the system bootstrap time. However, this designation can be changed during system operation by an internal or an external stimulus.

2.5 DATA BASE

Another important attribute of a distributed system is the treatment of the data base. The data base can be completely replicated in all subsystems or it can be partitioned among the subsystems. In addition, the data base directory can be centralized in one subsystem, duplicated in all subsystems, or partitioned among the subsystems. The AIPS approach is a combination of these.

For the mass memory data base, all GPCs will contain a directory of the MMU contents. This can be implemented as a 'directory to the directory' in order to limit the involvement of GPCs in the directory change process. The MMU directory will be static over extended intervals.

The data base that reflects the global system state will be maintain ed by the Global Computer in its local memory. A copy will be maintained by any alternate Global Computer, also in local memory.

The data base that reflects the distribution of functions among GPCs will be contained in all GPCs.

2.6 FAULT TOLERANCE

There is a considerable amount of hardware redundancy and complexity associated with each of the elements shown in Figure 1. This redundancy allows each hardware element to be reliable, fault tolerant, and damage tolerant. From a software viewpoint, however, this underlying complexity of the system is transparent. This is true not only in the context of the applications programs but for most of the operating system as well; however, those elements of the operating system that are concerned with fault detection and recovery and other redundancy management functions have an intimate knowledge of the underlying complexity.

Hardware redundancy in the AIPS is implemented at a fairly high level, typically at the processor, memory, and bus level. There are two fundamental reasons for providing redundancy in the system: one, to detect faults through comparison of redundant results, and two, to continue system operation after component failures. Processors, memories, and buses are replicated to achieve a very high degree of reliability and fault tolerance. In some cases coded redundancy is used to detect faults and to provide backups more efficiently than would be possible with replication.

The redundant elements are always operated in tight synchronism which results in exact replication of computations and data. Fault detection coverage with this approach is one hundred per cent once a fault is manifested. To uncover latent faults, temporal and diagnostic checks are employed. Given the low probability of latent faults, the checks need not be run frequently.

Fault detection and masking are implemented in hardware, relieving the software from the burden of verifying the correct operation of the hardware. Fault isolation and reconfiguration are largely performed in software with some help from the hardware. This approach has flexibility in reassigning resources after failures are encountered, and yet it is not burdensome since isolation and reconfiguration procedures are rarely invoked.

2.7 DAMAGE TOLERANCE

One of the AIPS survivability related requirements is that the information processing system be able to tolerate those damage events that do not otherwise impair the inherent capability of the vehicle to fly, be it an aircraft or a spacecraft.

The requirement for damage tolerance will be applied to redundant GPCs, intercomputer communications, and to communication links between GPCs and sensors, effectors, and other vehicle subsystems.

The internal architecture of the redundant computers supports the damage tolerance requirement in several ways. The links between redundant channels of a computer are point-to-point. That is, each channel has a dedicated link to every other channel. Second, these links can be several meters long. This makes it possible to physically disperse redundant channels in the target vehicle. The channel interface hardware is such that long links do not pose a problem in synchronizing widely dispersed processors.

For communication between GPCs and between a GPC and I/O devices a damage and fault tolerant network is employed. The basic concept of the network is as follows.

The network consists of a number of full duplex links that are interconnected by circuit switched

nodes to form a conventional multiplex bus. In steady state, the network configuration is static and the circuit switched nodes pass information through them without the delays which are associated with packet switched networks. The protocols and operation of the network are identical to a multiplex bus. Every transmission by any subscriber on a node is heard by all the subscribers on all the nodes just as if they were all linked together by a linear bus.

The network performs exactly as a virtual bus. However, the network concept has many advantages over a bus. First of all, a single fault can disable only a small fraction of the virtual bus, typically a link connecting two nodes, or a node. The network is able to tolerate such faults due to a richness of interconnections between nodes. By reconfiguring the network around the faulty element, a new virtual bus is constructed. Except for such reconfigurations, the structure of the virtual bus remains static.

The nodes are sufficiently smart to recognize reconfiguration commands from the network manager which is one of the GPCs. The network manager can change the bus topology by sending appropriate reconfiguration commands to the affected nodes.

Second, weapons effect induced damage or other damage caused by electrical shorts, overheating, or localized fire would affect only subscribers in the damaged portion of the vehicle. The rest of the network, and the subscribers on it, can continue to operate normally. If the sensors and effectors are themselves physically dispersed for damage tolerance or other reasons and the damage event does not affect the inherent capability of the vehicle to continue to fly, then the control system would continue to function in a normal manner or in some degraded mode as determined by sensor/effector availability. The communication mechanism, that is, the network itself, would not be a reliability bottleneck.

Third, fault isolation is much easier in the network than in multiplex buses. For example, a remote terminal transmitting out of turn, a rather common failure mode, can be easily isolated in the network through a systematic search where one terminal is disabled at a time. This, in fact, is a standard algorithm for isolating faults in the network.

Fourth, the network can be expanded very easily by adding more nodes. In fact, nodes and subscribers to the new nodes (I/O devices or GPCs) can be added without shutting down the existing network. In bus systems, power to buses must be turned off before new subscribers or remote terminals can be added.

Finally, there are no topological constraints which might be encountered with linear or ring buses.

2.8 SOURCE CONGRUENCY

An important consideration in designing AIPS is the interface between redundant and simplex elements. This interface design is crucial in avoiding single point faults in a redundant system. One must perform source congruency operations on all simplex data coming into a redundant computer. It is not sufficient to distribute simplex data to redundant elements in one step. The redundant elements must exchange their copy of the data with each other to make sure that every element has a congruent value of the simplex data. The AIPS architecture not only takes this requirement into account but also provides efficient ways of performing simplex source congruency through a mix of hardware and software. The simplex to redundant interface is also the place where the applications programmer gets involved in the processor redundancy and the applications code complexity multiplies. The AIPS processor level architecture is designed such that it separates the source congruency and computational tasks into two distinct functional areas. This reduces the applications code complexity and aids validation.

2.9 MASS MEMORY

The mass memory in AIPS provides the following capabilities.

1. System Cold Start/Restart.
2. Function Migration Support.
3. Overlays for local memory of General Purpose Computers.
4. System Table Backup.
5. Storage for system-wide common files.
6. Program Checkpointing.

3.0 PROOF-OF-CONCEPT SYSTEM

To demonstrate feasibility of the Advanced Information Processing System concept described in the preceding sections, a laboratory proof-of-concept system will be built. Such a system is now in the detailed design phase. The POC system configuration is shown in Figure 2. It consists of five processing sites which are interconnected by a triplex circuit switched network. Four of the five GPCs are uniprocessors, one simplex, one duplex, and two triplex processors. The fifth GPC is a multiprocessor that uses parallel hybrid redundancy. The redundant GPCs are to be built such that they can be physically dispersed for damage tolerance. Each of the redundant channels of a GPC could be as far as 5 meters from other channels of the same GPC.

Each of the triplex fault tolerant processors (FTPs) and the fault tolerant multiprocessor (FTMP) interfaces with three nodes of the Intercomputer (IC) node network. The duplex and the simplex processors interface with two and one nodes, respectively.

The mass memory is a highly encoded memory that interfaces with the GPCs on a triplex multiplex bus.

The Input/Output is mechanized using a 16 node circuit switched network that interfaces with each of the GPCs on 1 to 6 nodes depending on the GPC redundancy level.

Redundant system displays and controls are driven by the Global Computer and interface through the I/O network.

Each GPC has a Local Operating System and a portion of the Network Operating System. For the proof-of-concept system, initially the FTMP will be

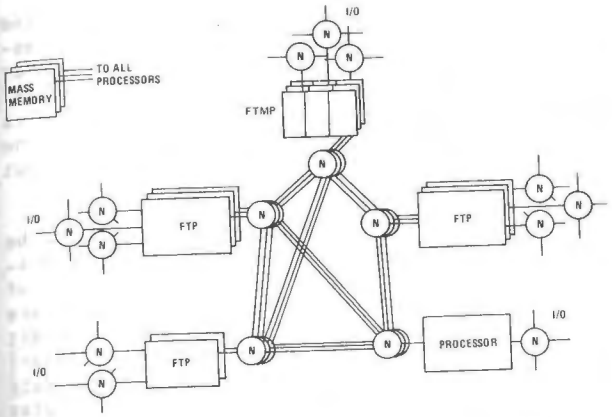


Figure 2. AIPS Proof-of-Concept System Configuration

the Global Computer.

3.1 ARCHITECTURE OF AIPS BUILDING BLOCKS

Architecture of the major hardware building blocks of the AIPS Proof-of-Concept System configuration is described in the following sections.

3.1.1 Fault Tolerant Processor

The architectural description of the FTP is divided into three sections: Software View, Hardware View, and External Interfaces.

3.1.1.1 Fault Tolerant Processor: Software View

The FTP or the uniprocessor architecture from a software viewpoint appears as shown in Figure 3.

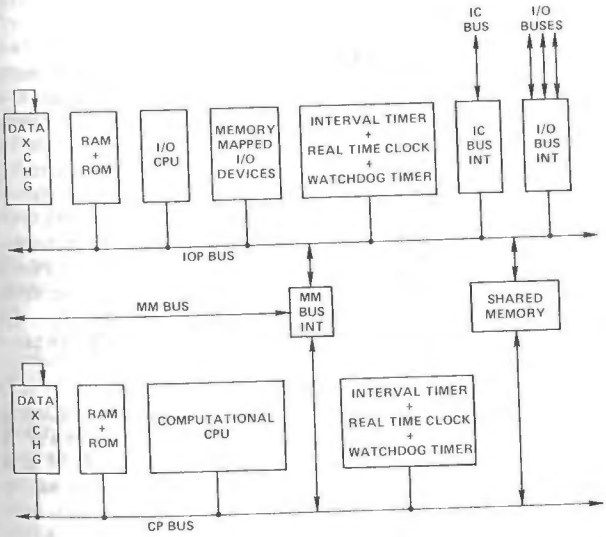


Figure 3. Fault Tolerant Processor Architecture: Software View

The uniprocessor can be thought of as consisting of two separate and rather independent sections: the computational core and the Input/Output channel.

The computational core has a conventional processor architecture. It has a CPU, memory (RAM and ROM), a Real Time Clock, and interval timer(s). The Real Time Clock counts up and can be read as a memory location (a pair of words) on the CP bus. Interval timers are used to time intervals for scheduling tasks and keeping time-out limits on applications tasks (task watchdog timers). An interval timer can be loaded with a given value which it immediately starts counting down and when the counter has been decremented to zero, the CPU is interrupted with a timer interrupt. A watchdog timer is provided to increase fault coverage and to fail-safe in case of hardware or software malfunctions. The watchdog timer resets the processor and disables all its outputs if the timer is not reset periodically. The watchdog timer is mechanized independently of the basic processor timing circuitry.

There also appears on the processor bus a set of registers, called the data exchange registers. These are used in the redundant fault tolerant processor to exchange data amongst redundant processors. From a software viewpoint, this is the only form in which hardware redundancy is manifested.

On a routine basis the only data that needs to be exchanged consists of error latches and cross channel comparisons of results for fault detection. These operations can be easily confined to the program responsible for Fault Detection, Isolation, and Reconfiguration. Voting of the results of the redundant computational processors is performed by the Input/Output processors. Therefore, the remaining pieces of the Operating System software and the applications programs need not be aware of the existence of the data exchange registers. The task scheduler and dispatcher, for example, can view the computational core as a single reliable processor.

The other half of the processor is the Input/Output channel. The I/O channel has a CPU (same instruction set architecture as the CP), memory (RAM and ROM), a Real Time Clock, and an Interval Timer(s). This part of the I/O channel is identical to the CP except that it has less memory than the CP.

The IOP has interfaces to the intercomputer bus, one or more I/O buses, and memory mapped I/O devices. The CP and the IOP also have a shared interface to the system mass memory. These external interfaces of the FTP will be discussed in the next two sections.

The IOP and CP exchange data through a shared memory. The IOP and CP have independent operating systems that cooperate to assure that the sensor values and other data from Input devices is made available to the control laws and other applications programs running in the CP in a timely and orderly fashion. Similarly, the two processors cooperate on the outgoing information so that the actuators and other output devices receive commands at appropriate times. This is necessary to minimize the transport lag for closed loop control functions such as flight control and structural control.

The CP and IOP actions are therefore synchronized to some extent. To help achieve this synchronization in software, a hardware feature has been provided. This feature enables one processor to interrupt the other processor. By writing to a reserved address in shared memory the CP can interrupt the IOP and by writing to another reserved location the IOP can interrupt the CP. Different

meanings can be assigned to this interrupt by leaving an appropriate message, consisting of commands and/or data, in some other predefined part of the shared memory just before the cross-processor interrupt is asserted.

For routine flow of information in both directions, the shared memory will be used without interrupts but with suitable locking semaphores to pass a consistent set of data. The interrupts can be used to synchronize this activity as well as to pass time critical data that must meet tight response time requirements. In order to assure data consistency it is necessary that while one side is updating a block of data the other side does not access that block of data. This can either be implemented through semaphores in software or through double buffering. Hardware support for semaphores, in the form of test & set instruction, is provided in the IOPs and CPs.

There are many attractive features of this architecture from an operational viewpoint. The most important of these is the decoupling of computational stream and the input/output stream of transactions. The computational processor is totally unburdened from having to do any I/O transaction. To the CP all I/O appears memory mapped. And this not only includes I/O devices but also all other computers in the system as well. That is, each sensor, actuator, switch, computer, etc. to which the FTP interfaces can simply be addressed by writing to a word or words in the shared memory.

Data from other processing sites is received by each IOP on the redundant IC buses, hardware voted, and then deposited in their respective shared memories. Simplex source data such as that from I/O devices, local processors, etc. is received by the single I/O processor that is connected to the target device. This data is then sent to the other two I/O processors using the IOP data exchange hardware. The congruent data is then deposited in all three shared memory modules. In either case, the computational processors obtain all data from outside that has already been processed for faults and source congruency requirements by the I/O processors.

The data exchange mechanism appears to the software as a set of registers on the processor bus. Data exchange between redundant processors takes place one word at a time. Two types of data exchanges are possible: a simplex exchange or a voted exchange. The purpose of a simplex exchange is to distribute congruent copies of data that is available only in one channel of the FTP to all other channels. The purpose of a voted exchange is to compare and vote computational results produced by redundant processors. In the FTP architecture, these exchanges are mechanized as follows.

To perform a voted exchange, each processor writes the value to be voted in a transmit register called X_V. Writing to this register initiates a sequence of events in hardware which culminates with the voted value being deposited in the receive register of each processor. The processor can read the receive register at this point to fetch the voted value. The whole transaction takes of the order of 5 microseconds. The hardware is designed to lock out access to the receive register while the exchange is in progress. If the processor tries to read the receive register before the transaction has completed, the processor hangs up. As soon as the data becomes available, the processor is released and the register read cycle completes normally. The processor wait is transparent to the software. It is not necessary to time the interval between writing of the transmit register and reading of the receive register in software. The two operations can be performed as a sequence of two instructions without an intermediate wait.

To perform a simplex exchange, the data to be transmitted is written to one of the simplex transmit registers. In the triple redundant version of the FTP there are three such registers. They are called X_A, X_B, and X_C. X_A is used to transmit simplex data from channel A to all others. Similarly X_B transmits data from B and X_C transmits data from C. Writing to one of these registers initiates a sequence of events in hardware which culminates with a congruent copy of the data word being deposited in the receive register of each processor. The receive register can be read at this point by each processor to fetch the congruent copy of the simplex data.

It has been pointed out earlier that the software appearance of the redundant FTP is the same as that of a simplex processor. All redundant processors have identical software and execute identical instructions at all times. This architecture is carried forth in the data exchange hardware and software as well. The data exchange hardware is designed such that all redundant processors execute identical instructions when exchanging data. As an example, consider a simplex source transmission from channel A. Assume that channel A has a sensor value in its internal memory location, called MEMORY, that it needs to send to channels B and C. This requires execution of the following sequence of four instructions:

```
1    LOAD  R0,MEMORY
2    STORE R0,X_A
3    LOAD  R0,X_R
4    STORE R0,MEMORY
```

The data to be transmitted is fetched from memory (instruction 1) and written to transmit register X_A (instruction 2). All three processors execute these instructions. However, only processor A's value is transmitted to the receive register of A, B, and C. Transmissions from B and C are ignored by the hardware. This will be explained in the next section which deals with the FTP architecture from a hardware viewpoint. In instruction 3 all processors read their receive register (X_R) to accept the congruent value of the data transmitted by A. In instruction 4 this value is transferred to an internal memory location.

Voted data exchange requires a similar sequence of instructions. The only difference is that in instruction 2, rather than storing the value in one of the simplex transmit registers, it is stored in the voted exchange register, X_V.

3.1.1.2 Fault Tolerant Processor: Hardware View

The triplex FTP architecture from a hardware viewpoint appears as shown in Figure 4.

There are three identical hardware channels. Each channel has a computational processor, an I/O processor, and some hardware that is shared by the CP and the IOP. The internal details of the CP and the IOP such as the CPU, memory, timers, etc., have

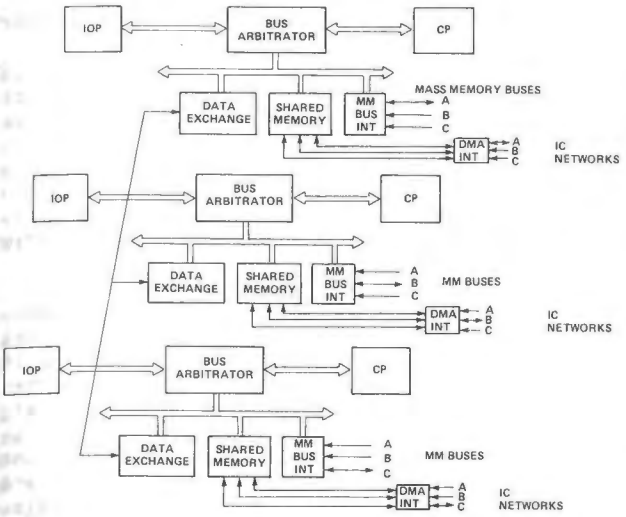


Figure 4. Fault Tolerant Processor Architecture: Hardware View

been described in the preceding section. They are not shown in Figure 4 so that other details such as the redundancy dimension be shown more clearly.

The common hardware consists of a shared memory, the data exchange registers, and the mass memory interface. The shared memory is used exchange information between the CP and the IOP while the data exchange registers are used to exchange information between redundant copies of the CP or IOP. Common hardware access conflicts between the CP and the IOP are resolved by a bus arbitrator. The bus arbitration logic is designed such that each channel resolves the conflict in favor of the same processor (that is, either the IOP or the CP) deterministically. This is necessary to maintain tight synchronism between redundant copies of processors. This is only one of several conditions necessary for synchronous operation. Stated in more general terms, two hardware conditions are necessary. First of all there should be a common time base that is used by all channels for timing events. Second, all timing events should be deterministic in nature. If these two hardware conditions are met, the redundant channels can be synchronized. Once they are synchronized, they will stay synchronized if all channels execute identical software.

To obtain a common time base, the oscillators in redundant channels are phase locked to each other. To assure that all timing events are deterministic, it is necessary to use a synchronous bus internal to each processor. As an example, when a CPU references a memory location on the bus the memory cycle should complete in a fixed time interval. This does not necessarily imply that all memory cycles should take the same time. It is possible in the FTP architecture to mix different types of memories, such as PROM and RAM, which may have different access times. The only necessary condition is that access to a given location always take the same length of time. This also applies to any I/O activity performed by the processor. The hardware is built such that when a processor accesses a device available in only one channel the other processors wait the same length of time.

A very important aspect of the FTP architecture is the interconnection hardware between redundant channels. This hardware serves three purposes. First of all, it provides a path for distributing simplex data available in only one channel to all other channels. Second, it provides a mechanism for comparing results of the redundant channels. And third, it provides a path for distributing and comparing timing and control signals such as the fault tolerant clock and external interrupts.

To distribute simplex data from one channel to all others without introducing single point faults in the design, it is necessary to adhere to source congruency requirements. One of these dictates that in order to tolerate single faults it is necessary to provide four fault containment regions. In the triplex FTP architecture six fault containment regions are provided. The triplex processor provides the basic three fault containment regions. Three additional regions are provided in the form of interstages which receive data from processors and rebroadcast them back to processors. The interstages are mechanized such that they have independent voltage and timing reference. This assures that faults in processors would not propogate to interstages and vice versa. Since an interstage is essentially a buffer with receivers and transmitters, it is relatively a small and simple piece of electronics. It is, therefore, much more convenient to provide three additional fault containment regions rather than just one as required for source congruency. It also makes the FTP architecture symmetric.

As explained in the preceding section, the data exchange hardware appears as a set of five registers on the processor bus. Four of these (X_A, X_B, X_C, and X_V) are the transmit registers and the fifth one is the receive register, X_R. For simplex source exchanges, say, a 'from A' exchange, data in X_A register in channel A is transmitted to the three interstages. The interstages rebroadcast this data to every processor. The three copies received by each processor are voted in hardware on a bit-by-bit basis. The voted result is deposited in X_R. For voted exchanges, each channel writes the data to be voted in X_V register. Writing to X_V results in the data being transmitted to the channel's own interstage. The second half of the operation is the same as for simplex exchange. In both cases, the exchange hardware masks any single faults while voting on three copies and also records the source of fault in an error latch. The error latch can be read by software as a memory location.

3.1.1.3 Fault Tolerant Processor: External Interfaces

The external devices that interface with the FTP are the mass memory, the Intercomputer network, and the I/O network.

Figure 4 shows the interface between a triplex FTP and the triply redundant mass memory bus. This interface hardware is shared in each channel by the CP and the IOP. Each channel of the FTP is enabled on one of the three buses. The FTP transmits commands and data synchronously on three buses to the mass memory where they are received and voted in hardware. The interface hardware performs the necessary parallel to serial data conversion, appends cyclic redundancy check byte (CRC), and transmits serial data on the bus. Each processor channel listens to all three mass memory buses. Data received

from the mass memory is voted in hardware on a bit-by-bit basis. Any disagreements on the mass memory bus are recorded in error latches for later analyis by software. Any CRC failures are also recorded separately. Voted data is then converted from serial to parallel format.

Figure 5 shows the interface between a triplex FTP and the triply redundant Intercomputer Network.

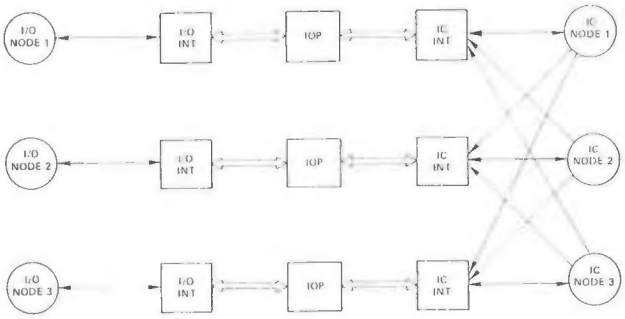


Figure 5. FTP Interface to IC and I/O Networks

The IC network interface is dedicated to the I/O processor. Other than that, it is very much like the mass memory interface. Each IOP listens to all three IC networks but can transmit on only one IC network. The IC network interface circuitry is responsible for parallel to serial data conversion, resolving network contention, and transmitting data on the network. In the other direction, the network interface hardware listens to three bit streams, deskews them, votes and masks single faults and converts the voted bit stream from serial to parallel. It stores any disagreements on the networks in error latches or registers. Non compliance with the network protocol is also recorded separately for each of the three networks.

An IOP also interfaces to one or more I/O networks. This interface is different from the IC network interface to the extent that the I/O networks may not be redundant. Redundant I/O networks interface with the FTP the same way as the IC network nodes. For simplex I/O networks, it is necessary that when the processor is communicating with an I/O device all other processors execute the same software and wait identical lengths of time to stay synchronized. Also, any data received from the I/O devices must be distributed to all other processors using the data exchange registers. Although an I/O network may not be redundant, an FTP may have more than one connection to an I/O network through multiple IOPs.

An I/O network may be dedicated so that only those I/O devices that are used solely by this FTP are on this network. Or the I/O network may be a shared network that connects multiple computers to shared I/O devices.

Finally, an IOP may also have local dedicated I/O devices that can be accessed directly by the IOP as memory locations. The memory mapped I/O may consist of local switches, discretes, A/Ds, D/As and interrupt driven devices. This interface differs from dedicated I/O bus interface in the sense that I/O signals on the bus may be already conditioned and processed by a local processor and the IOP interfaces through this bus to the local processor which may control a number of I/O devices.

Although it is possible to interface interrupt driven I/O devices to the IOPs, none will be included in the proof-of-concept system.

An IOP may transmit on an IC bus, a shared I/O bus, or a dedicated I/O bus only if it is enabled to do so by a majority of the IOPs. An IOP can also disable itself any time.

The interface of an FTP to the IC buses is somewhat different if the FTP redundancy level is not the same as the IC redundancy level. For example, if the FTP is a duplex system rather than a triplex then there will be only two IOPs, one IOP per channel. Each IOP will listen to and vote on all three IC buses and it will transmit on one IC bus. Transmissions from duplex processors will, therefore, be heard only on two out of three buses. Similarly, simplex processors will listen to all buses but will transmit on only one bus. Voting logic in the bus interface circuits is suppressed when the transmitting processor is simplex. It is necessary to rely on compliance with bus protocol to detect errors.

The next topic of discussion is the interface of the system in degraded mode. The FTP can degrade in 3 ways, failure of an IOP, failure of a CP, and the failure of the shared hardware.

When an IOP fails, the IC bus interface would degrade from triplex to a duplex configuration. Each of the other two IOPs would listen to all three IC buses but transmissions from this FTP would be heard by other computers only on two buses. At the receiving sites fault masking would be replaced by fault detection. Since there is some inherent coded redundancy in data being transmitted on each bus, one can hope to identify the second fault with fairly high coverage. This would normally be the case when communicating to a dual-redundant computer on the network under normal circumstances.

The other effect of the failure of an I/O processor is the loss of I/O devices attached to that processor. Actually only those I/O devices that were connected only to the failed processor would be lost. If the I/O devices are cross-strapped to other IOPs through the I/O bus, for example, they would still be accessible via the other I/O processors.

Finally, the loss of an IOP also means that the CP attached to that IOP does not have access to any inputs. In other words, failure of any element in a channel can be considered the same as failure of the whole channel. That is, if an IOP, a DPM or a CP fails in a channel one could shut down all the elements in that channel of the FTP. Although this is the most convenient way to operate the system from a system software viewpoint, one can use more sophisticated strategies to obtain reliability from the system. This, however, is achieved at the expense of more complex system software.

Specifically, if an IOP fails then the corresponding CP loses access to data being provided by that IOP through the shared memory. But since the CPs are cross-strapped to each other through data exchange hardware it is possible to provide the target CP the voted value of the inputs to the other two CPs. The same can be done on the IOP side when a CP fails. If the shared memory fails either side can get around the fault by cross-exchanging data. This

will be the POC system operational strategy.

3.1.2 Fault Tolerant Multiprocessor

The following two sections describe the software appearance of the multiprocessor and the redundancy and fault tolerance dimension of the machine, respectively.

3.1.2.1 Multiprocessor Software Appearance

The multiprocessor discussion is divided into three parts: processors, shared memory, and the external interfaces. These three parts are discussed in the following sections.

3.1.2.1.1 PROCESSORS The multiprocessor architecture from a software viewpoint appears as shown in Figure 6. This figure does not show the redundancy dimension of the computer.: FTMP, from a software viewpoint, appears as a conventional, homogeneous multiprocessor. There are a number of processing elements that have access to a common memory, called the shared memory. Each processor has a local memory which is composed of Read Only Memory (ROM) and Read/Write Memory. Although not all of the programs are always resident in the local memory, they must be loaded there from the shared memory before they can be executed. The local PROM is used to hold the bootstrap loader, cold start and restart programs, frequently executed parts of the operating system, and high frequency applications programs. All other programs are loaded into the RAM on a demand basis.

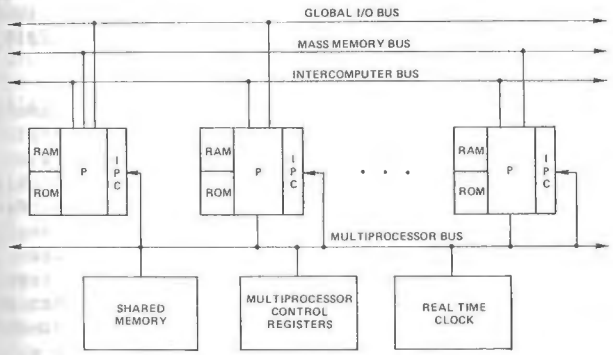


Figure 6. FTMP Architecture: A Software View

Each processor has internal interval timer(s) that can be set to any 16-bit value under program control. The interval timer decrements this count and interrupts the processor when the count reaches zero. Interval timers can be used for scheduling highest frequency tasks and also as task watchdog timers.

Communication between processors can be via the shared memory or it can be via Interprocessor Communication (IPC) Buffers. The shared memory path is the slower of the two paths. The sender can write the message in the shared memory, but it does not arrive at its destination until the receiving processor reads its mail box. An alternative to this is the IPC Buffer. A processor can write to any other processor's buffer. The receiving processor is interrupted when one of the buffer locations is written into. The receiving processor's IPC Interrupt Handler can then read the message in its buffer. The buffer length is of the order of 16 words. That is, it is not a large memory array. For large data transfers, shared memory would still be used although one could pass a shared memory pointer using the IPC mechanism.

3.1.2.1.2 SHARED MEMORY : The shared memory can be accessed by the processors using the multiprocessor bus. Contention for the bus amongst multiple processors is resolved in hardware. The hardware also provides the capability to test and set a word in the shared memory in a single atomic operation. Access to data elements that are shared amongst multiple processes can be limited to one process at a time by associating a lock or semaphore with each shared data set.

Although the shared memory appears as a single monolithic unit to the software, it should be noted here that it is, in fact, composed of several segments. Such a segmented shared memory, coupled with dedicated buses from processors to memory units, makes it appear as a multiported memory. All segments of the shared memory can be accessed simultaneously if different processors happen to request access to different segments. This feature should be taken into account when locking data sets in the shared memory.

There are several other elements of the multiprocessor that can be accessed by the processors using the memory bus. One of these is the Real Time Clock (RTC). The Real Time Clock is a 32-bit counter that counts up. The RTC can be set to a given value by writing to its 'memory' address. It can be read as a two word value on the memory bus also. Reading the high order word automatically latches the low order word so that the 32-bit value of the RTC is read as a consistent set.

IPC buffers also appear as shared memory addresses as far as write operations are concerned. They can only be written to on the multiprocessor bus. They can be read only by the host processor on the internal processor bus.

Other registers in the shared memory address space can be grouped under the heading of Multiprocessor Control Registers. Their functions include:

1. Memory Relocation (Write Only): This assigns a shared memory module to a given address space.

2. Triad Identification (Write Only on multiprocessor bus, Read Only on processor bus): This assigns a processor to a triad.

3. CPU Control (Write Only on multiprocessor bus, Read Only on processor bus): This controls various CPU operations such as reset, go, etc.

4. Bus Selection (Write Only on multiprocessor bus): This tells each processor and memory which buses to listen to.

5. Error Latch (Read Only on system bus): It records disagreements on the multiprocessor bus.

3.1.2.1.3 EXTERNAL INTERFACES : External interfaces of the FTMP consist of interfaces to other

GPCs through IC nodes, interface to I/O devices through I/O nodes, and interface to Mass Memory through Mass Memory buses. Not all FTMP processors have all external interfaces. Some processors may, in fact, have none of these interfaces. These processors perform only computational function in the FTMP.

At any given time, one processor triad is assigned to communicate on the IC network. Bus interface hardware performs a bit by bit majority vote on incoming redundant data. Each processor then should have an identical copy of the incoming data. For simplex data, the voting is bypassed and it is necessary to perform source congruency. This is done by simply writing the simplex data into shared memory. Voters in the shared memory perform majority voting on three copies of the word received from the three processors. The voted data then becomes the congruent value of the simplex word.

Mass memory interface is functionally identical to the IC network interface.

The FTMP also has interfaces to one or more I/O networks. The I/O networks may or may not be redundant. One of the processor triads, with approriate I/O interfaces, is assigned to the task of managing I/O devices.

3.1.2.2 Multiprocessor Redundancy & Fault Tolerance Features

As alluded to in the previous section, there is a considerable amount of complexity in the hardware that is largely transparent to the software but is responsible for making the machine fault tolerant. This complexity arises from two related features of the multiprocessor. One, every element in the system is replicated to some level. Every major element is at least triplicated, and some have even a higher level of redundancy. Two, all redundant operations must be compared to detect faults and to mask faults where appropriate. The fault detection and masking requires considerable amount of interconnections between redundant elements. These two attributes of the machine, viz., redundancy and intercommunication, are largely responsible for the multiprocessor complexity.

However, the redundant hardware elements are organized in such a fashion that this complexity is not carried over into the software. In fact, the other attribute of the machine, viz., the interconnection of the redundant elements, is what makes the hardware complexity transparent to the user. This should become clearer as the hardware architecture is described in the following sections.

3.1.2.2.1 PROCESSORS : Processor in the multiprocessor, CPs or IOPs, operate in groups of three, called triads. The three members of a triad are tightly synchronized using a fault tolerant clock. (The clock operation is described in another section.) Processor organization is such that any three processors can be formed into a triad with some exceptions such as the constraints that may be introduced by packaging and external bus interface considerations. Other than these constraints, a processor element can be used as a member of a CP triad or an IOP triad. A processor may be a member of a CP triad at one time and it may be a member of an IOP triad at some other time.

Once all the available processors have been formed into CP and IOP triads, the remaining processors (spares), can be used to 'shadow' normally operating triad members. A shadow processor is tightly synchronized with the three active members of the triad and executes the same instructions as the active members. It listens to all the buses to obtain the same input data as the active members. However, it is not enabled to transmit on any bus.

Each processor has a number of control registers which are either processor specific or triad specific. All of these registers have a 'Write To' address that is an extension of the shared memory address space. They also have a 'Read From' address that is an extension of the processor's local memory address space. An example of a triad specific register is the IPC (Interprocessor Communication) Buffer. An example of a processor specific register is the Triad Identification (ID) Register. Addresses of processor specific registers have a Processor ID field in the 'Write To' address. Addresses of triad specific registers have a Triad ID field in the 'Write To' address.

There are a number of other system control and status registers that are accessed in a manner similar to the processor control registers. Examples of these will be given where relevant.

3.1.2.2.2 SHARED MEMORY : The shared memory in the multiprocessor is triplicated and operates as one contiguous triad. Physically, it is partitioned into several smaller segments and the level of replication is at the segment level. A processor and a shared memory segment are packaged together in a box or Line Replaceable Unit (LRU). They share such items as the fault tolerant clock, LRU power supply, etc.

Associated with each memory segment is a Memory Relocation Register (MRR). This register is analogous to the Processor Triad ID Register in that it allows one to identify the memory triad to which a memory module belongs. The MRR forms the high order part of the address to which the associated memory module responds. The MRR itself has an LRU specific address and can be written to on the multiprocessor bus. Thus, one can group any three memory modules to form a shared memory triad by relocating them to the same address space.

3.1.2.2.3 PROCESSOR-MEMORY INTERFACE : The processors and shared memory segments are fully cross-strapped. Each box or LRU containing a processor and a shared memory is connected to every other LRU using dedicated buses. The processor-shared memory communication works as follows.

Assume there are N LRUs in the multiprocessor. When a processor triad wishes to write shared memory, each member broadcasts appropriate address, data, and commands on its transmit bus. Each member of the target memory triad receives three copies of address, etc. It selects 3-out-of-N transmit buses to listen to this processor triad using information from Bus Select registers. The three copies are then voted by each member of the memory triad and appropriate action (such as storing the data) taken. Any disagreements are latched in the Processor Error Latches indicating the identity of the disagreeing processor.

For a memory read operation, the first half of the transaction is similar to the write operation.

Each member of the memory triad broadcasts the data to the three requesting processors. Each processor in the requesting triad then receives three copies of the data which it votes in hardware and also latches identity of any disagreeing memory unit in its Memory Error Latch.

As indicated earlier there are a number of multiprocessor control and status registers. More examples of these have now been cited (MRR, Error Latches, Bus Select Registers). These registers are assigned addresses in an extended shared memory address space. They can be accessed by any processor triad just as if they were shared memory locations. The interface hardware to select Transmit Buses, vote on incoming processor requests, etc. can be the same hardware that is used to access the shared memory in a given LRU. This not only saves hardware but also makes use of the existing processor-memory 'bus' that cross-straps all LRUs. In fact, this same communication medium can be used to write to the Interprocessor Communication Buffers. Such an arrangement also makes the software appearance of the machine rather straightforward.

3.1.2.2.4 FTMP CLOCKING : The tight synchronization between processor elements is maintained using standard fault tolerant clocking techniques (analog phase locked loop or digital compensation) developed previously. The multiprocessor fault tolerant clock functions as follows.

Each processor and shared memory in every LRU has a common oscillator. Four of these oscillators, called the active elements, are chosen to form the quad-redundant fault tolerant clock. (Four clock elements are necessary to tolerate all single point clock failures). Any four operating oscillators can be chosen as active elements. The active clocks are distributed to every processor and memory LRU in the system. Each active element listens to the other three active clocks and locks itself to the majority. The nonactive clocks phase lock to any three out of four active clocks. If an active elements fails, it is replaced by a previously inactive element. In other words, every clock is sent to every LRU in the system. Since the basic philosophy is to have dedicated paths rather than multiplex buses, there is a dedicated clock bus that goes from every LRU in the system to every other LRU.

Each LRU in effect listens to three active elements and synchronizes itself to the majority. This clock is then used for all internal timing events such as processor clocking, memory clocking, decrementing of the interval timer, and incrementing of the real time clock.

The real time clock is a 32-bit counter. Such a counter exists in every LRU and is accessible on the system memory bus. Real time clocks in all the LRUs respond to the same system memory address. RTC counters always respond to write (or set) requests. Thus all the real time clocks in the FTMP can be set to a given time simultaneously. Once set, they all count up at the same rate since they are clocked by the fault tolerant clock. During normal operation, three of the counters are selected to be active. There is an RTC Select Register in every LRU that determines which three RTCs to listen to. When one of the counters fails, it is replaced by another operating RTC counter by updating all RTC Select Registers in all LRUs.

3.1.2.2.5 EXTERNAL INTERFACES : The multiprocessor interfaces to the external world through three different types of buses. Interface to other General Purpose Computers is via the IC network. Interface to the I/O devices is via the I/O network. And interface to the Mass Memory is via the MM bus.

The IC network is triply redundant and consists of three layers of a circuit switched node network. Three of these nodes, one from each layer, interface with the multiprocessor. On the multiprocessor side one processor triad interfaces with the triplex IC network (also referred to as the IC bus) at any given time. For data transmission, each member of this triad (called the IC triad) transmits on one bus. For receiving data, each IC triad processor listens to all three IC buses, deskews the data and votes on the three copies in hardware. Any bus disagreements trigger IC Error Latches in the processor-bus interface hardware. The identity of the disagreeing bus is stored in the Error Latch. Before performing bit by bit voting on incoming data, each serial data stream is checked for compliance with the bus protocol. Any deviations are recorded in error registers. The error latches and the protocol error registers can be read by a processor triad on the multiprocessor bus. If the data source GPC is not redundant, then voting circuitry is bypassed. Simplex source congruency is performed on the incoming data by writing it to shared memory and reading it back again. Since the IC bus is a contention bus, the FTMP contends for access to the bus with the other GPCs using a distributed arbitration algorithm known as the Laning Poll. One of the FTMP triads that is enabled on the IC bus participates in this poll.

The I/O bus is a single layer, circuit switched network. At any given time, one processor triad (called the IOP triad) is assigned to interface with the I/O network. The IOP triad operates in a fashion similar to the IC triad. However, there is one major difference in that only one IOP processor transmits on the I/O bus at any given time. All three I/O processors listen to the I/O bus and perform source congruency on all incoming data by writing it to the shared memory. Since the I/O bus is a contention bus, they all participate in the bus arbitration by each member listening to the bus but only one of them actually transmitting on the bus. Laning Poll is used to arbitrate I/O bus conflicts.

Interface to the mass memory triplex bus is very much similar to the IC bus interface.

The transmissions from a processor on the IC, I/O and MM buses are gated through enabling gates. The purpose of the enabling gates is to protect the buses from runaway processors that can not otherwise be turned off. The enabling gates allow a processor to transmit only if a majority of the processors agree to do so. Each processor sends an enabling signal to every other processor. These enabling signals are voted upon to create a master enable signal in each LRU. If the multiprocessor is built with some slots initially unoccupied, the master enable signal is created by voting enable signals from only those processors which are present. Each LRU also produces a presence signal to indicate whether it is populated. The presence signal is also helpful in writing FDIR software and therefore should be made available as part of some LRU specific control register.

3.1.3 IC and I/O Networks

The circuit switched nodes of the IC and the I/O networks are identical. For the proof-of-concept system each node will have five identical ports. The node will interface with other nodes, GPCs, and I/O subscribers (displays etc.) through these ports. All ports will be identical in terms of their ability to interface with any of the aforementioned entities.

The Intercomputer communication network for the proof-of-concept system configuration consists of three identical layers of a circuit switched network. Each layer consists of five nodes. Each node services one GPC. Although it is possible to service several GPCs from one node provided the node has enough ports to do this, this is not the case for the POC system configuration.

The three layers of the IC network are totally independent and are not cross-strapped to each other. The initial no-fault configuration of the three layers is identical although it does not have to be so. That is, after a link failure in one layer the virtual bus configuration of that layer would change as the network is reconfigured around the failed link. The other two layers do not have to be reconfigured to make their virtual bus path identical to the third one. The fault detection, isolation, and reconfiguration of the IC network are the responsibility of the Global Computer. Nodes keep track of any transmission errors which are protocol related and inform the Global of these errors when queried by the Global. This error data is analyzed by the Global to determine source of transient faults on the network. The nodes also respond to status queries with the status of the node and the ports. Other than this, the nodes are totally passive circuit switching devices. They listen for node reconfiguration commands from all ports whether or not that port is active. Valid reconfiguration commands must be preceded by a Gateman code. Reconfiguration commands are addressed to individual nodes although they are heard by all nodes.

The principles of operation of the I/O network are same as that of the IC network. The I/O network configuration for the POC system consists of a single layer of 16 nodes. This network can be configured either as a single global I/O network or as several regional I/O networks. Both configurations will be used in the POC system to demonstrate the global and regional I/O bus features of the AIPS system. System displays and controls would be attached to the I/O nodes.

3.1.4 Mass Memory

The AIPS proof-of-concept system mass memory requirements can be satisfied by Winchester disks, magnetic bubble memory, or Electrically Erasable ROM (EEROM) semiconductor memory. The choice will be made during preliminary design. The mass memory redundancy scheme will depend on this choice. If the decision is to use the disk or the magnetic bubble memory the memory will be replicated. Each copy will interface to the triplex mass memory bus through its own bus interface circuitry. The interface will be responsible for receiving redundant commands, address, and data from GPCs and performing deskewing, voting, and fault detection on the incoming data. Voted data will be stored in the memory. Each interface would also respond to the memory read requests on one of the three mass memory buses. If the memory is implemented as EEROM, the fault tolerance will be provided through encoding rather than triplication.

The mass memory interfaces will restrict access to mass memory by simplex GPCs to read-only operations. A 'Mass Memory bus hog' capability will also be provided in support of semaphores or locks associated with shared data in the mass memory. A GPC will be able to retain control of the MM bus, after gaining access to it, for multiple memory transactions. One can read a memory location, modify it, and write it back as a single atomic operation.

4.0 SUMMARY

An advanced information processing architecture has been defined for a broad range of aerospace vehicles. It is highly fault tolerant and damage tolerant and has a number of other desirable attributes such as graceful degradation, growth, and an ability to accept technology upgrade. A proof-of-concept configuration is in detailed design phase. This will be followed by fabrication, programming, test and evaluation phases to prove key AIPS concepts.

LIST OF REFERENCES

1. AIPS System Requirements, Report CSDL-AIPS-83-50, C. S. Draper Laboratory, Cambridge, MA, August 1983.

2. AIPS System Specification, Report CSDL-C-5709, C. S. Draper Laboratory, Cambridge, MA, May 1984.

3. Philip H. Enslow, Jr., "Distributed Data Processing - What Is It?", AGARD-CP-303, ISBN 92-835-0302-3, AGARD Conference Proceedings #303, Norway, 22-25 June 1981.

ACKNOWLEDGEMENT

This report was prepared by The Charles Stark Draper Laboratory, Inc. under Contract NAS9-16023, Task Order #84-18, with the Lyndon B. Johnson Space Center of the National Aeronautics and Space Administration. Publication of this report does not constitute approval by the NASA/JSC of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

A FAULT-TOLERANT SOFTWARE STRATEGY FOR DIGITAL SYSTEMS

84-2646

Ellis F. Hitt*, and Jeffrey J. Webb**
Battelle Columbus Laboratories
Electronics Department
505 King Avenue
Columbus, Ohio 43201

Abstract

Fault-tolerant software techniques discussed in this paper are the subject of research at many universities and industry. The research is directed toward further development and enhancement of the known techniques, such as recovery blocks and multi-version programming, development of tools for implementing fault-tolerant software techniques, and development of new techniques. The sources of software faults are described. Descriptions of the primary techniques are followed by an assessment of the current state-of-the-art of fault-tolerant software. Research needed to further the development and application of fault-tolerant software is discussed.

Introduction

Fault-tolerant software techniques have been the subject of research for the past decade since it became apparent that the cost and time required to produce fault-free software for complex systems using conventional development and testing methods still did not result in software completely free of faults. Fault-tolerant software design techniques have been developed to be used in the design and development of software and hardware required for implementing the techniques. A software fault is defined as a design defect in the software, where the term "design defect" encompasses all deficiencies introduced during the software life cycle.

The software faults which the fault-tolerant software is to detect and recover from may be due to incorrect specification, incorrect algorithm, incorrect logic, coding and other mistakes. Manifestation of the software fault places the system in an erroneous state which may cause the system to fail. A failure occurs whenever the external behavior of a system does not conform to that prescribed by the system specifications (1). Figure 1 represents the relationship between fault-tolerant software events.

Software fault tolerance is the ability of a system to provide uninterruptible operation in the presence of software faults through multiple implementations (i.e., redundancy) of a given functional process.

Background

It is extremely expensive to produce fault-free software. The primary methods used to attempt to produce fault-free software are based upon the principles of structured design and programming followed by extensive testing. One of the problems with this approach is that of determining when all of the software faults have been found and corrected so that the software can be delivered and put into operational use. Many attempts have been made to develop a model which can be applied to the software and results gathered from tests to estimate and measure the number of faults remaining in the program. These attempts have resulted in most models being immediately subjected to criticism since they do not fit a software development effort other than the specific one from which the data was collected to demonstrate the model. Nevertheless, model development efforts continue and are being made a part of formal software quality assurance programs (2,3).

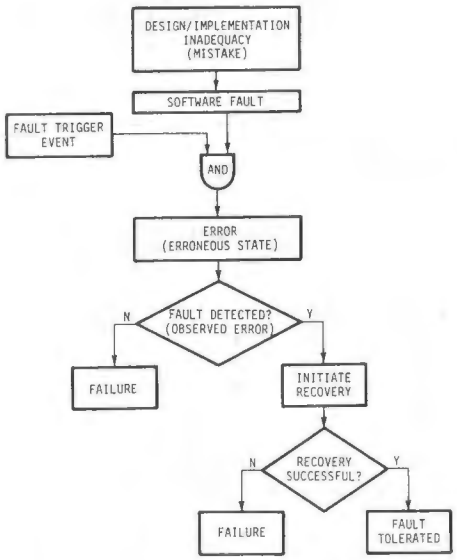


FIGURE 1. FAULT TOLERANT SOFTWARE EVENT RELATIONSHIPS

To date, software faults are still being found in operational digital systems developed using the methods described in the previous paragraph. Consequently, operational phase software maintenance has become a fact of life with significant effort required for locating and correcting residual software faults still present from the development phase as contrasted with the effort devoted to the addition of new functions. In the process of removing the residual software faults, additional faults are often introduced which cause a repeat of the fault location and correction steps in a seemingly endless process. If it were truly possible to produce fault-free software, the operational phase effort could be completely devoted to the addition of software implemented functions required by changes in the operational mission requirements. Unfortunately, this is not the case today.

*Projects Manager, **Research Scientist, Member of the IEEE

Statement of the Problem/Requirements

Software faults exist and are manifested as errors. The effort required for locating and correcting these errors has resulted in high software life cycle costs. Software life cycle costs must be controlled and even reduced and reliability increased.

Software must not cause loss of life, vehicle, or systems functions in real-time digital systems. Software must provide the functions needed to safely accomplish the mission at all times. In order to meet this requirement while minimizing software life cycle costs, new approaches to software development and new tools are required. Fault-tolerant software is one such approach whose time has come.

Sources of Software Faults

The success of any fault-tolerant software technique is critically dependent upon the effectiveness of the employed error detection techniques (1). If an erroneous state is not detected, the fault causing this error will not be tolerated. An understanding of the underlying source of faults can be an aid in designing and analyzing error detection techniques.

A software fault is a design defect which is directly or indirectly caused by a human mistake and can occur in every phase of the software life-cycle. Faults in a system specification, e.g., imprecisely stated or missing functions, are due to a human's inability to correctly translate a problem into a description of a system-level solution. These faults are propagated and new faults are introduced into the subsequent software specification, software design, source code, and object code. With the increased usage of software tools, the number of faults that are directly introduced by the software engineer can potentially be decreased. However, the tool itself can insert an indirect fault (referred to as indirect because a human originally programmed the tool). Past occurrences of this have been mainly restricted to compiler bugs. Indirect faults may increase with the use of automated tools and, like compiler bugs, may be extremely difficult to detect and isolate.

Methods of Handling Faults

Methods for handling software faults can be separated into fault prevention and fault-tolerant techniques (4,1). Fault prevention methods include design methodologies for dealing with the complexity of hardware and software, the use of highly reliable components, and comprehensive testing. Usually, fault prevention methods are utilized prior to system deployment. Fault-tolerant software techniques attempt to automatically overcome software faults without external intervention. The primary usefulness of fault-tolerant software techniques is to handle faults that remain after the system is in operation, but it is also useful in detecting system specification and software faults during development.

The process or steps that a fault-tolerant system goes through have been described in different terms (5,4,1). We will follow the terminology of Reference 1 for purposes of illustration.

The four steps of a fault-tolerant system are: Error detection, damage assessment, recovery, and fault treatment. The capability to detect errors is the crux of any fault-tolerant technique. Since it is doubtful that any error detection technique will occur concurrently with the encountering of a software fault, an assessment of the extent of damage (i.e., changes in state information) must take place prior to attempting error recovery. Often, damage assessment assumptions are a part of the design of the fault-tolerant software system. Error recovery returns the system from a erroneous to a correct state. Finally, fault treatment attempts to eradicate the source of all this trouble, the fault itself, and return the system to service.

Fault-Tolerant Software Techniques

Each of the ten fault-tolerant software techniques identified from the literature have been placed in the four categories shown in Table 1. Within the four category headings, multi-version software, recovery blocks, and exception handlers are themselves fault-tolerant software techniques. The following brief description of each technique should provide some reasoning for the selected categorization.

Table 1. Categorization of Fault-Tolerant Software Techniques

(1) Multi-Version Software

- (2) CRAFTS (Foodtaster)

(3) Recovery Blocks

- (4) Deadline Mechanism
- (5) Dissimilar Backup Software

Hybrid Multi-Version Software and Recovery Blocks Techniques

- (6) Tandem
- (7) Consensus Recovery Blocks

(8) Exception Handlers

- (9) Robust Data Structures
- (10) Hardened Kernel

Multi-version software is an updated term for N-Version programming (6). Multi-version software is defined as the execution of two or more alternate versions of a software component and the selection of a final result or a fault by a decision algorithm. For example, if the decision algorithm is a simple majority vote, three software versions are necessary to mask a software fault. Figure 2 is a graphic depiction of the above description. The goal in developing the

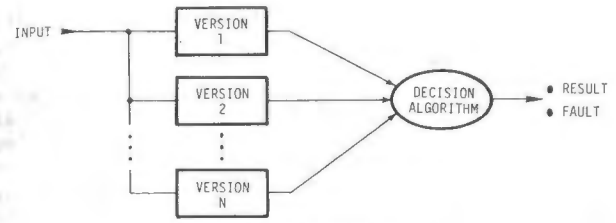


FIGURE 2. MULTI-VERSION SOFTWARE

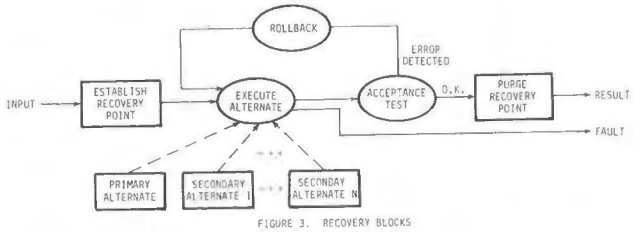


FIGURE 3. RECOVERY BLOCKS

alternate versions is software faults that may exist in one version not be contained in the other version(s). The two reported methods of developing alternate versions are by independent programming teams (6) and by forcing explicit differences in the versions (7,8).

The CRAFTS (Cranfield Algorithm for Fault-Tolerant Software), or Foodtaster as it is sometimes referred to, was originated by Morris and Shepard (9). CRAFTS is a form of multi-version software which is fault-tolerant with just two alternate versions of the software. The two alternates execute, and if the results do not agree, each alternate is compared against an extrapolated value obtained from the last three results. Whichever alternate has the least error between the extrapolated and the current result, within some predetermined maximum deviation, is used. Obviously, this technique is restricted to use in applications of continuous output functions.

Recovery blocks (10) can be regarded as analogous to hardware fault-tolerant 'stand-by sparing'. As the software component produces a result, checks are made on the acceptability of the result. Should one of these checks detect an anomaly, a spare alternate software component is switched in to replace the faulty component. Prior to executing the alternate, the system state must be returned from an erroneous state to an error-free state. This is accomplished by backward recovery restoring a previously accepted error-free state. This process continues until a result is accepted or no more alternates exist, i.e., the software component failed. The alternate software component is produced as described earlier. Figure 3 presents a graphical interpretation of the recovery blocks' operation.

The deadline mechanism (11) adopts the same structure as the recovery block but it performs a specific type of acceptance check. The deadline mechanism relys on violation of timing specifications to detect software errors.

Dissimilar backup software has the same basic structure as a recovery block system but with only one alternate and a fairly severe backward recovery scheme. When an error is detected by the embedded acceptance tests, a minimum amount of current data are available for use by the backup alternate. Therefore, the subsequent backward recovery is not as severe as a 'cold' restart, yet is not as incremental as that proposed for the recovery block.

The different means by which multi-version software and recovery blocks accomplish the process or steps of a fault-tolerant system have allowed researchers to combine their apparent strengths and avoid apparent weaknesses to create hybrid techniques. Two such techniques are tandem (12) and the consensus recovery block (13).

Tandem uses the comparison of two alternate versions to provide error detection similar to multi-version software. This was chosen because of the doubts surrounding verifying the error coverage provided by the acceptance test of recovery blocks (14,15). Based on the observation that errors will occur rarely, tandem opts for just two versions to be executed each cycle to minimize resource consumption (time or hardware). Therefore, masking--as in multi-version software--is not feasible. The recovery scheme chosen is backward recovery because of it's comprehensiveness in purging erroneous data. After rolling back to an error-free state, continued service is attempted through the selection of a different pair of alternates. Figure 4 presents the functional flow of tandem.

The consensus recovery block operates similiar to multi-version software that has a voter for a decision algorithm (refer to Figure 2). The difference occurs when a majority of alternate versions do not agree. This would be a failure state in the case of multi-version software. However, in a consensus recovery block the individual alternate's results are presented to an acceptance in a predetermined order until a result is accepted or all results are rejected. Figure 5 presents a diagram of consensus recovery block operation.

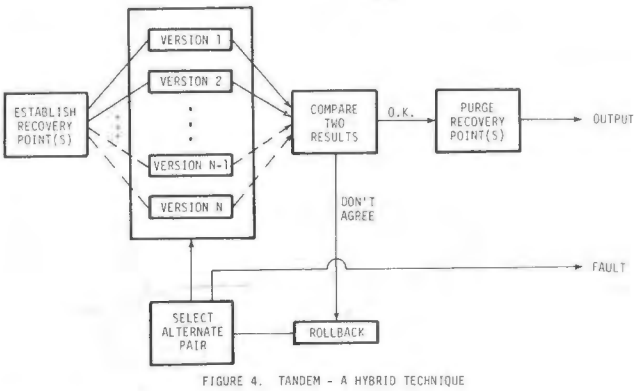


FIGURE 4. TANDEM - A HYBRID TECHNIQUE

The exception handler is a category of fault-tolerant software that uses forward error recovery. A software component's execution is monitored for errors by software and/or hardware checks. These errors are explicitly monitored for, i.e., they are anticipated. Upon detecting an error , the exception handler uses forward recovery to compensate for the erroneous state information so that a result is available to the calling routine and operation can continue. The classic example is that of a divide-by-zero error. The exception handler can set the result to the

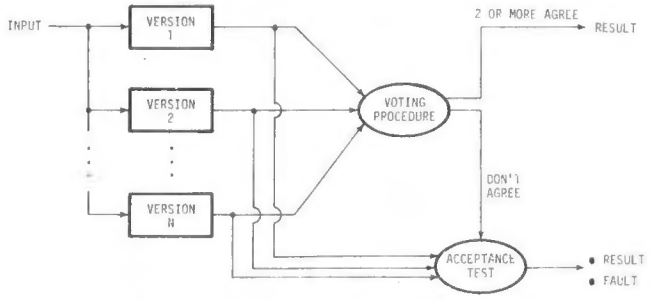


FIGURE 5. CONSENSUS RECOVERY BLOCKS - A HYBRID TECHNIQUE

largest number representable by the system. Figure 6 is a simplistic flow of an exception handler.

Robust data structures build onto existing data structures, redundant structural information to allow for error detection and recovery (16). Error detection checks can either be concurrent with access to the data structure or periodic by execution of an audit routine. Robust data structures use forward recovery to restore the erroneous structural information from the existing redundant information.

Hardened kernel (17) is a term used to describe a software organization which distinguishes between essential and non-essential software components. The essential software components are logically placed together in a kernel. All non-essential software components can be viewed as being attached to the kernel. The non-essential software components are monitored for errors. If an error is detected, the non-essential software component is aborted and is not scheduled for future use. In this manner, the entire software system can continue operation, albeit, in a degraded capacity.

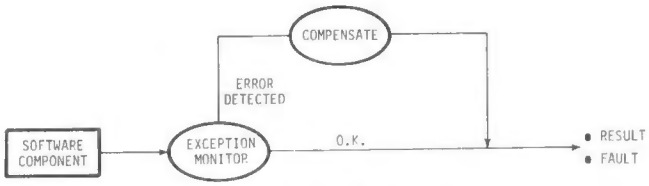


FIGURE 6. EXCEPTION HANDLERS

Fault-Tolerant Software Issues

Fault-tolerant software acceptance has been slowed due to the various issues which have been developed by those playing the role of the devil's advocate. Some of these issues can be attributed to assumptions made by the proponents of fault-tolerant software. Other issues have been in the form of questions which reflect the lack of a data base for fault-tolerant software. Each of these issues is discussed in the following paragraphs.

Assumptions

Applications. Fault-tolerant software has been primarily proposed for applications which require the system to provide continuous correct software operation in the presence of software faults. In the case of real-time digital control systems such as a flight control system, this requires that any fault-tolerant software method not impact the time-critical response of the closed loop system. This requirement has, in the past, eliminated consideration of some fault-tolerant software techniques which utilize backward recovery such as recovery blocks (18). The application dictates the performance requirements of the fault-tolerant software and must be thoroughly understood and the requirements carefully and thoroughly established prior to selection of a specific technique. In the past, the selection of a fault-tolerant technique may have been based upon incomplete requirements which resulted in the wrong technique being selected and hence a "black eye" for fault-tolerant software. Not all techniques are applicable to all problems.

Reliability. Reliability advantages over fault-intolerant software are assumed by many potential users to be the primary reason for using fault-tolerant software. Unfortunately, many of these potential users ignore the fact that fault-tolerant software should provide continuous service in the presence of software faults whereas the fault-intolerant software would fail to provide continuous service in the presence of software faults. As a result, attempts have been made to mathematically prove that fault-tolerant software is more reliable than fault-intolerant software. Few well-conceived and controlled experiments have been conducted to acquire usable data to validate proposed models or identify deficiencies of these models (19, 20). The majority of models for software reliability were developed for existing fault-intolerant software. These reliability models of fault-intolerant software are limited to attempting to estimate the reliability of single modules when applied to estimating the reliability of fault-tolerant software.

Recognizing the limitations of trying to apply fault-intolerant software reliability models to fault-tolerant software, researchers have begun to attempt to develop reliability models for fault-tolerant software. No single model has been developed that fits all fault-tolerant software techniques. Most of the models described in the open literature have been developed for a specific technique such as N-version programming, the recovery block, or hybrid techniques. Since many of these models have not been fully validated, the reliability of fault-tolerant software still remains an open issue.

Cost. Most critics of fault-tolerant software believe that it costs more than software developed using traditional fault-intolerant software. When life cycle cost is considered (not just development cost), fault-tolerant software has been shown to promise significant cost savings (21). Until experimental data is available to validate the results of models, the cost of fault-tolerant software will remain an issue.

Questions/Answers

Battelle recently completed a study of fault-tolerant software (22) which not only describes the various fault-tolerant software techniques, but also synopsizes the recent reliability modeling efforts. This study answered some of the questions about fault-tolerant software and

pointed out the need for further research to develop answers to some of the questions. Important findings of this report are contained in the remainder of this paper.

Does fault-tolerant software really work? That is, does fault-tolerant software provide error detection, damage assessment, recovery, and fault treatment for software faults? In the few real-time applications found (22), the evidence is clear that fault-tolerant software techniques do work.

Is fault-tolerant software more reliable than fault-intolerant software? Models to estimate the reliability of each fault-tolerant software technique are immature and do not permit any conclusion other than fault-tolerant software, properly developed, is more reliable than fault-intolerant software (22).

What constitutes properly developed fault-tolerant software? Research indicates that the majority of faults stem from software requirements definition mistakes. Consequently, formal tools are being developed and used to develop the software requirements and specifications (23,24,25,19). The development activities which follow are somewhat dependent upon the fault-tolerant software technique being implemented. Specific differences in test and integration methods for multiversion versus similar redundancy exist (18). These differences will continue to exist just as differences exist in the process of developing fault-intolerant software due to differences in software development tools, company procedures, and individuals' training and education.

Is one fault-tolerant software technique better than another? A blanket statement that one technique is better than another would be incorrect. There are applications where one technique may be better than another based upon real-time control performance considerations (18). The selection of a technique is dependent upon many considerations of which reliability is one. A mixture of techniques may be used in a total system such as an avionics system. For example, N-version programming might be used for the flight control function and recovery blocks might be used for the navigation function. The interaction of two such functions, though implemented using different fault-tolerant software techniques, is easily handled by the data exchange controlled by the executive software.

Research Needed for Fault-Tolerant Software

The tools available for developing fault-tolerant software are either the same software development tools used for fault-intolerant software or derivatives of these tools. While these traditional software development tools are constantly having new features added, such as the capability to generate well-structured applications source code, other programs support environment tools which would be useful for developing fault-tolerant software still require development. These include automatic data collection for software errors, data analysis and error classification tools, and associated database management tools.

The above set of tools do not address the primary source of faults attributed to software, the ambiguity and misinterpretation of the specification (19). Research on the development of specification languages and their application to fault-tolerant software is being sponsored by both government agencies and industry. Results to date indicate that none of the existing specification languages available are sufficiently automated, are difficult to understand, and are severely limited in power (19). The need for an automated specification language tool clearly exists, but the development of a good tool is still needed.

Automatic test generation tools are needed. Currently, test case software is either generated using manual techniques or test program generators derived from the traditional fault-intolerant software development tools. While better than manual methods, the tools are far from optimum for fault-tolerant software. In a similar sense, automatic fault insertion programs for fault-tolerant software are needed.

Integration of fault-tolerant software modules is currently a manual decision process. Tools which aid both top-down and bottom-up integration are needed. These tools should interface with the data acquisition tools, and provide the capability for rapid integration and make use of the features of the fault-tolerant software techniques being integrated to reduce the integration mistakes and minimize integration time and cost.

Summary and Conclusions

Fault-tolerant software for digital systems will significantly impact future systems development and use. To date, fault-tolerant software techniques have been employed only when a definite reliability advantage was required over fault intolerant software.

Models to estimate the reliability of each fault-tolerant software technique are immature and do not permit any conclusion other than fault-tolerant software, properly developed, is more reliable than fault-intolerant software. Tools required to support development of fault tolerant software are also immature or non-existent. A modeling and experimental test program should be undertaken to provide the foundation for the development of tools needed to cost-effectively develop and use fault-tolerant software.

Acknowledgment

This paper includes reference to work performed under Contract NAS1-17412.

References

(1) Anderson, T. and Lee, P. A., Fault Tolerance: Principals and Practice, Prentice-Hall Int'l., London, 1981.

(2) Bown, T. P., Post, J. V., Tsai, J., Presson, P. E., and Schmidt, R. L., "Software Quality Measure-ment for Distributed Systems, Distributed Computing Systems: Impact on Software Quality", RADC-TR-83-175, Vol. I & III, Boeing Aerospace Co., July, 1983.

(3) Angus, J. E., Bowan, J. B., and VanDenBerg, S. J., "Reliability Model Demonstration", RADC-TR-83-207, Vol. 1, Hughes Aircraft Co., August, 1983.

(4) Avizienis, A. "Fault Tolerance: The Survival Attribute of Digital Systems", Proc. of the IEEE, Vol. 66, No. 10, October, 1978.

(5) Siewiorek, D. P., "Architecture of Fault-Tolerant Computers", IEEE Computer, Vol. 17, No. 8, August, 1984.

(6) Chen, L. and Avizienis, A., "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation", 8th FTCS, June 21-23, 1978.

(7) Personal Communication with William Greve, Boeing, Seattle, Washington, July, 1983.

(8) Personal Communication with Jim McWha and Peter O'Toole, Boeing, Seattle, Washington, January, 1984.

(9) Morris, M.A., "An Approach to the Design of Fault Tolerant Software", MS Thesis, Cranfield Institute of Technology, Sept., 1981.

(10) Randell, B., "System Structure for Software Fault Tolerance", IEEE Trans. on Software Engineering, Vol. SE-1, No. 2, June, 1975.

(11) Campbell, R. H., Horton, K. H., and Belford, G. G., "Simulations of a Fault-Tolerant Deadline Mechanism", 9th FTCS, 1979.

(12) Soneriu, M. D., "A Methodology for the Design and Analysis of Fault-Tolerant Operating Systems", PhD Dissertation, Illinois Institute of Technology, Chicago, IL, 1981.

(13) Scott, R. K., Gault, J. W., and McAllister, D. F., "Modeling Fault-Tolerant Software Reliability", Proc. of the Third Symposium on Reliability in Distributed Software and Database Systems, 1983.

(14) Melliar-Smith, P.M., "Development of Software Fault-Tolerance Techniques", NASA-CR-172122, March, 1983.

(15) Hecht, H., "Fault-Tolerant Software", IEEE Trans. on Reliability, Vol. R-28, No. 3, August, 1979.

(16) Black, J. P., Taylor, D. J. and Morgan, D. E., "A Compendium of Robust Data Structures", 11th FTCS, 1981.

(17) Raney, L. H., "The Use of Fault-Tolerant Software for Flight Control Systems", NAECON, 1983.

(18) Martin, D. J., "Dissimilar Software in High Integrity Applications in Flight Control", Software for Avionics, AGARD-CP-330, Sept., 1982.

(19) Kelly, J.P.J., "Specification of Fault-Tolerant Multi-Version Software: Experimental Studies of a Design Diversity Approach", Report No. CSD-820927, UCLA Computer Science Dept., Sept., 1982.

(20) Scott, R. K., "Data Domain Modeling of Fault-Tolerant Software Reliability", Ph.D. Dissertation, N. Carolina State Univ., Raleigh, N.C., 1983.

(21) Migneault, G.E., "The Cost of Software Fault-Tolerance", NASA Tech. Memorandum 86546, NASA Langley Rsch. Ctr., Sept. 1982.

(22) Hitt, E.F., Webb, J. J., and Bridgman, M.S., "Comparative Analysis of Fault-Tolerant Software Design Techniques", Final Report, Contract NAS1-17412, Battelle Memorial Inst., Columbus Lab., Feb. 15, 1984.

(23) Jordan, D. and Hauxwell, B., "The MENTOR Approach to Requirements Specification", AGARD-CP-330, Software for Avionics, NATO, 1982.

(24) Hirschmann, L. and Christensen, N., "The Computer - Aided Specification System Easy", AGARD-CP-330, [ibid].

(25) Guttaag, J. and Horning, J. J., "Formal Specification as a Design Tool", Assn. for Computing Machinery, 1980.

EVALUATION METHODOLOGIES FOR AN ADVANCED INFORMATION PROCESSING SYSTEM

84-2647

Richard S. Schabowsky, Jr.[2]
Eliezer Gai[1]
Bruce K. Walker[3]
Jaynarayan H. Lala[3]
Paul Motyka[1]

The Charles Stark Draper Laboratory, Inc.
Cambridge, Massachusetts 02139

Abstract

The system concept and requirements for an Advanced Information Processing System (AIPS) are briefly described but the emphasis of this paper is on the evaluation methodologies being developed and utilized in the AIPS program. The evaluation tasks include hardware reliability, maintainability and availability (RMA), software reliability, performance and performability. Hardware RMA and software reliability are addressed with Markov modeling techniques. The performance analysis for AIPS is based on queueing theory. Performability is a measure of merit which combines system reliability and performance measures. The probability laws of the performance measures are obtained from the Markov reliability models. Scalar functions of this law such as the mean and variance provide measures of merit in the AIPS performability evaluations.

Introduction

The design, evaluation, implementation, validation and verification of an Advanced Information Processing System (AIPS) [1] , [2] has been undertaken by the C.S. Draper Laboratory via the sponsorship of NASA. The goal of this program is to develop and demonstrate a system architecture and the associated design and evaluation methodologies which can effectively serve the need for advanced information processing and control over a broad range of future NASA missions. The AIPS is designed to provide a fault tolerant and damage tolerant data processing architecture that meets both aeronautical and space vehicle application requirements. The requirements for seven different applications are described in the AIPS System Requirements [3]. The requirements can be divided into two categories: quantitative and qualitative. Examples of the former are processor throughput, memory size, transport lag, mission success probability, and so on. Examples of the latter are graceful degradation, growth and change tolerance, integrability, etc. The AIPS architecture is intended to satisfy the quantitative requirements and also have attributes that make it responsive to the qualitative requirements.

The system is comprised of fault- and damage-tolerant 'building blocks' which include processing elements (i.e., Fault-Tolerant Processors (FTPs) and Fault-Tolerant Multiprocessors (FTMPs)), input/output (IO) networks, an intercomputer (IC) network, a power distribution system, and the system software that ties these elements together in a coherent system. The system is managed by a set of global functions which allocate tasks to individual processing sites, perform system level redundancy management and reconfiguration, maintain knowledge of the system state for distribution to the component elements and provide intercomputer communications. Redundancy management, task scheduling, and other local services at individual processing sites are handled by local operating systems.

The architecture permits application designers to select the appropriate set of these elements and configure a specific processing system for their application. Design and evaluation methodologies to facilitate the architecture selection trade studies are being developed. This paper describes the present status and direction of future developments of the procedures and tools for the evaluation of hardware RMA, software reliability, performance and performability. This paper is a revised version of Sections 2.2 through 2.6 of [4] wherein a more complete set of references may be found.

Hardware Reliability, Maintainability, Availability

The RMA Evaluation Problem for AIPS Architectures

In ultra-reliable computer systems such as FTMP [5] sufficient hardware redundancy is provided to ensure that the dominant contribution to system unreliability is not the depletion of resources but rather the inability to accommodate (i.e., detect, isolate and reconfigure) all faults in a timely manner, i.e., imperfect fault coverage. Recent efforts to obtain more accurate reliability predictions have led to the development of sophisticated computer-aided reliability programs, such as CARE III [6] and HARP [7] , which expedite the modeling of the fault-handling processes but provide little assistance in reducing the effort required to model the fault-occurrence process. For many systems, the formulation of the fault-occurrence model can represent a considerable effort. In particular, the architectures arising from the AIPS concept are a class of systems where this modeling effort is substantial.

Several features of AIPS architectures make the associated RMA evaluations more complex than previous analyses of ultra-reliable computer systems. First, a building block of an AIPS architecture may itself be an ultra-reliable system, e.g., an FTMP. Since the unreliability driver for this subsystem is the inability to achieve perfect fault coverage, the RMA evaluations should utilize state-of-the-art fault-handling behavior models. Second, an AIPS architecture can be an ultra-reliable system with respect to some system function without the utiliza-

[1] Member AIAA.

[2] Member IEEE.

[3] Member AIAA, IEEE.

tion of an ultra-reliable processing site. This high function reliability is achieved via function migration, i.e., more than one processing site is capable of supporting the computations and IO required for a particular function. In this case the unreliability driver is not necessarily a lack of fault coverage within a processing site but perhaps a depletion of resources allocated to a particular function or unsuccessful function migration. Hence, less sophisticated but conservative fault-handling models for processing elements will often suffice and the analysis effort shifts to accurately modeling function migration and the depletion of resources, i.e., constructing the fault occurrence model.

The formulation of fault occurrence models for systems exploiting function migration is complicated by the introduction of sequence dependent failure modes. Since successful function migration is dependent upon the level of cooperation among the processing sites involved, the availability of the IC network must be considered in evaluating function reliability. Sequence dependency arises because it matters whether the IC network fails before or after function migration is required. Similarly, the sequence of component failures determines whether an (a priori specified) candidate reconfiguration is attainable and hence, whether a particular function migration will be attempted.

A third feature of the AIPS architecture which complicates reliability evaluations is the utilization of nodal networks for inter-computer communication and IO. The reliability evaluation of communications networks is a non-trivial problem which the AIPS Technology Survey [8] identified as an issue to be resolved. In addition, fault accommodation considerations dictate a design wherein the processing sites are not cross-strapped to the IC network. There is also no requirement that the IO network be cross-strapped to processing sites. The absence of cross-strapping among subsystems greatly increases the complexity of the analyses.

In summary, computer-aided reliability evaluation programs are available to facilitate the modeling of fault-handling processes, but corresponding computer-aided engineering tools for reducing the efforts required to construct fault occurrence models are unavailable. This is a shortcoming in the present context because the construction of fault occurrence models for AIPS architectures requires a formidable effort. Succinctly then, the problem to be addressed is the development of computer-aided engineering tools to support the formulation of fault-occurrence models for AIPS architectures and integrate these efforts with previous and on-going efforts in fault-handling behavior modeling.

RMA Evaluation Methodologies for AIPS Architectures

The advantages and flexibility of Markov models favor their selection as the basis for an RMA evaluation methodology. A major problem with Markov modeling, however, concerns the potential for a large number of states which may arise during the model formulation and yield an intractable model. This is, in fact, the situation when a single Markov model is constructed for an entire AIPS architecture. Several techniques exist which may be utilized to address the state proliferation problem. These include behavioral and structural decomposition, model truncation, and various state aggregation techniques.

Behavioral decomposition is a temporal decomposition of the system model into fault-handling and fault-occurrence models. This decomposition is based on the observation that the time constants of the fault-occurrence process are several orders of magnitude larger than those of the fault-handling process. Behavioral decomposition was utilized in the development of CARE III and HARP. These efforts will be exploited in a manner indicated below.

The 'building block' character of the AIPS concept naturally suggests the exploitation of structural decomposition in the analysis of AIPS architectures. Structural decomposition consists of dividing a system into smaller independent subsystems, analyzing the subsystems via independent Markov models and then combining the subsystem results combinatorially to obtain results for the system. This requires that subsystem failures be mutually exclusive, i.e., failures of components in one subsystem do not imply changes in the state transitions of Markov models for other subsystems. This requirement is satisfied in AIPS architectures if the subsystems are the basic AIPS 'building blocks,' i.e., FTPs, FTMPs and nodal networks. An advantage of this decomposition is the opportunity to analyze the general purpose computers (i.e., FTPs and FTMPs) with existing computer-aided reliability programs such as CARE III. This is possible because the formulation of a fault-occurrence model for an FTP or FTMP involves a reasonable level of effort whereas a prohibitive effort would be required to construct a fault-occurrence model for an entire AIPS architecture. Thus the fault-handling modeling efforts embodied in CARE III and/or HARP can be exploited.

A difficulty is encountered, however, in the application of structural decomposition to AIPS architectures when each 'building block' is associated with an independent Markov model. This difficulty arises in the formulation of the combinatorial equation which combines the subsystem results into a system result. The combinatorial equation formulation is very complicated due to the presence of sequence dependent failure modes introduced by function migration and the inherent lack of cross-strapping among AIPS 'building blocks.' To mitigate this difficulty, guidelines for a structural decomposition which balances the complexity of the Markov models and combinatorial equation will be developed. In addition, the Markov model and combinatorial equation formulations will be supported by computer-aided engineering tools.

To address the reliability evaluation of the circuit-switched nodal networks present in the AIPS architectures, a Markov model-based solution which determines the network unreliability while accounting for all network elements has been proposed. The approach is illustrated in Figure 1 for a two-fault tolerant system. In this diagram each circle represents a unique state and each column of states is associated with a failure level, labeled iF, where i is the minimum number of failures required to reach a state at that failure level. The arrowed lines represent possible transitions from each state. Intra-failure level transitions and transitions from higher to lower failure levels have not been shown. Transitions with the Ei label always utilize exact transition probabilities while transitions labeled Ai may be approximate for reasons

explained below. The same label on different transition paths does not imply that these transitions have equal probabilities.

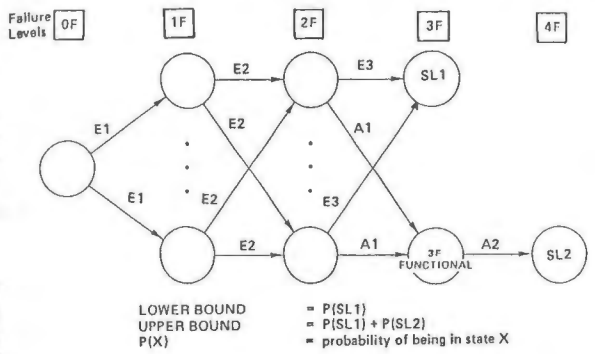


Figure 1. Markov Model for Evaluation of Two-Fault Tolerant Network

State aggregation and model truncation are exploited to substantially reduce the number of states required in the model formulation. State aggregation is the technique of aggregating dissimilar system configurations [4] in a common state without compromising the integrity of the analysis. Two state aggregation techniques are implicit in Figure 1. An obvious aggregation occurs at the three-failure level where all functional configurations are aggregated into a single state; it is not necessary to delineate the specific functional configurations arising at the three-failure level which represents a considerable savings in analysis. State aggregation is also performed at the two-failure level by aggregating into a common state all two-failure configurations which have the same transition probability to the system failure (loss) state, SL1. In performing this aggregation approximations may be introduced along the paths labeled A1. These approximations arise because the configurations which are aggregated on the basis of system failure transitions do not necessarily have an equal complement of hardware. Notice that this aggregation technique is a variation of the rule [9] to the effect that states which have the same exit transition probabilities can be aggregated without affecting the integrity of the analysis.

Model truncation refers to the practice of truncating a Markov model formulation at a failure level below the maximum failure level for which functional/operational configurations exist. Model truncation is implemented by defining any configuration associated with, say, the Nth failure-level to be a system failure state. This failure level truncation is motivated by the observation that only those failure sequences which significantly affect the unreliability estimate need to be explicitly modeled; the probability of N failures decreases rapidly as N increases. Notice that model truncation is a conservative approximation because functional configurations arising from N or more failures are counted as system failures. The conservative nature of model truncation is also dependent on the selection of the probabilities associated with the transitions labeled A1 and A2 in Figure 1. Specifically, care must be taken to select conservative approximations for these transition probabilities in order to guarantee conservative unreliability estimates for any failure rates input to the model. This is particularly important when sensitivity analyses are to be performed. Guidelines for the selection of these approximate transitions can be found in [10].

In order to assess the impact of the conservative approximations introduced by model truncation and the state aggregation techniques, lower and upper unreliability bounds can be derived. Referring to Figure 1, for example, a lower unreliability bound is obtained by examining the probability of being in state SL1. This system failure contribution establishes a lower bound because all failure sequences leading to system loss have not been counted (i.e., system losses due to four or more failures were neglected) and no approximations have been introduced to the transition probabilities leading to this state. An upper bound for the system's unreliability is obtained by summing the probabilities associated with states SL1 and SL2. This establishes an upper bound because 1) there exist system configurations with four or more failures which are not system losses but which are counted as system losses, and 2) only exact or conservative transition probabilities are used for the Ai transitions.

This methodology was applied to a 22-node, 38-link IO network designed for GNC functions of a manned space vehicle [11] . The first unreliability contributions occurred at the three-failure level as in Figure 1, which illustrates the model formulation for this example. The results indicate that the unreliability bounds were tight for shorter mission times (< 10 hours) but separated for longer mission times as the probability of four or more failures significantly increased. To tighten the bounds for longer mission times, the model would have to be truncated at a higher failure level, i.e., at the fifth-failure level. This would require considerable manual effort since the effort involved in carrying out the model formulation to the fourth-failure level was already significant. Fortunately, the basis of the methodology (i.e., identifying and tracking the occurrence of the dominant minimal cut sets) is amenable to automation. A computer-aided engineering tool to support network reliability evaluation is presently being developed. A secondary effort in this area will be the formulation of a state aggregation technique which can be exploited at the two-failure level when the model is truncated at the five-failure level. When the model is truncated at the five-failure level, the state aggregation techniques described above are applicable to the third and fourth failure levels. The observation that there can be many, many configurations at the two-failure level motivates the effort to define an appropriate state aggregation technique.

The preceding discussions were primarily applicable to reliability evaluations. Maintainability and availability studies will build upon the reliability oriented developments. Maintainability can be readily assessed by exercising reliability and availability models with various repair and maintenance scenarios. Methodologies and tools to facilitate availability analyses are, however, less available than their reliability counterparts. For example, neither CARE III nor HARP can support availability analyses. Fortunately, the use of Markov modeling techniques for reliability evaluation facilitates the extension to availability studies [12]. Undoubtedly a number of issues requiring further resolution will arise in cases for which availability is the parameter of primary interest.

Software Reliability

For digital systems with extremely high reliability (of the order of 10^{-7} to 10^{-10} failures per hour), the assumption that the unreliability of the software can be neglected is unwarranted. An analytical method to evaluate the reliability of the system software is therefore necessary. Since the mechanism underlying software failures is not fully understood, it is appropriate to define what is meant here by software reliability.

The reliability of a program (as a function of time t) is the conditional probability that the program yields correct outputs within some prespecified tolerances in the execution interval (0,t) for all possible inputs from the user environment, given that the hardware is in a specified operational state.

Software reliability is thus defined in a manner which is analogous to hardware reliability, where the notion of execution time replaces the notion of operating time. This hypothesis is not universally accepted and others (e.g., [16]) have assumed that software reliability is not a function of execution time. Later it will be shown how the proposed software model can be formulated to accommodate both cases.

Since this definition is probabilistic, an underlying assumption is that the time associated with the occurrence of a software failure is a random variable. The justification for this assumption is that even though a software failure may be attributed to a programming error which escaped detection during the program verification cycle, this error does not result in a software failure each time the code containing the error is executed. Otherwise, the error would have been removed during testing. Such errors persist because it is impractical to test all the possible conditions under which the code can be executed.

The definition of software reliability implies that it is dependent on the operational state of the hardware. Although the error rate is expected to be low and constant in the operational phase of software development [13], it has been suggested [14] that the number of software errors increases as the state of the hardware deteriorates. Hence the system unreliability due to software will be predicted from the results of the software reliability models developed for the operational phase and the hardware reliability model (see Software/Hardware Reliability section).

Relatively few software reliability models have been considered for the operational phase [15], the most promising one being the Markov model [16],[17]. A Markov model takes a microscopic approach to reliability evaluation by breaking a program into discrete modules with known attributes. The reliability of the overall program is then predicted by taking into account the structure of the software.

The Markov model approach assumes that the software has been written in a modular form such that the decomposition into modules is unambiguous and the following two conditions are met:

1. Module failures are independent, i.e., the occurrence of a failure in one module does not cause a failure to occur in another module.
2. In a given processor, module executions are mutually exclusive in time. This rules out the possibility of simultaneous failures in different modules when those modules are executed by the same processor. In a multiprocessing environment simultaneous multiple module failures can occur but the probability of occurrence is small relative to single failures (similar to simultaneous hardware failures).

It is also assumed that each module can be characterized by three attributes:

1. Execution time. In many cases this attribute is not deterministic. In such cases, an expected value will be used. It should be noted that in those cases in which the execution times vary as the result of failure modes, the mean value of the execution time is not significantly affected by these relatively low probability events.
2. Reliability. Module reliability is defined as the probability of failure per unit execution time. This is probably the most difficult attribute to obtain quantitatively, since it depends on such attributes as the length of the module and its complexity as well as the richness of its inputs.
3. Transition probabilities. This attribute represents the exit probabilities from each module and can be obtained from the structure of the program and the reliabilities of the individual modules.

The first step in establishing the Markov model is the definition of the states. If the software is decomposed into p modules, p states will be required to represent the execution of the modules. An additional state will be required to represent the failure state. This state will be the only absorbing state [9] in the model when software reliability is modeled as a function of execution time. Consequently, this model does not have a steady state solution, and the probability of software failure thus monotonically increases with time; however, errorless software can be accounted for with an appropriate choice of initial conditions. To remove the dependence of software reliability upon execution time, an additional absorbing state, which represents the reliability of the software, is included; in this case a steady state reliability prediction is obtained. If, following a module failure, the software is reinitialized q times before a software failure is declared, q additional states will be needed when the restarts begin with the first module of the program. If the reinitializations begin with the module in which the error is detected, pxq additional states will be required.

The second step is to define the single-step state transition probabilities among the n states. In general the transitions among the first p states can be determined directly from the data flow in the program and the individual module reliabilities. When software reliability is a function of execution time, there is a transfer from the pth state to State 1 to represent the cyclic execution of the code. For software reliability evaluations which are independent of execution time there is a transition from the pth state to the reliability absorbing state which represents successful termination of the program

with correct program output. There is also a transfer from each of the first p states to the first reinitialization state(s) in order to model the failure and recovery process. The transition probabilities representing successful and unsuccessful reinitializations are also required.

The propagation of the single-step state transition matrix is performed via the basic Markov process relations [9]. However, unlike Markov reliability models for hardware, each transition in the above model does not represent the same time interval. Each block has a different (possibly not deterministic) execution time. The transformation from M transitions to time can be done using averaging techniques. This is justified due to the large number of transitions (approximately 10^7) in observation intervals of reasonable size. An application of this method to software which was developed for a fault-tolerant process controller is given in [18].

Since the Markov model merely reflects the structure of the software, its construction is straightforward albeit potentially tedious. The more difficult problem is to derive the failure rate of each module. This must be done during the design and validation phases of the software development by collecting and processing error rate data. Unfortunately, the input data to most models is generated long after these phases have been completed [14], thus reducing confidence in the model predictions because of the incomplete input information. In order to avoid such problems in the AIPS program, error data collection will begin as soon as any module is available. Procedures will be established to fulfill the data collection function in such a way that compliance will be assured. First, the configuration management tools in the Ada Programming Support Environment will be used for configuration control, and AIPS project management will emphatically insist that all software changes be made through this system. Secondly, data collection regarding software errors will be managed in an automated fashion, and a cross-reference system between reported errors and software changes will be maintained to ensure that all errors are reported and that all detected errors are corrected.

Important procedures to be emphasized during the data collection phase are:

- Errors that were discovered by inspection as well as those discovered via simulation will be included.
- Both execution and calendar time will be recorded whenever an error is discovered.
- The effects of each error on the overall system will be verified.

Software/Hardware Reliability

Methodologies based on Markov modeling were proposed for evaluating hardware reliability and the reliability of software during the operational phase. This section addresses the combined software/hardware reliability evaluation problem.

Let

$P_{FH}(t)$ = probability of system failure due to the hardware before or at time t
$P_{FS}(t)$ = probability of system failure due to the software before or at time t
$P_{FU}(t)$ = probability of system failure due to failures which cannot be unequivocally assigned to either hardware or software before or at time t

The system is working (possibly in a degraded mode) if none of the above failures have occurred [19],[20]. Let $R(t)$ represent the reliability of the system at time t. Then

$$R(t) = [1 - P_{FH}(t)]\,[1 - P_{FS}(t)]\,[1 - P_{FU}(t)]$$

Since it was assumed in the previous section that the software reliability depends on the hardware status, $P_{FS}(t)$ will be defined as

$$\begin{aligned} P_{FS} = {} & \text{Prob(Software failure/No hardware failures)} \\ & \times \text{Prob(No hardware failures)} \\ & + \text{Prob(Software failure/Degraded hardware)} \\ & \times \text{Prob(Degraded hardware)} \end{aligned}$$

Performance Evaluation

In order to evaluate the performance of a particular AIPS design,it is necessary to select the quantitative indices which will be used to describe the performance. Since the AIPS is intended to be a multipurpose real-time information processing architecture for a variety of aerospace and space vehicles, many of the quantitative performance indices which apply to mainframe computers and time-sharing systems will not be applicable. For example, many of the AIPS application requirements [3] include descriptions of information processing workloads which are relatively static and merely repetitive in nature. Under these circumstances, the system throughput, which is frequently used as a performance measure of a time-sharing system, is not a good measure of AIPS performance because as long as the throughput exceeds the workload the system specification for throughput will be satisfied although other aspects of the system performance (such as relative utilizations of devices or the delays associated with the processing tasks) may be undesirable. Instead, the performance evaluation studies conducted so far have emphasized the average time delays in processing the various tasks and the utilizations (percentage of time busy) of the processing sites and data communication elements as indicators of AIPS performance. The use of time delays as a performance measure is justified by analysis of real-time control computer performance in [21]. The following paragraphs briefly describe the methodology which has been developed to generate numerical performance results for AIPS architectures. It is assumed in these analyses that all the system elements are working at all times. The next section will address the issue of performance when faults have degraded the system capabilities.

There are two fundamentally different philosophies for scheduling the tasks that make up the AIPS workload. One is the quasistatic task assignment philosophy wherein for a given mission phase each task is assigned permanently (except for contingencies induced by faults or emergencies) to a particular processing site. The other is the dynamic task allocation strategy which involves assigning each task, as it becomes necessary to execute it, to the first available processor subject to a priority structure. These task scheduling strategies and the workload they engender for a particular applicaton have been described in [22] through [24].

The system operation proceeds approximately as follows. Each task in the workload must be executed at a given rate. At its scheduled time for execution, the task enters the queue of the appropriate priority to await an available processor. In the quasistatic task assignment case, it simply enters the queue for the processing site to which it is assigned. Once a processor (or the assigned processor) becomes available, the task begins execution. The first step in its execution is a request by the executing processor to the shared memory for the program code which is necessary to execute the task if this code does not reside in the processor's cache memory. This request to shared memory joins a queue with the requests from all of the other processing sites for the use of the data communications network connecting the processors to the shared memory. Once the program code has been transmitted, the processor will generally initiate a second request to shared memory for the data necessary to execute the task. Upon the satisfaction of this request, the processor executes the task and then, in general, makes a third request to shared memory, this time to write the output data for use by other tasks. The delay which occurs in executing each task then consists of a scheduling delay in obtaining a processor (particularly in the dynamic task allocation case), as many as three delays associated with contention for the shared memory, and the delay in actually executing the program code.

Queueing theory provides the basis for the methodology used to calculate the measures of performance for AIPS architectures. Unfortunately, the variety of execution rates and the variety of sizes of the tasks to be executed by the AIPS makes the direct application of the theory presented in [25] and [26] impossible. As a result, it is necessary to introduce some approximations in order to apply queueing theory to the operation of the AIPS. For the quasistatic task assignment case, it is first assumed that the tasks are prioritized for each processor in descending order of execution rate and that the processing of lower rate tasks is interrupted (without penalty) in order to process higher rate tasks. It is also assumed that no interruption can occur while a processor is awaiting the servicing of a shared memory request. These two assumptions eliminate the need for a processor queue for each processing site because they imply that the processing rate for a particular task is merely reduced by the nonavailability of the processor due to the execution of higher priority tasks and the associated idle time while awaiting service of memory requests. The memory requests are assumed to be serviced on a first-come first-served (FCFS) basis. In order to apply queueing theory to the memory requests, it is necessary that they all arrive with a common average rate. In light of the various task execution rates, this is not true of the AIPS. Therefore, the approximation is made that each request generated by a task with an execution rate lower then the highest such rate actually consists of several requests of proportionately smaller size at the highest execution rate. This highest rate then becomes the common arrival rate for the memory request queue. The variety of request sizes (after the scaling to obtain the common arrival rate) is handled via the use of bulk queueing theory [25]. Here, the requests are assumed to consist of some integer multiple of a standard block size. The blocks are analogous to "customers" in standard queueing theory parlance. Thus, when a request arrives at the shared memory queue, it is represented by the arrival of an integer-valued number of blocks. With these assumptions, it is now possible to apply the formulas for bulk queues given in [25] to calculate average delay times and other related results.

More details on the performance evaluation methodology for the quasistatic task assignment case can be found in [27] . It should be noted that an implicit assumption is made in the application of queueing theory to this problem, namely that the task execution times are random and obey a Poisson distribution with average rate equal to the execution rate for the task. This is generally not true of the operation of the AIPS. However, the average delay results and average utilizations are unaffected by the invalidity of this assumption. The second order statistics of the delay times produced by queueing theory will, however, be affected by this assumption and hence are not calculated.

The dynamic task allocation case presents further difficulties because additional queues must be introduced into the model to reflect the behavior of the task scheduling algorithm. The tasks are assumed to have priorities assigned to them in decreasing order of execution rate and it is assumed that higher rate tasks always interrupt lower rate tasks, similar to the quasistatic case. Here, however, the analysis of the time required for processing becomes quite difficult for all but the highest rate tasks because the possibility exists that tasks may be interrupted on one processor and resumed on another. This means that the analysis involves the investigation of a multiple priority, pre-emptive interrupt and resume queue where the various priority classes all have different average arrival rates. At present, approximate results are being generated for the behavior of the task scheduler in terms of the average delay to servicing of a given task. Investigations will follow on the interaction of the task scheduler with the shared memory queue.

Performability: Reliability/Performance Interaction

In the previous section, system performance was defined in the traditional way, namely, as how well the system is performing given that all of its elements are operating within their specification. This definition is proper for a simplex system, but is incomplete for a fault-tolerant system.

A simplex system operates under a fixed set of rules and ceases performing when any of its components fails. Therefore the performance of simplex systems can be defined in the traditional way and is typically specified and evaluated on the basis of one or several scalar measures of merit.

A fault-tolerant system, on the other hand, is able to adapt to failures of its components in a planned, systematic way. Consequently, depending on the failure status of the system's components, fault-tolerant systems can operate in any one of several modes or **operational states**. Thus a fault-tolerant system usually represents a set of different **equivalent-simplex systems** operating under possibly different sets of rules (algorithms), each with its own performance level. Since the operational state of a fault-tolerant system can change at arbitrary times, the value of a scalar performance measure will change accordingly. Hence, a "complete" measure of fault-tolerant system perform-

ance is a unified performance-reliability measure, which quantifies a system's performability [28].

The approach to be taken in the AIPS program to evaluate the performability of a fault-tolerant system is based on ideas suggested in [29] through [32] and requires that those performance measures be characterized as stochastic processes. The development of the probability laws for these stochastic processes is based upon the following three observations:

1. For a given mission of finite duration there exists a sequence of states, the operational state history, that defines the system operation during the mission.

2. The value of a performance measure at any given time may either depend on the operational state history or depend only on the current operational state. In both cases traditional scalar performance measures exist. This one-to-one correspondence is established by noting that for a given operational state or state history the system can be viewed as an equivalent-simplex system, and consequently the value of the measure for that given operational state or state history is equal to its value for the corresponding equivalent-simplex system.

3. The evolution of the operational states as a function of time can be modeled as a finite-state random process. Consequently, the probability mass function of the operational states is the state probability vector of the system Markov reliability model. Thus a Markov reliability model can be used to determine the probability of occurrence of each operational state and each possible state history [31].

It therefore follows that associating the probability of each operational state or state history with the corresponding value of the scalar performance measure provides a complete probabilistic characterization of the measure as a stochastic process [31]

When the performance measure depends only on the current operational state, knowledge of the state probability vector for all values of time of interest (i.e., the solution of the Markov reliability model) and the mapping of system states to values of the performance measure completely establishes the measure's probability law. When the performance measure depends on the state history, however, the determination of the measure's probability law requires additional computation. In this case the state history probabilities must be calculated and associated with the corresponding value of the performance measure.

In order to establish the probability of experiencing a particular state history the time interval of interest is divided into K subintervals. These subintervals are defined by the rate at which fault detection and isolation (FDI) tests are performed and it is assumed that any change in the system's operational state occurs at the beginning of a subinterval. Thus, there is a total of K times at which a state transition may occur. For a Markov model of N states an upper bound on the number of possible state histories is, therefore, N^K. Even for reasonably small values of N and K, N^K will be large; constructing the probability laws for large K and N would be prohibitively time-consuming. This computation can be substantially reduced by reducing N and/or K.

Among the approximation techniques by which N can be reduced are state aggregation and model truncation which were described earlier in the RMA Evaluation Methodologies section. The value of K can be reduced by introducing the notion of mission phases. Rather than dividing the interval of interest into subintervals corresponding to FDI decision intervals, it can be divided into a smaller number of intervals, $M << K$, with each of these intervals corresponding to a mission phase. Although the operational state can actually change at each FDI time during a mission phase, the number of possible state histories is reduced to a manageable size by assuming that state changes occur only at the beginning of mission phases. Then in evaluating the performance measures for a particular state history, transitions to state Si, which represents the state of the system at the end of phase i, are assumed to occur at the beginning of that phase. Hence the effect of this assumption on the numerical results which are obtained is conservative [29] .

Although the probability law of a scalar performance measure completely defines the performance of a fault-tolerant system with respect to the given scalar measure, evaluating system performance by means of this law can be difficult. Therefore, a scalar function of this law (such as the mean and variance of the measure) is suggested as a measure of merit to be used in the specification and evaluation of the performance of a fault-tolerant system.

Once the performability measure has been defined and its derivation mechanized, it can be used (1) for design tradeoffs with respect to components, namely, choice of components, their quality (performance, reliability), and the level of redundancy, and (2) for design tradeoffs with respect to the redundancy management architecture, namely, the FDI algorithms and the failure thresholds. A more detailed description of the approach can be found in [31] and [32].

For the AIPS program the Markov reliability models will provide the state's probability vector at any given time for any given mission. The traditional performance measures will be derived using queueing theory models. It will be necessary to reduce each of the performance characteristics, (e.g., various task delays and processor and bus utilizations) into a corresponding single scalar measure for each operational state. Then the performability can be computed using the numerical considerations discussed in the previous paragraphs.

LIST OF REFERENCES

1. AIPS System Specification, CSDL Report C-5709, C.S. Draper Laboratory, Cambridge, MA, May 1984.

2. J. Lala, "Advanced Information Processing System", Proc. 6th AIAA/IEEE Digital Avionics Syst. Conf., December 1984.

3. AIPS System Requirements, CSDL Report AIPS-83-50, C.S. Draper Laboratory, Cambridge, MA, August 1983.

4. E. Gai et al., AIPS Methodology Report, CSDL Report CSDL-C-5699, C.S. Draper Laboratory, Cambridge, MA, May 1984.

5. A.L. Hopkins, Jr., T.B. Smith, III, J. Lala, "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft", Proc. IEEE, Vol. 66, October 1978.

6. J.J. Stiffler, L.A. Bryant, and L. Guiccione, CARE III Final Report, Phase I, NASA Contractor Report 159122, November 1979.

7. R.M. Geist, K.S. Trivedi, J.B. Dugan and M.K. Smotherman, "Design of the Hybrid Automated Reliability Predictor", Proc. 5th IEEE/AIAA Digital Avionics Syst. Conf., November 1983.

8. AIPS Technology Survey Report, CSDL Report C-5691, C.S. Draper Laboratory, Cambridge, MA, February 1984.

9. J.G. Kemeny and J.L. Snell, Finite Markov Chains, Springer-Verlag, New York, 1976.

10. W.W. Weinstein, R.S. Schabowsky, Jr., E. Gai, Digital Fly-by-Wire Flight Control System for a Tilt-Rotor Aircraft: Architecture and Reliability Studies, CSDL Report R-1636, C.S. Draper Laboratory, Cambridge, MA, June 1983.

11. G. Schwartz and A. Green, "Application of AIPS Architecture to Manned Space Vehicle", AIPS Memo 84-25, C.S. Draper Laboratory, Cambridge, MA, March 1984.

12. D. Siewiorek and R. Swarz, The Theory and Practice of Reliable System Design, Digital Press, MA, 1982.

13. L.S. Gephart et al., Software Reliability Determination and Prediction, AFFDL Report No. TR-78-77, June 1978.

14. Gary Holden, "Software Reliability and the DMSP Block SD Program Error History", presented at C.S. Draper Laboratory, March 1984.

15. C.V. Ramamoorthy and F.B. Bastani, "Software Reliability - Status and Perspectives", IEEE Trans. on Software Engineering, Vol. SE-8, No. 4, July 1982.

16. R.C. Cheung, "A User-Oriented Software Reliability Model", IEEE Trans. on Software Engineering, Vol. SE 6, No. 2, March 1980.

17. M.L. Shooman, "Structure Models for Software Reliability Prediction", Proc. 2nd Conference on Software Engineering, October 1976.

18. J.B. Dewolf, F.C. Furtek, E. Gai, J.V. Harrison, Reliable Software for Fault-Tolerant Systems, CSDL Report C-5487, C.S. Draper Laboratory, Cambridge, MA, April 1982.

19. W.E. Thompson, "System Hardware and Software Analysis", Proc. of the Annual Reliability and Maintainability Symposium, January 1983.

20. A.L. Goel et al., "Hardware/Software Availability: A Cost Based Tradeoff Study", Proc. of the Annual Reliability and Maintainability Symposium, January 1983.

21. K.G. Shin, C.M. Krishna, and Y.H. Lee, "A Unified Method for Emulating Real-Time Computer Controllers: A Case Study", manuscript provided by authors.

22. J. Kernan, G. Schwartz, A. Green, "Sizing Estimates for Multiprocessor Throughput Study (Long Term Processor Assignment Strategy)", Memo No. AIPS-83-77, C.S. Draper Laboratory, Cambridge, MA, October 1983.

23. J. Kernan, G. Schwartz, A. Green, "Association of Functions with Shared Memory Segments for Multiprocessor Throughput Study", Memo No. AIPS-83-89, C.S. Draper Laboratory, Cambridge, MA, November 28, 1983.

24. J. Kernan, A. Green, G. Schwartz, "Multiprocessor Shared Memory Traffic with Fully Dynamic Task Allocation", Memo No. AIPS-84-02, C.S. Draper Laboratory, Cambridge, MA, January 1, 1984.

25. L. Kleinrock, Queueing Systems, Vol. 1: Theory, Wiley, 1975.

26. D. Ferrari, Computer Systems Performance Evaluation, Prentice Hall, 1978.

27. B. Walker, "The Quasi-Static Task Assignment Case", Memo No. AIPS-84-36, C.S. Draper Laboratory, Cambridge, MA, April 1984.

28. J.F. Meyer, D.G. Furchtgott, and L.T. Wu, "Performability Evaluation of the SIFT Computer", IEEE Trans. on Comp., Vol. C-29, No. 6, June 1980.

29. J.V. Harrison, K.C. Daly, and E. Gai, "Reliability and Accuracy Prediction for a Redundant Strapdown Navigator", Journal of Guidance and Control, Vol. 4, No. 5, September-October 1981.

30. J.F. Meyer, "Closed-Form Solutions of Performability", IEEE Trans. on Comp., Vol. C-31, No. 7, July 1982.

31. E. Gai and M.B. Adams, "Measures of Merit for Fault-Tolerant Systems", AIAA Guidance and Control Conference, Gatlinburg, TN, August 1983.

32. D.G. Furchtgott and J.F. Meyer, "Performability Evaluation of Computing Systems Using Reward Models", to be published.

Acknowledgement

This report was prepared by The Charles Stark Draper Laboratory, Inc. under Contract NAS9-16023, Task Order #84-18, with the Lyndon B. Johnson Space Center of the National Aeronautics & Space Administration. Publication of this report does not constitute approval by the NASA/JSC of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

A VALIDATION METHODOLOGY FOR FAULT-TOLERANT CLOCK SYNCHRONIZATION

84-2648

Sally C. Johnson
Ricky W. Butler

NASA Langley Research Center
Hampton, Virginia 23665

Abstract

A validation method for the synchronization subsystem of a fault-tolerant computer system is presented. The high reliability requirement of flight crucial systems precludes the use of most traditional validation methods. The method presented utilizes formal design proof to uncover design and coding errors and experimentation to validate the assumptions of the design proof. The experimental method is described and illustrated by validating an experimental implementation of the Software Implemented Fault Tolerance (SIFT) clock synchronization algorithm. The design proof of the algorithm defines the maximum skew between any two nonfaulty clocks in the system in terms of theoretical upper bounds on certain system parameters. The quantile to which each parameter must be estimated is determined by a combinatorial analysis of the system reliability. The parameters are measured by direct and indirect means, and upper bounds are estimated. A nonparametric method based on an asymptotic property of the tail of a distribution is used to estimate the upper bound of a critical system parameter. Although the proof process is very costly, it is extremely valuable when validating the crucial synchronization subsystem.

Introduction

Clock synchronization is an essential function in fault-tolerant multicomputer systems. Most fault-tolerant flight control systems utilize exact-match voting algorithms that depend critically upon the synchronization of the redundant computing elements. In many systems, the entire communication mechanism depends fundamentally on maintaining adequate synchronization between the replicated system clocks. Therefore, any validation effort must carefully analyze the synchronization subsystem of a fault-tolerant computer system.

This paper investigates the problem of validating the fault-tolerant clock synchronization algorithm used in the SIFT system, an experimental fault-tolerant computer system designed for flight-crucial applications. Weaknesses of classical validation methods are discussed, and a new method of validation relying on formal design proof and experimental testing is introduced.

The Validation Method

Due to the criticality of synchronization systems, credible methods of validation must be developed for these systems. However, severe requirements for flight-crucial systems, such as a probability of failure not to exceed 10^{-9} for a ten-hour flight, preclude the use of classical life testing as an assessment method. Typical alternatives to life testing, such as combinatorial analyses or Markov models, are inadequate because they assume failure independency. Although the clocks are physically separated, clock failures are not independent because each clock uses values from the other clocks in the system to remain synchronized with them. Because of this failure dependency, a fault-tolerant clock synchronization algorithm is used to prevent the propagation of a clock failure to another clock in the system. The validation process must establish the correctness of this algorithm in a system context. One must be assured that a single faulty clock cannot compromise the system reliability. Thus, the following failure modes must be considered:

1. A majority of clocks fail before time T.
2. An error exists in the system design.
3. An error occurred in coding the synchronization algorithm.
4. Even though none of the above have occurred, the design assumptions have been exceeded.

Combinatorial calculations can help only in estimating the probability of failure mode 1 above. To avoid the possibility of failure mode 2, SRI International developed a new algorithm and a mathematical proof characterizing the performance of this algorithm in terms of certain system parameters. If a mechanical verification of the proof were performed (i.e., using formal software verification methods and automatic theorem provers), this failure mode could virtually be eliminated. The use of code verification could also eliminate failure mode 3. However, the possibility of a mode 4 failure must be considered. Fortunately, the formal proof process encapsulates precisely the design assumptions in the form of a set of axioms. Thus, the following validation method is appropriate:

1. Mathematically prove a theorem which characterizes the guaranteed maximum clock skew in terms of measurable system parameters defined through formal axioms.
2. Mechanically verify that the implementation code correctly implements the algorithm.
3. Experimentally verify the axioms required in the design proof.

Although the SIFT synchronization code has not yet been mechanically checked, a mathematical design proof has been performed on the algorithm. The mechanical proof will be performed by SRI International under NASA contract NAS1-17067 during 1984 and 1985. In the following section this algorithm and its proof will be discussed.

SRI Clock Synchronization Algorithm

To discuss the SRI clock synchronization algorithm properly, a few definitions and some notation must be introduced. The theory in this section was developed by SRI under the SIFT development contract NAS1-15428 (1).

It is convenient to define a clock as a function from real time t to clock time T: $C(t) = T$. Real time will be distinguished from clock time by the use of small letters for the former and capital

letters for the latter. It is sometimes useful to use the inverse clock function $r(T) = C^{-1}(T) = t$. Using this inverse function, the concept of synchronization can be defined:

DEFINITION Two clocks r_p and r_q are synchronized to within δ of each other at time T if

$$|r_p(T) - r_q(T)| < \delta.$$

Next, the notion of a good clock is defined:

DEFINITION A clock r is a good clock during the interval $[T_1,T_2]$ if it is a monotonic, differentiable function on $[T_1,T_2]$ and there exists a ρ such that for all T in $[T_1,T_2]$:

$$|dr/dT(T) - 1| < \rho/2.$$

Thus, the drift rate of a good clock from real time is bounded by $\rho/2$, as illustrated in figure 1.

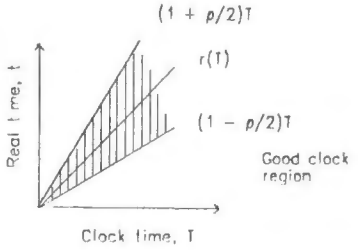


Fig. 1. Definition of a good clock.

A clock synchronization algorithm periodically resets the clocks in the system. As the processors synchronize clocks every R seconds, each processor's time base is a sequence of redefined clock functions. Using $T^{(i)}_{th}$ as the clock time at the beginning of the i^{th} interval,

$$T^{(i)} = T^{(0)} + iR$$

and $R^{(i)}$ = the interval $[T^{(i)},T^{(i+1)}]$.

For each such interval there is a new clock definition:

$$C^{(i+1)}(t) = C^{(i)}(t) + A^{(i)}$$

where $A^{(i)}$ is the i^{th} clock correction.

The clock synchronization algorithm requires that each processor exchange clock values with every other processor during the subinterval

$$S^{(i)} = [T^{(i+1)} - S, T^{(i+1)}],$$

which is the last S seconds of the interval $R^{(i)}$. Since this clock value exchange is subject to error, it is necessary to introduce a notation and an axiom which characterizes this error:

AXIOM If processors p and q are nonfaulty and their clocks are synchronized up to time $T^{(i+1)}$ then p obtains a value Δ_{qp} during the interval $S^{(i)}$ such that

$$|r_p^{(i)}(T_o + \Delta_{qp}) - r_q^{(i)}(T_o)| < \varepsilon$$

for some time T_o in $S^{(i)}$. Thus, the error in reading another processor's clock is bounded by ε.

The SIFT synchronization algorithm is as follows:

ALGORITHM For all p

$$C_p^{(i+1)} = C_p^{(i)} + \Delta_p$$

where

$$\Delta_p = (1/N) \sum_{r=1}^{N} \bar{\Delta}_{rp}$$

if $r \neq p$ and $|\Delta_{rp}| < \Omega$ then $\bar{\Delta}_{rp} = \Delta_{rp}$

else $\bar{\Delta}_{rp} = 0$

where $\Omega = \delta + \varepsilon$.

The following theorem was proved by Leslie Lamport and Michael Melliar-Smith of SRI International:

THEOREM: If

$$\begin{aligned} 3m &< N \\ \delta &\geq [N/(N-3m)](2\varepsilon + \rho[R + 2(N-m)S/N]) \\ \delta &\geq \delta_o + \rho R \\ \delta &<< R \\ \delta &<< \varepsilon/\rho \end{aligned}$$

and if up to time $T^{(i+1)}$ no more than m processes are faulty, then for all p and q:

S1. If up to time $T^{(i+1)}$ processes p and q are nonfaulty, then for all T in $R^{(i)}$:

$$|r_p^{(i)}(T) - r_q^{(i)}(T)| < \delta$$

S2. If process p is nonfaulty up to time $T^{(i+1)}$, then

$$|r_p^{(i+1)}(T) - r_p^{(i)}(T)| < \Omega.$$

where N = number of clocks in the system
m = maximum number of faulty clocks

Assumptions of Design Proof

The design proof effectively establishes the correctness of the algorithm assuming a set of axioms is correct. Many of these axioms are well-established mathematical theorems. Other axioms define the behavior of the computer system on which the algorithm executes. For example, the SRI design proof assumes every processor can read another processor's clock to within an error of ε. The correctness of such assumptions must be established by experimentation. The following is a list of the system behavior axioms which are assumed in the SRI design proof:

1. The maximum drift rate between any two working clocks is $< \rho$.
2. If two processors are nonfaulty then one processor can read another processor's clock to within an error of ε.
3. The clocks of the system are initially synchronized to within δ_o.
4. The system executes the algorithm every R seconds and provides enough CPU time for the algorithm to complete.

Design assumption 3 corresponds to a process that would occur at system initialization. Since this process is performed while the aircraft is on the

ground, it need not be fault-tolerant. Design assumption 4 is strongly dependent on the scheduling method employed in the system, so a generic method canot be presented for validation of this assumption. Hence, only the first two design assumptions will be analyzed in detail.

Each of these assumptions must be established to a confidence level consistent with the reliability requirements. Thus, although life testing of the system as a whole can be avoided, life testing must effectively be performed on certain system parameters. However, the behavior of these low-level components of the system are typically far less complex than the system as a whole and are thus easier to validate. Furthermore, the formal proof provides a precise statement of exactly which properties of the system must be measured.

Validation of an Experimental System

The SIFT system was originally implemented on seven Bendix BDX-930 processors. However, extracting synchronization data from that system is currently difficult. To investigate methods of validation, the SIFT synchronization algorithm was implemented on four VAX-11/750 computers in the AIRLAB research facility at NASA Langley Research Center (2). The validation method was demonstrated by the following approach:

1. Identify the mathematical parameters of the system to be validated.
2. Determine the quantile to which each parameter must be estimated.
3. Estimate the parameters to the required reliability.
4. Determine the guaranteed maximum clock skew for the system.
5. Determine if this maximum clock skew exceeds the value assumed in the system design.

Validation Step 1

The first step in validating the system is to identify the mathematical parameters of the system to be validated. Analysis of the system reveals that all system parameters can be directly measured or calculated except the maximum read error ε and the maximum drift rate ρ. These two system parameters must be chosen conservatively enough to meet the system reliability requirements.

As defined earlier, ε is an upper bound on $|e_{qp}|$ over all processor pairs and over all time, where

$$e_{qp} = r_p^{(i)}(T_0 + \Delta_{qp}) - r_q^{(i)}(T_0).$$

Unfortunately, e_{qp} is defined in terms of real time rather than observable clock time. The following formula defining e_{qp} in terms of observable clock times may be shown to be a highly accurate approximation to the theoretical e_{qp}:

$$A_{qp} + e_{qp} = \Delta_{qp},$$

where $A_{qp}(t)$ is the difference between clocks p and q at real time t (i.e., actual skew at t: $c_p(t) - c_q(t)$).

It is necessary to characterize e_{qp} in a system context. In the system, a processor p reads a processor q's clock by the following method: At a pre-specified time, processor q reads its clock and transmits the value $C_q(t_1)$ to processor p. Upon receiving the message, processor p immediately reads its clock to obtain $C_p(t_2)$. As shown in figure 2, if the exact communication delay X_{qp} were known, then the exact skew could be calculated by $A_{qp} = C_p(t_2) - C_q(t_1) - X_{qp}$.

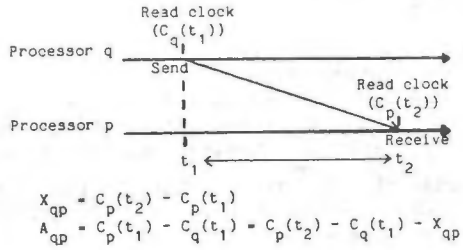


Fig. 2. Calculation of actual skew, A_{qp}.

Thus, the designer of the synchronization system chooses a value v approximately equal to $E(X_{qp})$ to be used by the system to compute an apparent skew Δ_{qp} by the following formula:

$$\Delta_{qp} = C_p(t_2) - C_q(t_1) - v.$$

Because the communication delay is variable, each calculation of Δ_{qp} is subject to an error of $X_{qp} - v$. There are two components to this error:

$$e_{qp} = X_{qp} - v = [X_{qp} - E(X_{qp})] + \mu,$$

where $\mu = E(X_{qp}) - v$. The first component, $X_{qp} - E(X_{qp})$, is the variation due to the random nature of the communication. The second component, μ, is a constant offset due to the system designer's error in choosing v. Also, it follows from the above formula that $E(e_{qp}) = \mu$.

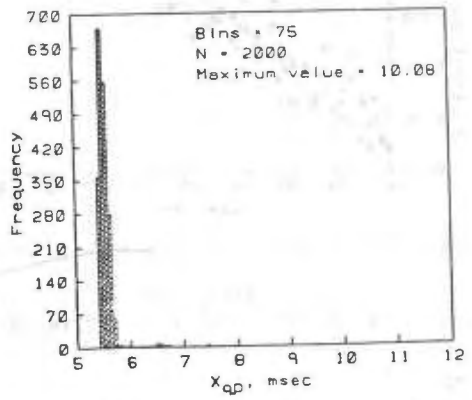


Fig. 3. Histogram of communication delays.

The distribution of e_{qp} may be obtained from measurements of the one-way communication delays. In the AIRLAB VAX system, a special Pulse Network was used to measure this delay. Because the delay for sending a pulse is considerably less variable than the communication delay for sending a message, reasonably accurate measurements of the communication delay were made by the following

method: One processor's clock was read, and the value was sent to a second processor. When the second processor received the message, a pulse was immediately sent over the Pulse Network to the first processor. When the first processor received the pulse, its clock was read again. By subtracting the first clock value from the second clock value, the communication delay plus the pulse delay was measured. Subtracting an estimate of the mean pulse delay from these values provides an accurate measurement of the communication delay. Figure 3 is a histogram of 2000 such estimates of X_{qp}'s.

Next, a second method is discussed that provides a means of estimating both ε and ρ using the internal state information of the synchronization system. The physics of crystal clocks dictates that the drift rate ρ_{qp} between any two clocks q and p is constant over time. Thus, if the system is run without synchronization, then the following model describes the system:

$$\Delta_{qp}(T) = \delta_{qp}(0) + \rho_{qp}T + e_{qp}(T),$$

where $\delta_{qp}(0)$ is the initial skew between clocks q and p at time 0, and ρ_{qp} is the drift rate between clocks q and p. Since the Δ_{qp}'s are computed every R seconds, a set of

$$\Delta_{qp}(T^{(i)}),\ i = 1, n$$

can be collected where $T^{(i)} = T^{(0)} + iR$.

A linear least squares analysis can be used to estimate the parameters $\delta_{qp}(0)$ and ρ_{qp} (see fig. 4).

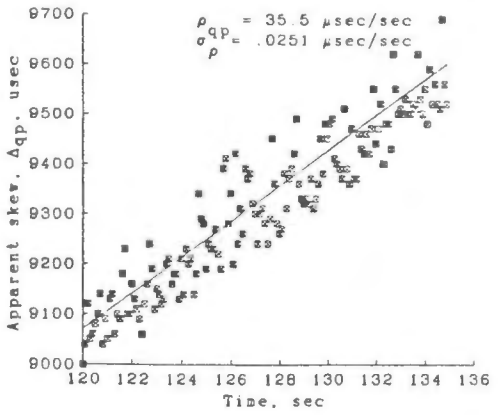


Fig. 4. Least squares fit to measured skews.

Let the residuals from the regression be represented by ϕ, and rewrite the above equation:

$$\Delta_{qp}(T) = \alpha + \beta T + \phi.$$

If the e_{qp}'s are distributed with mean zero, then β is an estimate of ρ_{qp}, and the residuals have approximately the same distribution as the e_{qp}'s. However, this may not be the case. Suppose that $E(e_{qp}) = \mu \neq 0$. Then $\alpha = \delta_{qp}(0) + \mu$, and the residuals have approximately the same distribution as $e_{qp} - \mu$. Thus, a histogram of e_{qp}'s can be obtained by adding μ to the residuals.

Next, a simple independent method to estimate μ will be presented. Since $\Delta_{qp} = A_{qp} + e_{qp}$,

$$\Delta_{qp} + \Delta_{pq} = e_{qp} + e_{pq} + A_{qp} + A_{pq}.$$

Furthermore, since $A_{qp} = -A_{pq}$,

$$\Delta_{qp} + \Delta_{pq} = e_{qp} + e_{pq}.$$

Thus, $E(e_{qp}) = E((\Delta_{qp} + \Delta_{pq})/2)$

since $E(e_{qp}) = E(e_{pq})$.

Therefore, μ can be estimated by the sample average of several observations of $(\Delta_{qp} + \Delta_{pq})/2$.

In a properly "tuned" synchronization system $\mu = 0$ to minimize the absolute value of the read error. The following method may be used to "tune" the synchronization system. As shown previously, the e_{qp}'s consist of two components, the deviation from the mean communication time and the constant offset μ. Since μ can be expressed as follows:

$$\mu = E(X_{qp}) - v = E(X_{qp} - v) = E(e_{qp}),$$

the constant offset can be eliminated by simply adding the above estimate of μ to v. The code in the implementation may thus be corrected to use this new value of v.

Validation Step 2

In this step, the required quantile to which the parameters ε and ρ must be estimated is determined from a reliability analysis of the system. Analysis of the failure modes discussed previously reveals that the system reliability depends on the following three probabilities of failure:

p_h = Prob(one or more hardware component failures in a processor)
p_1 = Prob(two nonfaulty clocks drifting apart faster than ρ)
p_2 = Prob(one or more read errors > ε occurring on a specific processor)

The probability p_h may be obtained from a MIL STD 217D analysis of component failure data. This well-known analysis method will not be discussed here. A processor failure rate of 10^{-5}/hour will be assumed. The probability p_1 is the measurement error in determining the upper bound ρ. This probability may be made arbitrarily small by increasing the bound and/or increasing the accuracy of the measurements (e.g., by using a larger sample size). The probability of failure p_2 arises from the stochastic nature of the communication system used to read another processor's clock. This probability may also be reduced by increasing the bound ε; however, this is done at the expense of increasing the estimated maximum clock skew, $\hat{\delta}$. This trade-off is discussed in detail in the section "Additional Observations".

Since p_1 is the probability of a measurement error rather than an intrinsic failure mode of the system, p_1 can be made arbitrarily small by improving the measurement technique. Therefore, $\hat{\rho}$ will be chosen such that p_1 is small in comparison with p_h, and the required quantile for ε will be determined from a reliability analysis of the

system. Typically, a detailed Markov model is necessary to calculate the reliability of a fault-tolerant system. However, for simplicity, no reconfiguration capability in the system is assumed, and thus a simple combinatorial analysis can be used to compute the reliability of the system. Since the algorithm can tolerate m processor failures, the probability of m + 1 processor failures during the flight must be determined. The system is thus a 2-out-of-4 system.

If the probability of processor failure is

$$p = p_h + p_1 + p_2,$$

then the probability of a system failure during a T-hour mission, p_{sys}, is thus given as:

$$\begin{aligned} p_{sys} &= 1 - \text{Prob(0 failures)} - \text{Prob(1 failure)} \\ &= 1 - \binom{4}{0}p^0(1-p)^4 - \binom{4}{0}p^1(1-p)^3 \\ &\approx 6p^2 \quad \text{since p is small.} \end{aligned}$$

Given a reliability requirement of $p_{sys} = 10^{-9}$, a processor failure rate of $p_h = 10^{-5}$/hour and $p_1 = 10^{-7}$, then

$$p_2 = \sqrt{p_{sys}/6} - p_h - p_1 = 2.809\times10^{-6}.$$

Since the bound ε refers to a single clock read, the number of clock reads during a mission must be determined in order to calculate the quantile needed for ε. Let

$$p_\varepsilon = \text{Prob(obtaining a read error} > \varepsilon \text{ during a single clock read).}$$

The probability p_2 is easily expressed in terms of p_ε:

$$p_2 = 1 - \binom{n}{0} p_\varepsilon^{\,0}(1-p_\varepsilon)^n$$

where n = (N-1)T/R, (i.e., the number of clock reads a specific processor will make during a mission of length T). Using the Poisson approximation to the binomial,

$$p_2 = 1 - \text{Exp}(-np_\varepsilon).$$

Furthermore, by a Taylor series approximation (valid since $np_\varepsilon \ll 1$),

$$p_2 = np_\varepsilon = (N-1)(T/R)p_\varepsilon.$$

Using N = 4, R = 30 sec, and T = 10 hours, the probability p_ε is determined:

$$p_\varepsilon = 7.805\times10^{-10}.$$

This analysis assumes independence of clock read error failures. The design proof has thus reduced the strong assumption of independent clock failure to independent communication. This analysis makes a strong case for avoiding contention-based communication protocols in a fault-tolerant architecture.

Validation Step 3

The third step in validating the system under investigation is to estimate each parameter to the required reliability.

Estimating ρ. It was determined in validation step 2 that ρ must be estimated such that the probability of exceeding design assumption 1 is 10^{-7}. To determine the probability that this design assumption is violated, it is necessary to calculate

$$\text{Prob}(\rho_{qp} > \rho) \text{ for some q and p}$$

where ρ_{qp} represents the drift rate between clocks p and q.

If there are n processors in the system being validated, then there are $n_c = \binom{n}{2}$ drift rates between processor pairs. For simplicity, these will be referred to as ρ_i, i = 1, n_c.

Using the linear regression analysis on the $\Delta_{qp}(T^{(i)})$ data described previously, a set of estimates

$$\{ (\hat{\rho}_i , \hat{\sigma}_i^2) \mid i = 1, n_c \}$$

can be obtained, where $\hat{\rho}_i$ is an estimate of the drift rate between processor pair i and $\hat{\sigma}_i^2$ is an estimate of the variance of $\hat{\rho}_i$.

From the experimental data, an estimate of the upper bound of the drift rates, $\hat{\rho}$, must be determined such that $\text{Prob}(\text{Max}(\rho_i) > \hat{\rho}) = \alpha$ is sufficiently small. The following estimate is easily shown to be adequate:

$$\hat{\rho} = \text{Max}(u_i)$$

where u_i is defined by:

$$\text{Prob}(\rho_i > u_i) = \sqrt[n_c]{1-\alpha}.$$

THEOREM. $\text{Prob}(\text{Max}(\rho_i) < \rho) \geq 1 - \alpha$.

PROOF:

$$\begin{aligned} &\text{Prob}(\text{Max}(\rho_i) < \hat{\rho}) \\ &= \text{Prob}(\text{Max}(\rho_i) < \text{Max}(u_i)) \\ &= \text{Prob}(\rho_1 < \text{Max}(u_i) \wedge \rho_2 < \text{Max}(u_i) \wedge \ldots \wedge \rho_n < \text{Max}(u_i)) \\ &\geq \text{Prob}(\rho_1 < u_1 \wedge \rho_2 < u_2 \wedge \ldots \wedge \rho_n < u_n)) \\ &= \prod_{i=1}^{n_c} \sqrt[n_c]{1-\alpha} \\ &= 1 - \alpha. \end{aligned}$$

ENDPROOF.

The u_i's are easily obtained by use of the following formula:

$$u_i = \hat{\rho}_i + t(\nu,\theta)\hat{\sigma}_i$$

where

- $\nu = n_s - 2$
- n_s = number of data points used in the regression analysis to obtain $\hat{\rho}_i$ and $\hat{\sigma}_i$.
- $\theta = \sqrt[n_c]{1-\alpha}$
- $t(\nu,\theta)$ = θ percentage point of student's t distribution with ν degrees of freedom.

For $n_s > 100$, $t(\nu,\theta)$ may be replaced by a percentage point of the Standard Normal distribution.

A regression analysis of the $\Delta_{qp}(T^{(i)})$ data produced the following table:

Processor pair	i	$\hat{\rho}_i$	$\hat{\sigma}_i$	$\hat{u}_i$
12	1	30.02	0.2687	31.462
13	2	9.01	0.0245	9.143
14	3	35.50	0.0251	35.635
23	4	14.92	0.0954	15.432
24	5	5.48	0.0390	5.686
34	6	40.97	0.0851	41.427

The $\hat{u}_i$'s were calculated using $\hat{u}_i = \hat{\rho}_i + t(\nu,\theta)\hat{\sigma}_i$, where

$$\theta = \sqrt[n_c]{1 - 10^{-7}} = 1 - 1.667\times10^{-8}$$

and $\nu = n_s - 2 = 1998$.

Thus $p_1 = 10^{-7}$, which is small relative to p_h. The maximum drift rate ρ is chosen as the maximum u_i, thus

$$\hat{\rho} = 41.427.$$

Estimating ε. In step 2 it was determined that ε must be estimated as the $1-p_\varepsilon$ quantile of the read error distribution, where $p_\varepsilon = 7.805\times10^{-10}$. Two methods were developed in previous sections to obtain a histogram of the clock read errors e_{qp}. Figure 5 is a histogram of $|e_{qp}| = |X_{qp} - v|$ obtained from direct measurements of the one-way communication times. This data will be used to illustrate the determination of $\hat{\varepsilon}$.

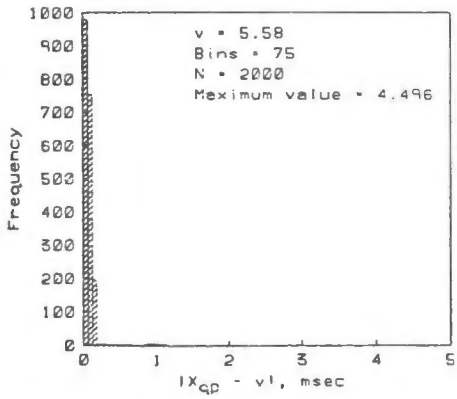


Fig. 5. Histogram of $|e_{qp}| = |X_{qp} - v|$.

Traditional methods of estimating quantiles are inadequate for inference with respect to the tail of a distribution. The traditional nonparametric method would require a sample size on the order of 10^8 observations to estimate this large a quantile. Another traditional technique is to assume an underlying parametric model of the distribution or assume some special properties of the distribution. However, any statistical inference made would be strongly dependent on the assumption that the experimental data was generated from the chosen parametric family of distributions. Clearly, some other method of estimating properties of the tail of a distribution is needed.

Fortunately, a statistical method was developed by Ishay Weissman for the estimation of large quantiles of the underlying population distribution from the k largest observations of a random sample (4). The major assumption of the method is that the underlying distribution function F is in the "domain of attraction" of some known distribution function G. This property is satisfied by a large class of distribution functions including the gamma, Weibull, normal, exponential, lognormal, and logistic. Therefore, the method is essentially nonparametric. Mathematically this assumption is as follows: If $x_1, x_2, x_3, \ldots, x_n$ is a random sample from a distribution F(x), and if Z_n is the largest observed x, then the distribution function for Z_n is $F^n(x)$. If there exist sequences $a_n > 0$ and b_n for all n and a distribution function G such that

$$F^n(a_n x + b_n) \longrightarrow G(x) \quad \text{as} \quad n \longrightarrow \infty$$

for all x where G is continuous, then G(x) is an "extremal distribution function", and F(x) lies in its "domain of attraction" (5). This distribution function G must be from one of the following families of distributions:

$$\text{LAMBDA}(x) = \text{Exp}(-e^{-x}), \qquad -\infty < x < \infty$$

$$\text{PHI}_a(x) = \text{Exp}(-x^{-a}), \qquad x > 0, \; a > 0$$

$$\text{PSI}_a(x) = \text{Exp}(-(-x)^a), \qquad x < 0, \; a > 0$$

The Weissman technique uses only the largest k values of the random sample. The choice of the k is arbitrary, although it should be small in comparison to the sample size, e.g. k=10 and n=1000. The value of k is chosen prior to the examination of the data.

Once k is chosen and the limiting distribution family is determined, one of the following calculations is performed depending on the family. In each case below the order statistics of the random sample are represented by $X_{1n} >= X_{2n} >= \ldots >= X_{kn} >= \ldots >= X_{nn}$.

CASE 1 (G = LAMBDA)

$$1\text{-}c/n \text{ QUANTILE} = a_n[-\ln(c)] + b_n$$

$$\text{where} \quad a_n = \left(\sum_{i=1}^{k} (X_{in})/k \right) - X_{kn}$$

$$b_n = a_n \ln(k) + X_{kn}.$$

CASE 2 (G = PHI)

$$1\text{-}c/n \text{ QUANTILE} = (k/c)^{1/a} X_{kn}$$

$$\text{where } 1/a = \left(\sum_{i=1}^{k} \ln(X_{in})/k \right) - \ln(X_{kn}).$$

CASE 3 (G = PSI)

Since case 3 applies only to negative x, it is not appropriate for this application and is not discussed in this paper.

The remaining problem is the determination of the limiting distribution G. It is possible to test the hypothesis that G = LAMBDA by testing whether the set of normalized spacings

$$D_{1,n}, {}^2D_{2,n}, \ldots, {}^{(k-1)}D_{(k-1),n}$$

are independent, identically distributed exponential random variables, where

$$D_{i,n} = X_{in} - X_{(i+1)n}.$$

Similarly, the hypothesis that G = PHI can be tested by determining whether the normalized spacings of $\ln(X_{in})$ are independent, identically distributed exponential random variables. Standard statistical methods, such as the Gini statistic (6) are available for testing the exponentiality of a distribution.

The upper bound ε for our system was determined using Weissman's technique as follows. Prior to examination of the data, k was chosen to be 20, (i.e., one percent of the sample), as recommended by Weissman. The experimental data best supports LAMBDA as the limiting distribution; however, there was no clear rejection of either one of the limiting distributions. The standardized form of the Gini statistic was applied to the 19 normalized spacings of the 20 largest observations from the sample, and the corresponding observed significance levels for the tests were:

Family	Standardized Gini Statistic	Observed significance level
LAMBDA	0.5262	59.2%
PHI	1.449	14.7%

The inability to choose the limiting distribution with precision is of some concern here. Examining the test results for various values of k provides additional insight into discerning the limiting distribution family. In figure 6, the standardized Gini statistics for the LAMBDA and PHI tests using various values of k are plotted.

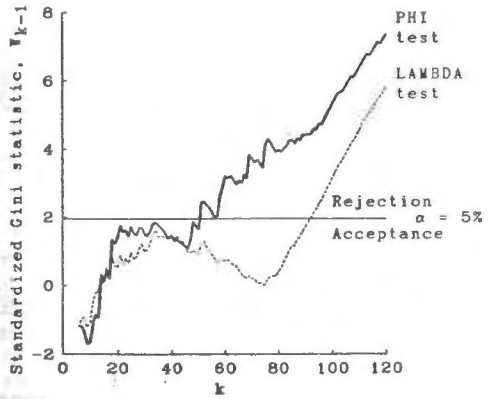


Fig. 6. Tests for limiting distribution family.

The tests show a consistent tendency towards selection of the LAMBDA distribution. In fact, for some values of k there is a strong rejection of the PHI and strong acceptance of the LAMBDA. As k becomes larger, eventually both models are rejected since Weissman's theory only applies to the tail of a distribution. Although the additional information obtained by varying k intuitively leads to a choice of the LAMBDA distribution, how to use such information has not been formalized statistically.

An alternate solution to the problem of discerning the limiting distribution family is to calculate the $1-p_\varepsilon$ quantile from both family models and use the most conservative value. However, sometimes the poorly fitting model gives astronomical values leading to unacceptable answers.

The remaining calculations are performed with the assumption that LAMBDA is the correct limiting distribution. The combinatorial analysis has shown that ε must be at least the $1-p_\varepsilon$ quantile to meet the system reliability requirements. Using the LAMBDA case analysis the $1-p_\varepsilon$ quantile was estimated to be 15.383 msec. Using this quantile to estimate the upper bound ε,

$$\hat{\varepsilon} = 15.383 \text{ msec.}$$

The conservative nature of this estimate may be seen by comparison with the maximum observation, 4.49 msec.

Validation Step 4

The directly measured and indirectly estimated values of the system parameters must be inserted into the theoretical expression for maximum clock skew from the theorem:

$$\delta \geq [N/(N-3m)]\left(2\varepsilon + \rho[R + 2(N-m)S/N]\right)$$
$$\delta \geq \delta_0 + \rho R$$
$$\delta << \text{Min}(R, \varepsilon/\rho).$$

The directly measured and indirectly estimated values of the system parameters are as follows:

$N = 4$
$m = 1$
$R = 30$ sec
$S = 615.334$ msec
$\varepsilon = 15.383$ msec
$\rho = 41.42657$ μsec/sec

The maximum clock skew can be computed from these values as follows:

$$\begin{aligned}\delta &= [N/(N - 3m)]\left(2\varepsilon + \rho[R + 2(N-m)S/N]\right)\\ &= 123.061 + 1.65706\times10^{-4}[30000 + 3(615.334)/2]\\ &= 128.185 \text{ msec.}\end{aligned}$$

Thus, the clocks will remain synchronized to within 128.185 msec with probability not less than $1-10^{-9}$ if the synchronization period is 30 sec. The contribution of the second term is small relative to the first. This reveals that in this implementation, the clocks are much more accurate than the interprocess communication subsystem.

Validation Step 5

As discussed previously, communication in a real-time system depends critically on synchronization being maintained within a certain bound. If the calculated skew is less than the bound used in the design of the communication subsystem, then the synchronization system has been validated. Otherwise, the real-time system must be redesigned if the reliability requirements are to be met. This may be accomplished by either slowing down the communications system (i.e., waiting longer for interprocess data) or by making improvements to reduce ρ and/or ε. The trade-off between performance and reliability will be explored in detail in the next section.

These validation steps can be reversed to compute system reliability given a specific design value for the maximum clock skew, δ. Essentially ρ is

estimated as above such that $p_1 = 10^{-7}$. Then ε is computed from the formula of the theorem, using these values of ρ and δ. Next, the probability that ε is exceeded, p_2, can be computed and hence p_{sys} determined. Thus, the system reliability is determined given a specific design choice for the maximum clock skew.

Additional Observations

A real-time communication system relies on synchronization between processors. The minimum communication time is the maximum communication delay plus the maximum clock skew, since the system must wait at least this long to insure a data value has arrived before accessing it. This minimum communication time represents the impact of the synchronization system on the performance of the real-time system. As shown in figure 7, as

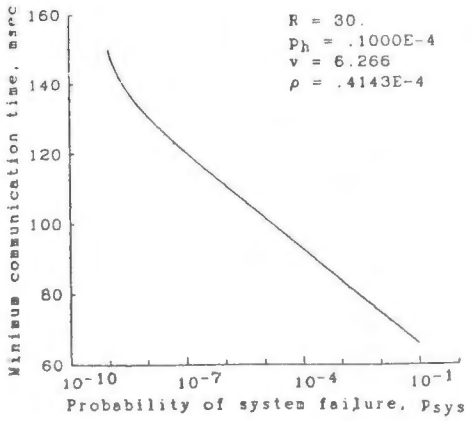


Fig. 7. Performance/reliability trade-off.

the reliability requirement is relaxed, the performance of the system increases. Also, the performance is a function of the synchronization period, R, or the fraction of time spent synchronizing, S/R. This performance/overhead trade-off is illustrated in figure 8.

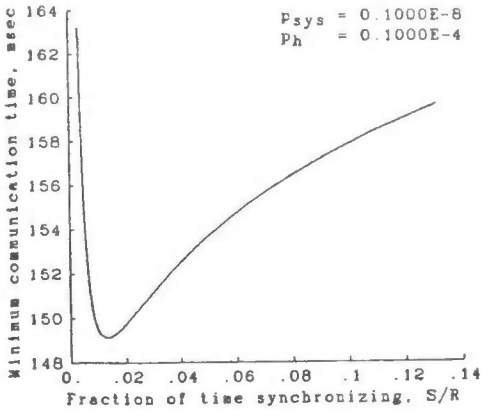


Fig. 8. Performance/overhead trade-off.

Conclusions

The validation method presented in this paper exploits the precision with which the formal proof method reduces the complexity of the system to verifiable axioms about the system behavior. The validation method introduced in this paper is essentially to: (1) perform a design proof of the synchronization algorithm under the assumption of low-level system behavior axioms, (2) perform a code proof of the synchronization code, (3) experimentally estimate the probability that the system behavior axioms will be violated and include these failure probabilities into a reliability analysis of the system. The SIFT synchronization design proof provided the basis for step (1). Step (2) has not yet been attempted, but will be performed under NASA contract NAS1-17067. Step (3) is explored in detail in this paper. The validation theory was applied to data obtained from an experimental system in AIRLAB instead of to actual SIFT data. The goal of this paper is to define a validation method rather than specifically validate the SIFT synchronization subsystem. After the development of the SIFT data retrieval system in early 1984, this theory will be applied to the SIFT hardware.

The design proof process reduces the performance of the clock synchronization algorithm to an algebraic expression of certain system parameters. These parameters, defined by formal axioms, represent worst-case bounds on system performance. By combinatorial analysis, the system reliability requirements can be translated to reliability requirements on these bounds. Because the estimation of a bound of a random variable is required, statistical methods applicable to the tail of a distribution are employed. The estimated parameters are substituted into the algebraic expression which calculates the worst-case performance of the synchronization system. This estimated worst-case performance (given the specified reliability constraints) is compared against the system design value. This validation process thus yields estimates of both performance and reliability.

References

(1) Goldberg, Jack, et. al.; Development and Analysis of the Software Implemented Fault Tolerance (SIFT) Computer, NASA CR-172146, 1984, pp. 145-165.

(2) Butler, Ricky W., and Johnson, Sally C.; Validation of a Fault-Tolerant Clock Synchronization System, NASA TP-2346, 1984.

(3) Feller, William; An Introduction to Probability Theory and Its Application, Volume I. Third ed. John Wiley & Sons, Inc., 1950, pp. 174-179.

(4) Weissman, Ishay; "Estimation of Parameters and Large Quantiles Based on the k Largest Observations"; Journal of the American Statistical Association; Dec. 1978, Vol. 73, No. 364, pp. 812-815.

(5) Pickands, James, III; "Statistical Inference Using Extreme Order Statistics"; The Annals of Statistics, Jan. 1975, Vol. 3, No. 1, pp. 119-131.

(6) Lawless, J. F; Statistical Models and Methods for Lifetime Data, John Wiley and Sons, New York, 1982, pp. 446-448.

TEST EXPERIENCE ON AN ULTRARELIABLE COMPUTER COMMUNICATION NETWORK

84-2649

Larry W. Abbott

Flight Systems Engineer
NASA Ames Research Center
Dryden Flight Research Facility
Edwards, California

Abstract

The dispersed sensor processing mesh (DSPM) is an experimental, ultrareliable, fault-tolerant computer communications network that exhibits an organic-like ability to regenerate itself after suffering damage. The regeneration is accomplished by two routines — grow and repair. This paper discusses the DSPM concept for achieving fault tolerance and provides a brief description of the mechanization of both the experiment and the six-node experimental network. The main topic of this paper is the system performance of the growth algorithm contained in the grow routine. The characteristics imbued to DSPM by the growth algorithm are also discussed. Data from an experimental DSPM network and software simulation of larger DSPM-type networks are used to examine the inherent limitation on growth time by the growth algorithm and the relationship of growth time to network size and topology.

Introduction

The dispersed sensor processing mesh (DSPM) is an ultrareliable structure for gathering sensor data and distributing effector data. An ultrareliable system requires an ultrareliable communications structure as a complementary partner to the ultrareliable computational element. The DSPM concept is the forerunner to the ultrareliable input/output (I/O) network specified in Ref. 1 for Charles Stark Draper Laboratory's advanced information processing system (AIPS).

The reliability of the DSPM network is greatly enhanced by the ability of the DSPM communication network to reconfigure (2). Two software algorithms — grow and repair (2,3) — perform the reconfiguration task. This paper describes the growth algorithm resident in the grow routine and the effects of the growth algorithm on the characteristics of the DSPM network. The characteristics of the experimental DSPM system are explained in terms of the experimental results, and the experimental results are used to certify the validity of the DSPM simulation software. The resulting validated DSPM simulation is used to derive the generic characteristics for a broad range of DSPM networks.

Concept

Figure 1 depicts a generic DSPM network taken from several examples given in Ref. 4. The network is formed with nodes (shown as circles), links (shown as lines), and a central bus controller (each channel of the quadraplex bus controller is shown as a rectangle). The growth algorithm and other network creation and maintenance software reside in the bus controller. The links carry the network communications, and the nodes gather and distribute data. A brief description of the growth algorithm (Fig. 2) is given; detailed descriptions can be found in Refs. 2 and 3.

Initially, the growth algorithm grows the network to the nodes surrounding the bus controller by activating the link (shown as a solid line) to each node (nodes 1, 2, 3, and 4) and making the node a member of the network. Activating a link requires that the destination node not be a member of the network and that it responds to the bus controller configuration commands.

In each successive growth cycle, the network is grown from nodes activated in the previous cycle until each node is attached to the network through a tree that has the bus controller as its root. During the growth process, the network is unavailable to process inputs to a node or outputs from a node. If faults exist in the links or in the nodes, the growth algorithm circumvents the fault in an "organic like" regeneration of the network by way of another configuration. The fault tolerance of the DSPM concept is a result of the ability of a DSPM network to reconfigure around failures.

Mechanization

The DSPM system is a complex, experimental communications system. To obtain valuable data on practical implementation issues, the ultrareliable DSPM communication concept was interfaced to a state-of-the-art fault-tolerant system and the tests were run on NASA's F-8 Ironbird simulator. NASA's F-8 digital fly-by-wire (DFBW) Ironbird simulator provides a safe, yet realistic means of testing the complex interaction of a highly reliable communication network with a state-of-the-art, triply redundant, digital flight-control system that contains redundant computers, sensors, actuators, and flight-critical software (5).

The DSPM experimental system (for additional details see Refs. 2 and 3) consists of three major components (see Fig. 3): the F-8 DFBW Ironbird simulator (with triplex flight-control computer and control-law software); a central computer with simulation software; and a six-node (plus a triplex bus controller where each channel is based on a 5 MHz MC68000 microprocessor) version of the DSPM network. The bus controller in the experimental system has relatively low performance; a production system should be expected to run much faster. The details of the DSPM hardware implementation are beyond the scope of this paper, but are presented in Refs. 2 and 6.

Experimental Results

The preliminary tests on the DSPM system measured the actual performance of the growth algorithm under varied fault conditions (for the purposes of this paper, a fault is a static failure of a port to the off condition). Performance data for the growth algorithm used on the Ironbird DSPM network (see Fig. 4) is available for all failure combinations up to three failures per node for both the adjacent and disjoint node pairs. An example of data for both an adjacent and a disjoint node pair is shown in Fig. 5. The data discloses two characteristics of the DSPM growth algorithm: (1) the mean growth time is linearly related to the number of faults, and (2) although the growth algorithm is deterministic, there is a wide deviation in the growth time for a given number of faults.

The deviation in growth time for a given number of failures on a given node pair is caused by the different amounts of time needed to process different types (2) and combinations of faults. The differences in growth time for configurations with the same number of failures is because of the topology of the network, the position of the node within the network, and the preferential order of growth (clockwise in DSPM). These deviations are not generic to the DSPM approach, but are dependent on the topology and the preferential growth algorithm. However, the relationship of growth time to inbound versus outbound ports is a generic DSPM characteristic. Specifically, on the experimental DSPM system, if a failed port is an outbound port (it relays bus controller (BC) messages to other nodes), the growth algorithm spends approximately 4.8 msec processing the failed port. If the failed port is an inbound port (it returns node responses to the BC), the growth algorithm spends about 7.7 msec processing the failed port. The fact that an outbound port must be grown before the inbound port of the next node can be grown accounts for the difference in growth time. Thus a fault on an outbound port can always be detected sooner than a fault on an inbound port.

While the linearity of the growth times are influenced by the implementation of the hardware and software in each DSPM-type system, the linear nature of the growth times is a generic DSPM characteristic. All fully operable DSPM configurations must have exactly the same number of good links as nodes and, because the growth time for good links varies very little, the only difference in growth time is related to the number of faulty links. While the time required to determine if a link is faulty varies, depending on whether it is used as an inbound or an outbound link, the values are roughly the same, and the relationship turns out to be approximately linear.

In Fig. 5, note that for a few faults, the growth times for failure sets on disjoint and adjacent node pairs are appoximately the same; but, as more faults are injected, the disjoint node pair requires more growth time than an adjacent node pair. A single fault in a disjoint node pair appears the same as a single fault in an adjacent node pair. Therefore, it is not surprising that the growth times are similar for a few faults. As the number of faults increase, the failures on the disjoint node pair normally disrupt two or more trees while failures on the adjacent node pairs, which share a common link, normally disrupt a single tree.

Naturally, one might ask what happens if more failures are injected into the network than are shown in Fig. 5. Can we linearly extrapolate the growth time? Unfortunately, the answer is no. There are two reasons for this. First of all, each nonlatent fault (a latent fault is a fault in a link with an existing dominate fault or a fault in a link that will not be activated) causes the link associated with it to fail during an activation attempt. Because valid DSPM configurations must have a good link for each node in the network, the maximum number of failed links (and therefore nonlatent faults) equal the number of links minus the number of nodes (see Eq. (1)).

$$\text{Maximum number of faults} = \text{number of links} - \text{number of nodes} \qquad (1)$$

For the Ironbird DSPM network that is used on the F-8 Ironbird simulation (see Fig. 4), the maximum number of faults is six.

The second reason that the growth time cannot be linearly extrapolated provides much better insight into the behavior of the network. Exceeding the maximum number of nonlatent faults replaces inbound failures with outbound failures. However, to see this behavior, latent faults that occur in links where activation attempts occur must be counted as faults.

As shown in the following maximum growth time scenario, when all the faults previously defined are counted, the growth time still does not exceed the growth time for the maximum number of nonlatent failures, even when the number of faults exceed the maximum number of nonlatent failures. Because the scenario is constructed with a homogenous fault type (that is, worst case faults), the linear relationship of growth time is shown without any deviation. An example of maximum growth time is to grow the DSPM network with the maximum growth time by using the following heuristic rules to choose a worst case fault: (1) failures on an inbound port contribute more to growth time than failures on an outbound port and (2) growth at a node proceeds clockwise with port 0 first and port 3 last.

To achieve the maximum growth time we must grow a network with as many inbound ports failures as possible and with growth occurring at the most counterclockwise port possible. The following growth phases form the maximum growth time scenario for the Ironbird DSPM network.

Phase 1: Grow out of the most counterclockwise port of the BC. To force the growth to BC port 2, the links to BC ports 0 and 1 must be failed at one end or the other (see Fig. 6(a)). To obtain the maximum growth time, the first failure (at an inbound port) is injected at node 1, port 0 (N1P0) and results in a growth time of 24.8 msec. The process is cumulative. For two failures, a failure at N4P0 is added to the failure at N1P0, resulting in a growth time of 31.4 msec.

Number of failures	Fail vector	Growth time
1	N1P0	24.8 msec
2	N1P0, N4P0	31.4 msec

Phase 2: Grow out of the most counterclockwise port of node 6. Force the growth to N6P3 by failing N2P2 and N5P2 (see Fig. 6(b)).

Number of failures	Fail vector	Growth time
3	N1P0, N4P0, N2P2	39.1 msec
4	N1P0, N4P0, N2P2, N5P1	46.8 msec

Phase 3: Grow out of the most counterclockwise port of node 4. Force the growth to port 3 of node 4 by failing N5P0 (see Fig. 6(c)).

Number of failures	Fail vector	Growth time
5	N1P0, N4P0, N2P2, N5P1, N5P0	54.5 msec

Phase 4: Grow out of the most counterclockwise port of node 1. Force the growth to port 3 of node 1 by failing N3P0 (see Fig. 6(d)).

Number of failures	Fail vector	Growth time
6	N1P0, N4P0, N2P2, N5P1, N5P0, N3P0	62.1 msec

When six maximum growth time faults are injected into the Ironbird DSPM network, only six links remain, which is the minimum number of links required for this network to function. Any additional failure must be injected into links that have already failed. Since all the links have failed at the inbound end, any new failure would occur at the outbound end and would reduce the growth time shown in Fig. 7 (each point in Fig. 7 is the accumulation of all the previous failures plus the failure listed as a label for the point). Therefore, the maximum growth time occurs at the maximum number of faults defined by Eq. (1).

Simulation

Research on systems such as the DSPM is expensive. Because of the cost, building large DSPM systems or systems with special attributes solely for research is not feasible. As an alternative, simulation software offers a means of studying larger networks with different attributes while still keeping down the cost.

The simulation software was designed to mimic the detailed growth algorithm flowchart used in the development of the DSPM. Each block or group of blocks in the flowchart was timed experimentally to establish the constituent times for the simulation. Each function in the flowchart was functionally implemented in the simulation, and the appropriate time was added to the total simulation time whenever the function was performed.

The validity of the simulation was established by exhaustively comparing the results of the simulation with the actual data from similar known situations. After many simulation runs the simulated growth time for a fault-free network was a good approximation to the actual time. The simulation then had to be validated for faulty configurations.

The simulation was designed to allow the user to fail links or nodes. Additional tests using the fault injection capability established the validity of the simulation in a faulty environment. Figure 8 demonstrates the accuracy of the simulation.

In Ref. 2, the DSPM6 network (Fig. 9) is determined to be the smallest DSPM-type network acceptable for applications requiring ultrareliability. Accordingly, DSPM6 is used as the lower bounds for DSPM network performance. Because DSPM16 (Fig. 1) appears frequently in literature (2, 3, and 4), it was arbitrarily chosen as the upper bound on DSPM performance. The area between the bounds is filled in with simulated data from DSPM8 (see Fig. 10) and DSPM11 (see Fig. 11). All the simulated networks were grown by using a heuristic algorithm to achieve near-worst case growth times.

The fault-free growth times of all the simulated networks is plotted in Fig. 12. Because of the linear relationship of fault-free growth time to the number of nodes in the network, and the linear relationship of incremental growth time and failure, a simple model of worst case growth time (G(n,f)) is possible for a network of nodes (n) and failures (f). The graphical form for such a model is shown in Fig. 13. The mathematical form is given as Eq. (2).

$$G(n,f) = 3.46n + 7.7f \text{ in msec} \quad (2)$$

As an example, the worst case growth time for a 32-node network with five failures is

$$G(32,5) = 3.46 \times 32 + 7.7 \times 5 = 149.2 \text{ msec} \quad (3)$$

By restricting the growth simulation to the maximum growth time scenario, the worst case times are obtained and the growth time relationship is linear with the number of failures. For all simulated networks, there is a 7.7 msec increment between failures. For instance, in DSPM6 the difference in growth time between three failures (40.2 msec actual) and four failures (47.9 msec actual) is 7.7 msec.

Given the fault-free growth time for a specific network, a good approximation of worst case growth times can be obtained with a straight line with a slope of 7.7 msec per failure. Further, the fact that the relationship is true for four networks representing three different topologies (DSPM6, DSPM16, and DSPM8/DSPM11) strongly indicates that the linear relationship is a generic DSPM characteristic.

The growth time of a network should be linearly related to the number of links that the network must grow. Because one link must be grown to every node in the network, the fault-free growth time of a 16-node network should be twice the fault-free growth time of an 8-node network. For DSPM16, the fault-free growth time is 51.6 msec, which is approximately twice the 24.0 msec fault-free growth of DSPM8.

Given 149 msec for G(32,5), or even 127 msec for G(16,10), one could question whether the DSPM

growth process is fast enough. During the growth process, no inputs or outputs can traverse the DSPM network. As a result, the aircraft would be flying open loop, and any departure would continue until the process is over and one control-law update is complete. This break in the control-law update certainly must be considered in any vehicle or system design; however, the maximum growth time is expected to be somewhat shorter given improved computational hardware that would be readily available in any operational application.

Conclusions

Tests show that the generic growth characteristics of DSPM-type systems are independent of the network topology. They also show that growth time is linearly dependent on the number of nodes and the number of failures occurring in the DSPM network. However, tests show that the growth time increases as the number of failures increase and the growth time is bounded. As a result of linear and bounded growth times, the growth time relationship can be modeled by a simple, accurate linear equation.

References

(1) "Advanced Information Processing System (AIPS) System Specification," CSDL-C-5709, Charles Stark Draper Laboratory, May 1984.

(2) Abbott, Larry W., "An Ultra Reliable Computer Communication Network — the Dispersed Sensor Processing Mesh," University of Kansas, Jan. 1984.

(3) Abbott, Larry W., "Operational Characteristics of the Dispersed Sensor Processor Mesh," Proceedings of the IEEE/AIAA 5th Digital Avionics Systems Conference, Seattle, Wash., Oct. 31-Nov. 3, 1983, pp. 9.4.1 to 9.4.8.

(4) Hopkins, Albert L. and Brock, Larry D., "Interim Report on Fault-Tolerant Aircraft Signal and Power Transmission Structures," R-1298, Charles Stark Draper Laboratory, Aug. 1978.

(5) Szalai, Kenneth J., Jarvis, Calvin R., Krier, Gary E., Megna, Vincent A., Brock, Larry D., and O'Donnel, Robert N., "Digital Fly-By-Wire Flight Control Validation Experience," NASA TM-72860, Dec. 1978.

(6) Megna, Vincent A., "Tactical Airborne Distributed Computing and Networks," Dispersed Sensor Processing Mesh Project, AGARD-CCP-303, June 1981, pp. 30-1 to 30-14.

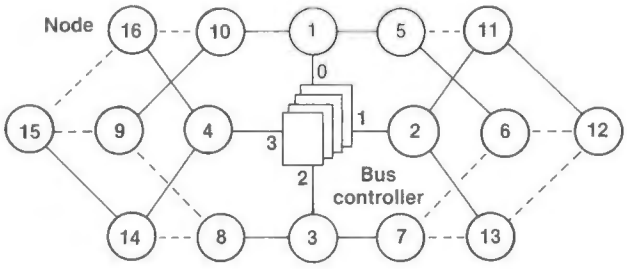


Fig. 1. DSPM16 generic DSPM-type network.

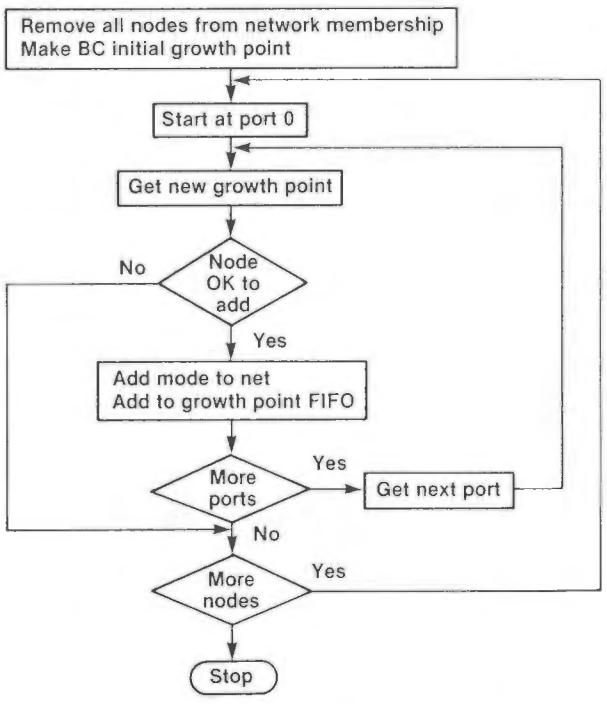


Fig. 2. Simplified growth algorithm.

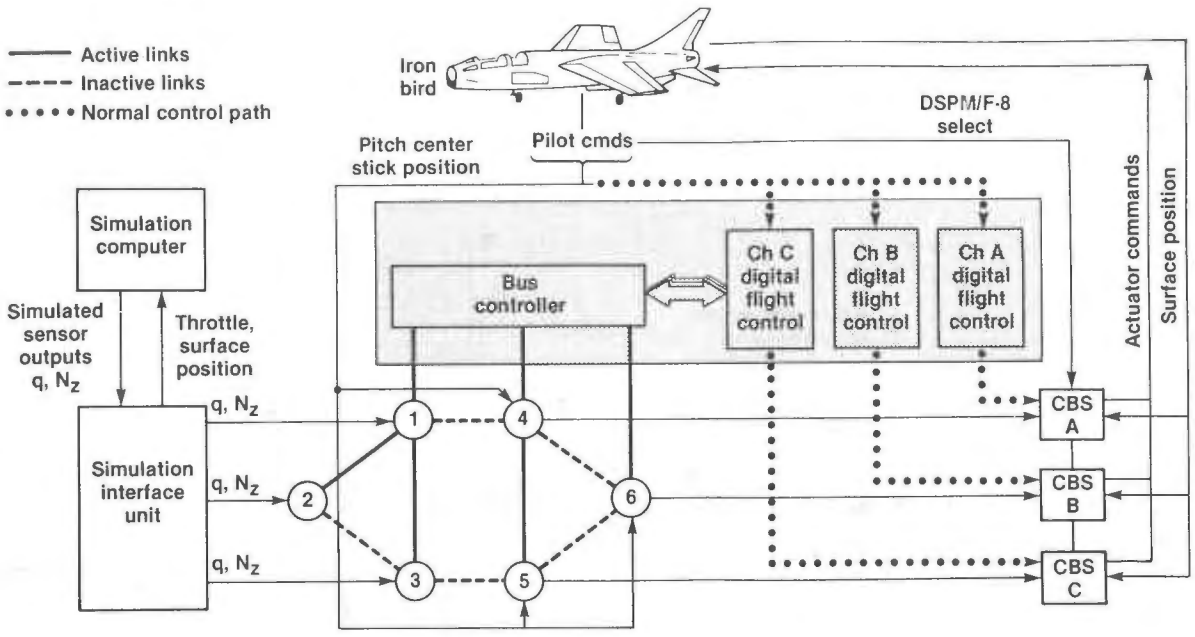


Fig. 3. DSPM experimental system.

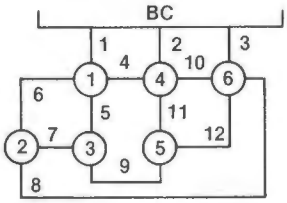


Fig. 4. Ironbird DSPM network used on the experimental system.

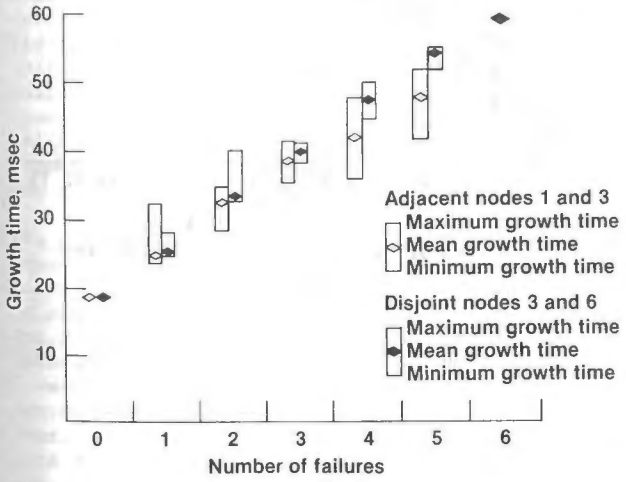


Fig. 5. Number of combined failures occurring on a node pair for two sets of node pairs.

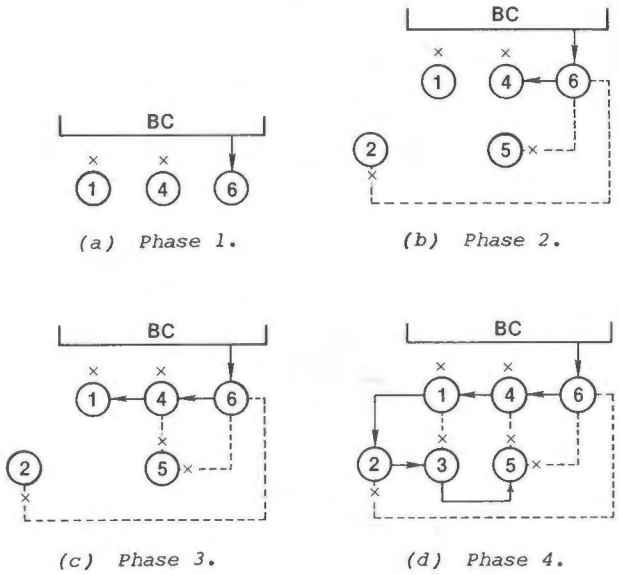


Fig. 6. Maximum growth time scenario.

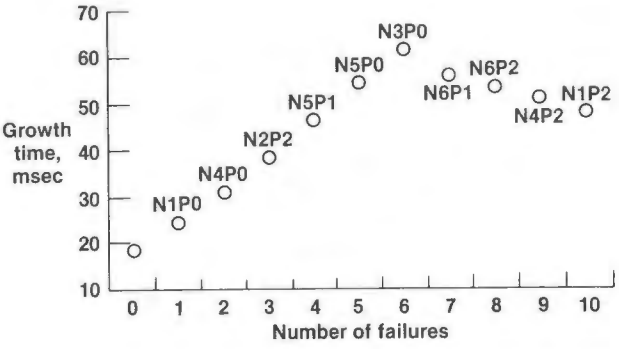


Fig. 7. Maximum growth time as a function of the number of faults.

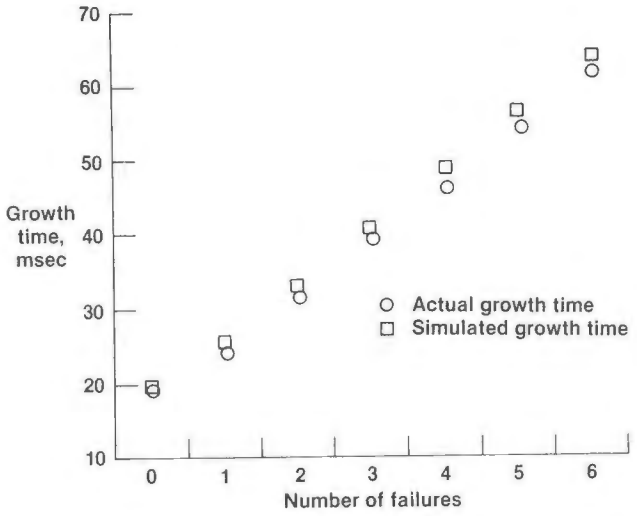


Fig. 8. Simulated versus actual growth time for a maximum growth time example (Ironbird DSPM).

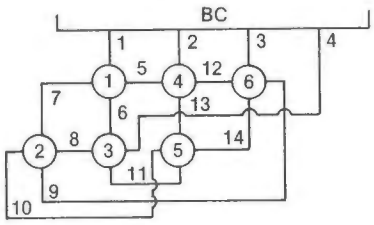


Fig. 9. DSPM6 network.

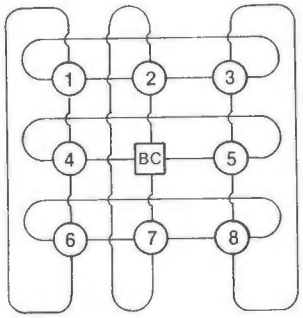


Fig. 10. DSPM8 network.

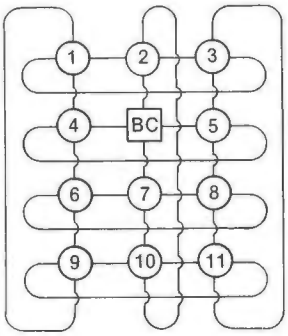


Fig. 11. DSPM11 network.

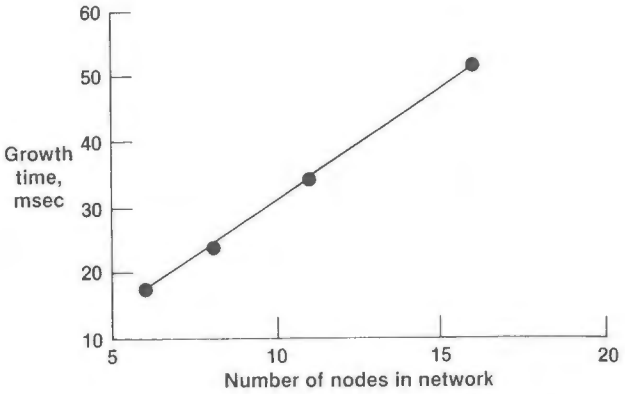


Fig. 12. Fault-free growth time as a function of number of nodes.

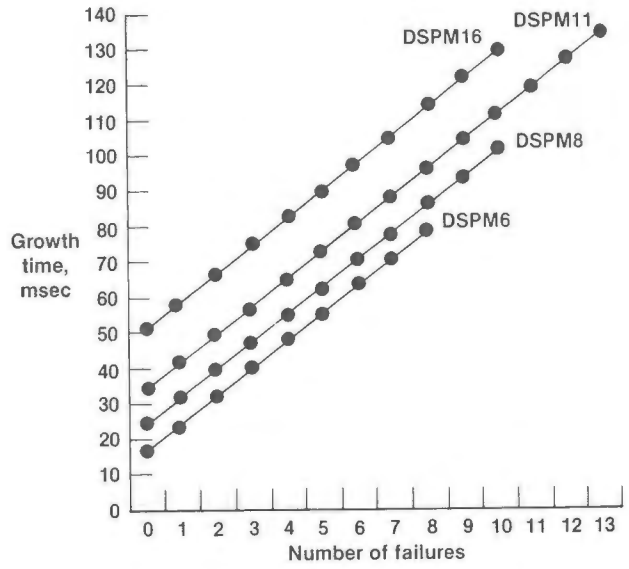


Fig. 13. Worst case growth time G(n,f) for networks with nodes (n) and failures (f).

ULTRARELIABLE FAULT-TOLERANT CONTROL SYSTEMS

84-2650

Larry D. Webster, Roger A. Slykhouse, Lawrence A. Booth, Jr.,
Thomas M. Carson, Gloria J. Davis, and James C. Howard

NASA Ames Research Center
Moffett Field, California

Abstract

An Ultrareliable Fault-Tolerant Control System (UFTCS) concept is described using a systems design philosophy which allows development of system structures containing virtually no common elements. Common elements limit achievable system reliability and can cause catastrophic loss of fault-tolerant system function. The UFTCS concept provides the means for removing common system elements by permitting the elements of the system to operate as independent, uncoupled entities. Multiple versions of the application program are run on dissimilar hardware. Fault tolerance is achieved through the use of static redundancy management.

Introduction

The use of redundant concepts enables fault-tolerant systems to be developed which are significantly more reliable than their simplex counterparts. Unfortunately, no system can be developed which can be shown to be entirely free from design error. Latent faults can exist which will ultimately cause failure. The first attempt to launch the Space Shuttle Columbia was aborted as a result of a latent design fault in a common system feature: the computer synchronization scheme. Fortunately, the failure was not catastrophic. However, it was costly in terms of financial loss, schedule delay, and prestige.

Fault-tolerant systems which do not contain features common to redundant elements are inherently more reliable than system designs which do contain common features. The effect of common elements on fault-tolerant performance generally are not considered when the reliability of a given system is calculated. The difference between concepts containing common elements and those not containing common elements will become apparent when the reliability of the operational system is assessed. Common elements are not required in systems where the redundant elements operate independently.

The objective of our work, both theoretical and experimental, is to provide proof that fault-tolerant systems based upon independent operation of redundant elements is a practical alternative for fault tolerant system designs. To this end, laboratory simulations were developed which allowed investigation of the responses of various control system structures in the fault-tolerant environment, and the structuring and analysis of redundant-data management (voting) schemes. A typical example of an Ultrareliable Fault-Tolerant Control System (UFTCS) was constructed, using the architecture proposed by Dunn and Meyer (1), to allow investigations into the theoretical and practical bases of the concept with operational hardware and software. The control system implemented in this experimental test bed was that of a UH-1H helicopter. Some results of the experimental research are presented which establish an empirical basis indicating that ultrareliable systems which employ independent, uncoupled redundant elements can be produced.

Project History

The postulate that UFTCS systems could be developed from completely independent elements came from research begun in the mid-1970s at NASA Ames Research Center. Dunn and Meyer recognized that significant increases in the computational power of microelectronics, coupled with similar decreases in their cost, size, and weight would permit the development of ultrareliable fault-tolerant system designs (1). They developed a system structure which uses an efficient blend of hardware and software to achieve fault tolerance based upon independent redundant elements. The resulting UFTCS system concept consisted of asymptotically stable independent control elements in a parallel, cross-strapped system environment. Fault tolerance was achieved through the use of static redundancy management (2).

Static redundancy management was previously impractical because of the enormous penalties of cost, size, weight, and power incurred in n-module (replicated) redundant systems. Because of this, the SIFT (3) and FTMP(4) fault tolerant programs concentrated on hardware conservation techniques based on dynamic redundancy management. These techniques trade off hardware size for software and general system complexity. The current computational power available in today's microelectronic technology makes possible the use of N-module, static redundancy in fault-tolerant system concepts.

Using the original work as a basis, the authors have developed laboratory facilities, simulators, and an operating experimental test bed capable of examining the fundamentals of the theoretical work. The laboratory has the capacity to synthesize and analyze ultrareliable fault-tolerant systems concepts. The experimental test bed provides a real-time implementation of a UH-1H helicopter control system and is used to examine the characteristics of independently operated fault-tolerant elements. It consists of autonomous elements configured to operate independently, in parallel, and can be arranged in quadruple, triple, or dual redundant configurations. The UH-1H control algorithm implemented is asymptotically stable (5,6).

Ultrareliable Concepts

Ultrareliable Fault Tolerance—A Definition

Ultrareliability can be achieved in a fault-tolerant system by eliminating the potential of system failure due to latent design errors which exist in functions held "in common" among the redundant elements. The presence of these latent common faults in any design can be assumed. Reliability analyses

of fault-tolerant systems do not account for the presence of the latent design error (some general work has, however, been accomplished in this area (7)). Therefore, the demonstrable reliability of a fault-tolerant system which contains common failure modes will be less than the calculated reliability due to the presence of latent design faults. Ultrareliable fault tolerance can be defined as "a concept which produces ultrareliable systems that have the capability of delivering expected reliability."

Ultrareliable Fault Tolerance—The Criteria

Historically, the development of fault-tolerant systems has been motivated by the persistent need of the user for ever-increasing system reliability. Numerous fault-tolerant system concepts have been developed to satisfy this need. Each fault-tolerant structure is constructed attempting to satisfy two basic criteria: (1) the redundant system elements must be independent and (2) the redundantly generated outputs must be unambiguous.

It is the relative compliance with these two criteria which determines the inherent reliability of a given fault-tolerant structure. In the past, it has not been possible to comply completely with both criteria for reasons which are discussed later. Usually, a satisfactory compromise between compliance and system structure will be reached which satisfies the second criterion, but only partially satisfies the first.

On initial consideration, it appears that these two criteria are mutually exclusive. The first condition establishes that the redundant elements must be independent and totally autonomous with no common features. But the second criterion requires that the outputs produced by the independent redundant elements cannot be ambiguous. For functioning elements, the output values which are generated must be close enough in value throughout time so that logical comparisons can be made. These comparisions are performed by a portion of the system, usually known as the "voter," which is responsible for determining the failure status of the redundant elements.

Independent Elements—A Historical Perspective

It would seem that independent elements executing, for example, a flight control algorithm would not produce control values which could be logically voted for any length of time. Under certain conditions, this has been experimentally shown to be the case. The original system architecture of the AFTI-F16 triplex, fault-tolerant system was structured such that the elements were independent. However, tests of the system, as described by Mackall (8), disclosed significant differences between redundantly generated control values. The magnitude and unpredictability of the discrepancies made channel selection and fault detection virtually impossible. In attempting to comply with the first fault-tolerant criterion, it became impossible to fulfill the second. Steps to correct the problem were implemented. A second case, described by Osder (9), disclosed the results obtained from experimental hardware-in-the-loop simulations of tri/quadraplex fault-tolerant systems employing independent elements. As with the AFTI-F16 control system experience, significant differences existing between independently generated redundant data were recorded and a method was devised to correct for the differences. It is interesting to note that both systems solved the divergence problem by implementing a form of cross-channel equalization to force the control values generated to conform to each other. Regardless of the technique employed, the intent of cross-communications between the redundant elements is the same. Osder (9) properly describes the need "to correct static or long-term differences (between the control law processors) so that the channels track . . ."

However, the use of cross-channel communications to force the generation of votable, redundant, output sets violates the first premise upon which ultrareliable concepts are based: the system elements must be independent. Cross-channel communications are needed only if static or long-term differences between redundant elements exist. Both requirements for the development of ultrareliable fault-tolerant systems are met by a systems concept which contains independent elements whose outputs do not produce long-term differences. In the next section, the mechanism through which these differences are produced is demonstrated. Several simple techniques which may be employed to cause long-term differences between redundant element outputs to effectively disappear without resort to cross-channel communication is then displayed in a later section.

Sources of Common Elements

The generation of fault-tolerant system concepts based upon independent elements requires the implementation of certain philosophies which maintain the independent relationship. By specifying that the redundant elements must be independent, systems are produced which are common element free. Common elements can reside not only in the hardware, but also in the application programs embedded in the hardware and the philosophies involving redundancy management. All three areas of the system structure contain the human element. The sources of commonality in fault-tolerant systems are: (1) use of global redundancy management schemes, (2) use of redundancy synchronization, (3) cross-communication between redundant elements, (4) replication of identical hardware, and (5) replication of identical application programs.

Common Element Free Concepts

Unsynchronized Operation. The need for tight synchronization of the redundant elements is predicated on the requirement that the output values generated by these elements be votable. Synchronization is used to force the redundant elements to march in lock-step, transducing input state variables together and providing computed outputs for voting simultaneously. Synchronization is used because it forces the redundant channels to track, thereby making the voting process simple. Thus, synchronization fulfills the criterion that the generated outputs must be unambiguous. However, a synchronized fault-tolerant system violates the criterion that the redundant elements must be independent. Because of this, the synchronized system cannot fulfill projected reliability as the synchronization scheme (hardware, software, and human element) represent an uncalculated common element in which a latent fault will cause total loss of system function. Further, synchronized systems tend to contain other sources of common elements such as the use of identical hardware and application programs.

To make the resulting system more reliable, unsynchronized redundant elements should be used.

Multiple-Version Redundant Elements. The same reasoning can be invoked to discuss the use of (1) identical hardware, (2) identical application programs, and (3) cross communication. All three methods of implementing redundancy violate the independence criterion. The human element is involved with the development of each. They can, therefore, contain latent faults embedded into the design as a result of human error which, when disclosed, could lead to catastrophic loss of system function. The use of cross-channel communication is unnecessary if the outputs produced are unambiguous. The use of identical hardware and application programs should be supplanted with the use of dissimilar redundancy for each item. The voting techniques implemented need not be identical; different voting philosophies may be used.

Static Redundancy Management. Independent operation of system elements obviates the use of the global or dynamic forms of redundancy management. The philosophy of static redundancy management is employed instead. Static redundancy management eliminates the interelement coupling found in fault-tolerant systems which employ dynamic or analytic redundancy management. The static method of redundancy management may best be described as management wherein nothing of system-wide significance occurs to the system structure on the advent and detection of elemental failure.

Static redundancy management eliminates the need for global redundancy managers performing dynamic reconfiguration of the system. Statically managed systems do not have the common element represented by the global management scheme and are therefore more reliable. Moreover, static redundancy management does not require system-wide knowledge of how the system can fail, global knowledge of the current fault status of the system, nor development of any system-wide reconfiguration strategy. The static fault managers are autonomous entities which neither share fault status nor cause any external activity when failure of system elements are detected. Basically, it is the function of the manager to control the flow of redundantly generated data from "Task A" to "Task B" as shown in Fig. 1. Incoming data sets are correlated by the voter and only uncorrupted data reach the following task.

In conclusion, fault-tolerant systems employing independently operated redundant elements controlled through static redundancy management and whose system elements perform different versions of the application programs (both process and redundancy management) in dissimilar hardware are inherently more reliable than those which do not.

Sources of Channel Divergence

Ultrareliable systems can only be generated if the elements which perform redundant processes are functionally independent and the outputs they produce are unambiguous. However, the system design practices which have currently attained industry-wide acceptance do not allow for independent operation of the redundant elements. The need to produce unambiguous results has led industry to develop methods which obviate the sources of "channel divergence." These methods also cause the redundant elements to become highly coupled and

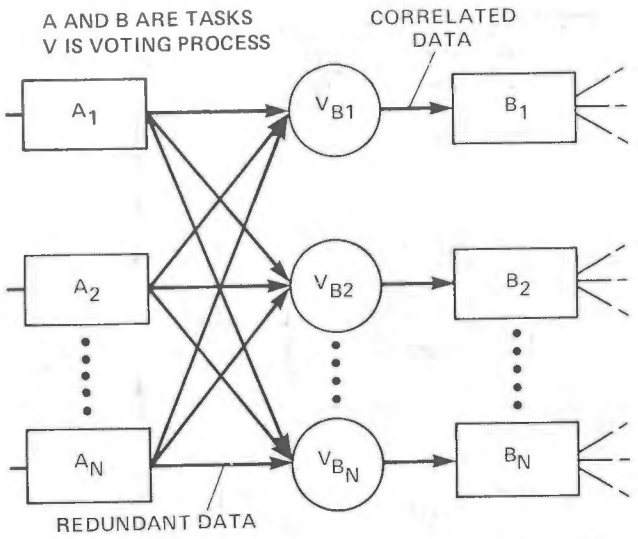


Fig. 1 Static redundancy management with independent elements.

dependent. It is clear that other methods need to be found which allow for both unambiguous output and independent elements. But it is first necessary to fully understand the origins of channel divergence and the effect that each source has upon the magnitude of the divergence between channels. One important variation in philosophy will be introduced which takes the criterion of unambiguous results literally. In the fault-tolerant system, it is not necessary that the agreement between results, as determined by the system voting elements, be perfect. It is only necessary that the results not be ambiguous. It is acceptable to relax the voting criteria to one which, for example, will tolerate long term (10+ sec) channel differences of 1% of full scale, and short term differences (300 ms) of 3% or 4% of full scale. This relaxation has little or no effect on field reliability.

Variation between Personal Clocks. As shown in Fig. 2, each redundant computer contains a source of "personal time." The base of this time is usually a crystal-controlled oscillator within each computer. It is a practical impossibility that the relative passage of personal time noted by Computer 1 will be identical to that noted by Computer 2 or Computer N at any instant. With all other factors equal, results of computed functions which are based on personal time will vary from computer to computer. Digital integration is a process affected by the computer's perception of time. Time differences between computers appear as if the constant of integration is varied between the N computers. For processes which are cyclic (roll or pitch control, etc.), the divergence between results peaks when the cycle peaks and returns to zero when the cycle returns to its null point. Channel divergence does not accumulate, and can be made small through proper specification of the time accuracy between redundant computers. For integration processes which are not cyclic, such as are found in navigation, the perceived differences in time accumulate. Given sufficient time, channel divergence (caused by various channels wanting to

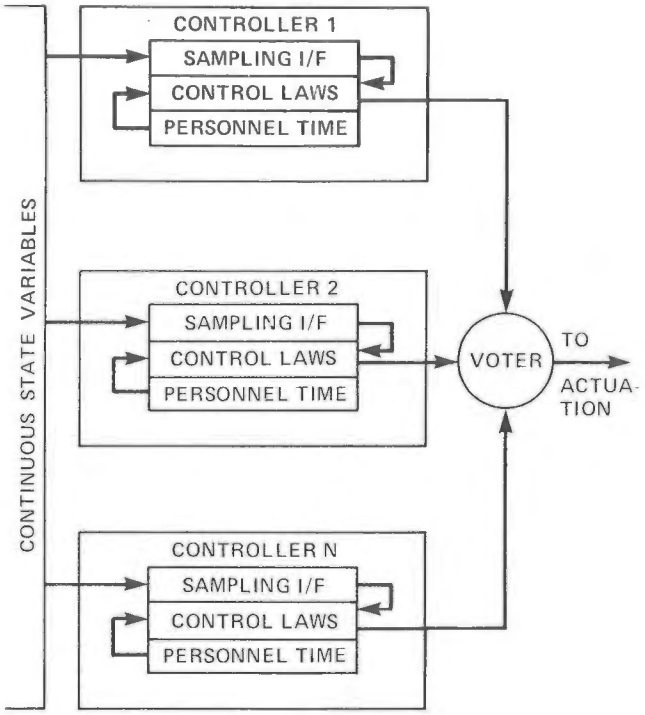


Fig. 2 Independent redundant controllers.

be in different places at the same instant of real time) will grow until channel failure is declared by the voter. If mission time is short compared to the intercomputer time drift rate, then this effect can be ignored. If mission times are long, such as in autonavigation spaceflight, this variation must be taken into account.

Personal Clock Inaccuracies. The personal clocks shown in Fig. 2 not only vary relative to each other, but drift relative to the value of real time. If the application program presumes a knowledge of real time (e.g., a Euler first order integration) and the real time differs from presumed time, output variations from proper outputs (those generated if the presumed time were equal to real time) occur from the point of view of the voter. As above, integration errors caused by cyclic operation of the system return to zero when the system returns to its initial condition, and noncyclic errors accumulate.

Bias, Scale, and Nonlinear Factor Variations. The sampling interfaces shown in Fig. 2 are replicated and independent. As physical devices, it is not possible to produce N samplers with identical properties. Therefore, even if the state variable set was transduced at exactly the same moment in real time, it can be assumed that each independent computer will subsequently not contain exactly identical digital values for each of the variables. The differences appear as bias, scale, and nonlinear variations in the sampled input data sets from computer to computer. When the application program operates on different data, different results will be recorded at the voter. When the control law contains free integrators, the effect is pronounced as the effect is summed from cycle to cycle.

Sampling Skew and Sampling Rate. These two related parameters have the most pronounced effect on the channel divergence. State variable dynamics, sampling rates (the data are processed between samples), and varying sample skews will combine to produce situations where the channel divergence varies in what would seem a random manner. However, if all of the system variables were known, the process is actually deterministic. A simple timing diagram, Fig. 3, displays a possible relationship of the N independent computers shown in Fig. 2. Note that in independently operated redundant elements, both the sample skew and sample rate may vary with time.

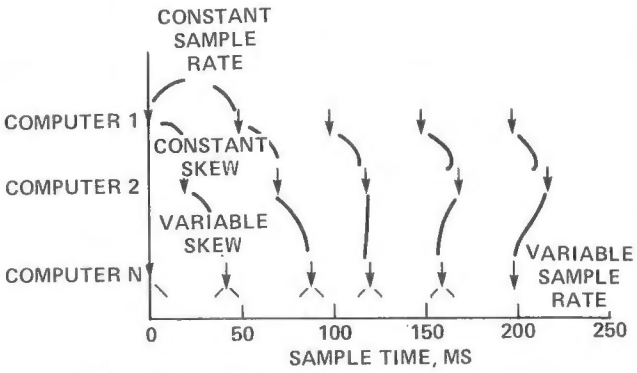


Fig. 3 Sample rate and sample skew.

If redundant processes do not sample dynamic state variables at the same instant of real time, variations between redundantly computed outputs will be recorded by the voter. Sampling skew is the difference in time between input state variable samples taken by one redundant computer relative to the sample time of any other. The sample skew relationship can be fixed or can vary as a function of the loop time of the application program. Presume, for example, that Computer 1 and Computer N have exactly the same real time sampling rate (say 20 ms) and are identical in all other ways except that Computer N always samples data 9.3 ms after Computer 1. As the value of the input data is changing from instant to instant, Computer N will "see" different values for sampled variables than will Computer 1. If a particular parameter is increasing in value, then Computer N will see a larger value than will Computer 1. As the output of the computational process is based upon the value of the input parameters, the redundant computers must produce output values which are different. The instantaneous amount of channel divergence recorded at the voter is dependent upon the absolute difference of the variable value between the time it is evaluated by Computer 1 and Computer N (how fast the parameter is moving), and how the parameter is used by the control law.

Two factors dealing with sampling rates affect channel divergence: the absolute real time sample rate and the relative sampling rate of one computer to another. The first factor, in conjunction with sampling skew, determines the absolute difference between the value sampled by Computer 1 and the value sampled by Computer N for state variables which are dynamic. For a given variable dynamics and sampling skew, increasing the sampling rate for all of the redundant computers will cause the absolute value difference between samples to decrease. Regardless of inter-computer skew, the amount of channel divergence decreases as sampling rate increases.

If the redundant channels have different or dynamic (e.g., data driven programs) sample rates, the sampling skew of Computer 1 to Computer N will vary with time. The closer the times are to each other, the slower the skew varies. This can effect channel divergence by causing the relationship of the redundant channels to change with increasing time. Depending on the dynamics of the sampled data, this can either increase or decrease channel divergence.

Use of Free Integrators. "Integral control" is an extremely useful tool to the control systems designer. Its main use is to control long-term variations in or compensate for incomplete definition of the state of the plant being controlled. It provides zero-error, steady state control of the plant. Integrators also contain the sum of all past events occurring at the input to the integration process. If the control laws processed by the redundant computers of Fig. 2 contain free integrators, then the sampling, skew, and personal time differences will cause the output values produced by the integrators to vary in effectively a random fashion. For instantaneous input differences, the same may be said for lead/lag or constant-gain circuits. The difference with the pure integrators is that when the input difference goes to zero, the difference between the integrator outputs will not. The integrator will "remember" the past history and maintain the difference indefinitely. Given that the process of channel divergence owing to integration is effectively a random one, scenarios can be easily constructed wherein the integrators cause the channel divergence to increase to indefinitely large values. Lead/lag and constant-gain elements do not contribute to long-term or steady state channel divergence.

Nullifying Channel Divergence—A Methodology

The factors which cause channel divergence are a function of the practical constraints in producing redundant (replicated) elements, and the skew/sampling relationship between them. These involve the method of input data collection of each redundant member. In cycling through the transfer function, which must be identical from redundant computer to redundant computer, output values are produced which are predicated on the sampled value of the input. At any given instant, the redundant values produced by independently operated redundancy will be different. To produce unambiguous results, the tracking error must be kept within acceptable limits. Unambiguous means, for example, that the channels track to within 1% of full scale on average and the instantaneous variations cannot exceed 4% of full scale. Also the time to declare a fault can be extended from one or two frames to several seconds of disagreement. Nothing is lost through relaxation of the failure-detection parameters. The fault-tolerant structure will not propagate improper or faulty data. What is gained is a relaxed failure criterion which permits the fault-tolerant system designer to employ design techniques that do not require common elements in the system structure: As the results do not have to agree exactly, different application programs, different voting routines, dissimilar hardware, and independent redundant elements may be employed. Common latent design faults will not then exist if the concept of static redundancy management is also employed.

Use of Asymptotically Stable Controllers. The differences which are inherent between independently operated redundant computers cannot be allowed to cause channel divergence which exceeds a reasonably small time/magnitude threshold at the voter. The threshold need not be static, but can be varied as a function of the dynamics of the voted variables. Instantaneous differences caused by the mechanics of data sampling and time generation can be minimized by specifying sufficiently tight tolerance on the hardware that constitutes these functions. This specification is the one item over which the designer has good control. Free, pure integration terms cannot be allowed in the control system design. However, the integration terms can be approximated by lead/lag elements with long time constants. What is lost is the steady state zero tracking error feature of control loops which employ free integrators. What is gained is that the lead/lag element will "forget" the short-term differences between redundant channels. Instantaneous differences will still exist, but their effects on channel output will be reduced to zero as time passes.

A laboratory simulator has been developed which provides insight into the mechanics of the independently operated fault-tolerant processes. It is a computer model of a dual-redundant, fault-tolerant system. Redundant-element cycle times, initial skews, control-system parameters, and more may be varied by the operator. The simulator outputs as an x-y plot the current state of the input (Input) and plant (System Output), the values produced by the redundant control elements (Output A, Output B) and the difference between the outputs of the two elements (Error A-B) versus time for each run. Currently, two simple voting algorithms have been modeled: (1) Voter Output equals Channel A Output, and (2) Voter Output equals the average of the current values of Channel A and Channel B outputs.

The two models of a simple redundant control system run on the simulator are displayed in Fig. 4. The plant being controlled in this example is

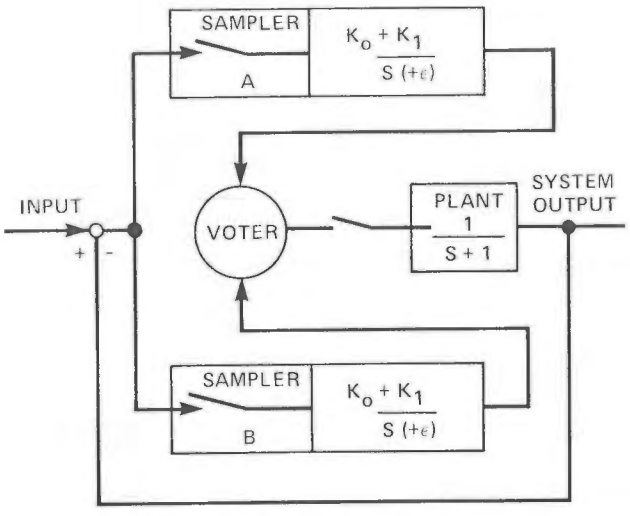


Fig. 4 Fault-tolerant system using integral control.

represented by a $1/S + 1$ term. A variety of waveforms may be input to the system. The control system is built as a unity-gain feedback system with two redundant controllers and one voter. Each controller consists of two control elements in the forward loop. The first is a simple, constant-gain term. The second may be a free integration term ($1/S$) or a lag term ($1/S + \varepsilon$) as determined by the operator. Plots of system responses to varying parameters of loop times, skews, voting algorithms, and input waveshapes were made. The performance of the control elements using free integrators is compared to the performance of the control elements which use lag elements (note that this controller is asymptotically stable). In this control system, channel divergence, which results from the redundant channels responding to different input data, will disappear with the passage of time. However, a price must be paid for the improvement in performance. The steady-state error is not zero. Typical values for K_0, K_1, and ε are, respectively, 2.1, 0.2, and 0.01 or 2.8, 4.0, and 0.1. This will leave the system with a 0.5% to 2.0% steady-state error. This steady-state error is sufficiently small that it is within the overall accuracy of most flight systems. Additionally, new control-system designs can be structured to compensate for the error.

Plots are presented to show the response of the two types of control systems to various inputs. These plots demonstrate the effect which sample skew and sample rate, and bias, scale, and nonlinearities of the sampling interface have on channel divergence. The response of the integral control system to a step input is shown in Fig. 5.

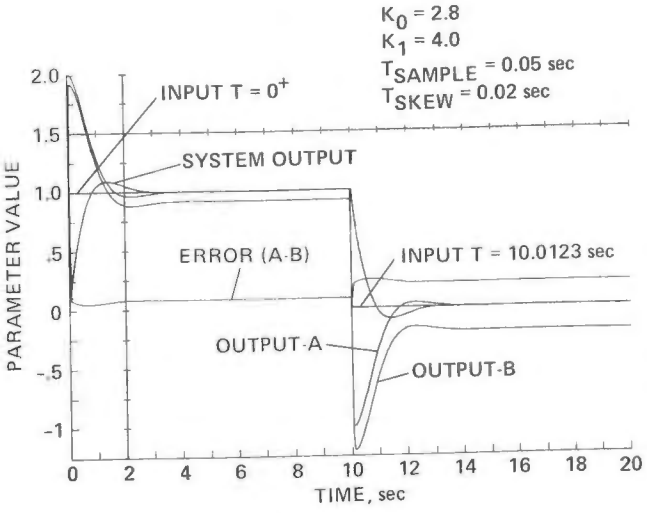


Fig. 5 Integral system (Fig. 4) step response.

The voting philosophy adopted here is "choose Channel A." Channel A sees the step first and produces an output for the plant. The state value for the plant, which has started to move, decreases the magnitude of the error signal seen by Channel B when it samples the variable 20 ms later. Because Channel B always samples 20 ms after Channel A (their clocks are highly accurate and they are running the same application program), Channel B consistently sees smaller values for the error signal during most of the transient response time. When the system has achieved its steady state, a constant channel divergence of about 15% of the step size has been introduced into the system. At 10.0123 sec, the value of the input is returned to zero. The relationship of sample time to system dynamics is important. If the input in Fig. 5 had been returned to zero at exactly 10.0000 sec, the steady-state channel divergence generated by the step input would go to zero. However, using the scenario of Fig. 5, bringing the input back to zero causes an increase in the steady-state channel divergence to 30% of the amplitude of the original step. Notice that the system has returned to its original null state, but the output values of the redundant channels have become highly divergent. The steady-state channel divergence is due solely to the presence of the integrator term (4.0/S) "remembering" the different sequence of input events as seen by each channel.

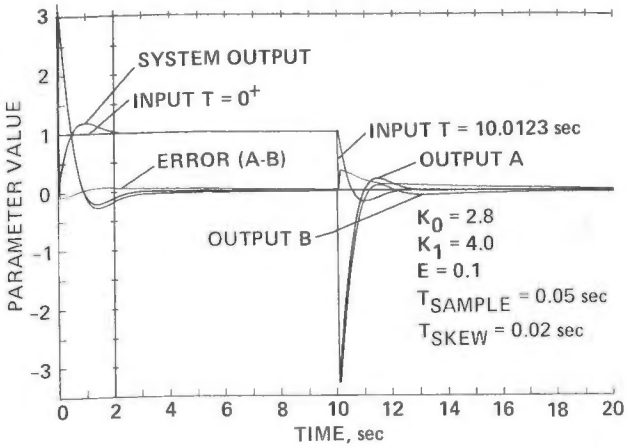


Fig. 6 Asymptotic system step response (Fig. 5).

Now, compare the response of Fig. 5 to that of Fig. 6. Exactly the same conditions and sequence of events were established for this simulation, except that the integrator has been replaced by a $(4.0/S + 0.1)$ term. The response of this term is slightly slower than that of the integrator. The steady-state gain has decreased from infinity to 40.0. But notice the significant effect this slight modification to the integration term has made. The channel divergence generated by the asymptotically stable controller at Time = 0.0+ is much less than that generated by the integral control system. Further, the channel divergence decays to zero as Channel B "forgets" the different sequence of input events and begins to operate solely on the currently available information. The response of the plant (system output) to the two controllers (integral versus asymptotic) is virtually indistinguishable. At 20.0 sec, the asymptotically stable redundant controllers have reduced the channel divergence to 0.0% of the value of the step (versus 30% for the integral control system). The voter must tolerate instantaneous channel divergence which may be very large (three units for the first 20 ms after the advent of the step in Fig. 6). The average channel divergence for asymptotically stable redundant controllers, however, can be made very small.

Increased Sampling Rate. As stated earlier, the channel divergence which can be expected from independently operated redundant elements is very heavily dependent upon the dynamics of the input relative to the sampling rates of the redundant

elements if the samples are not taken concurrently. This is brought forth in Figs. 7 and 8. In the former, the sampling rate of both redundant elements is 50 ms with Channel B's sampling time skewed 20 ms after Channel A. The voter algorithm is "Choose A." A sinusoidal input is injected into the system. The dynamics of the input and plant are sufficient to cause the channel divergence to oscillate about zero with a peak magnitude of about 0.25 unit. Notice that as the rate of change of the input decreases, the value of the channel divergence decreases, thus clearly demonstrating the sensitivity of channel divergence to input dynamics. The simulation shown in Fig. 8 is equivalent to that of Fig. 7 except the sampling period of the redundant elements has been decreased to 10 ms. Notice that decreasing the sampling period to 10 ms decreased the channel divergence to 20% of its original value. This demonstrates that the channel divergence is also sensitive to the sampling rate.

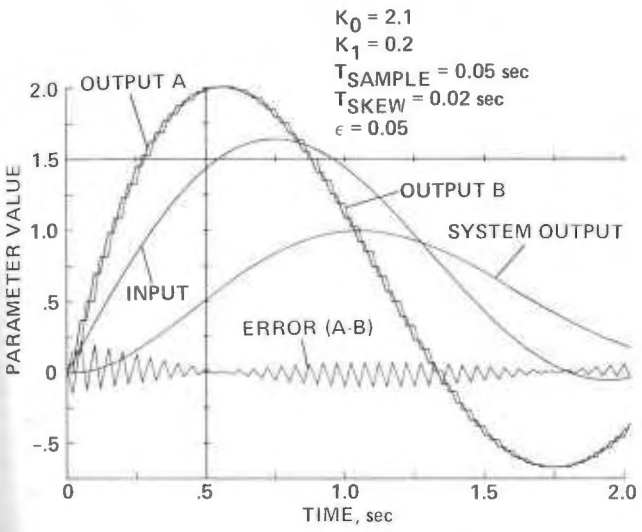


Fig. 7 Asymptotic system—sine response with slow sampling rate.

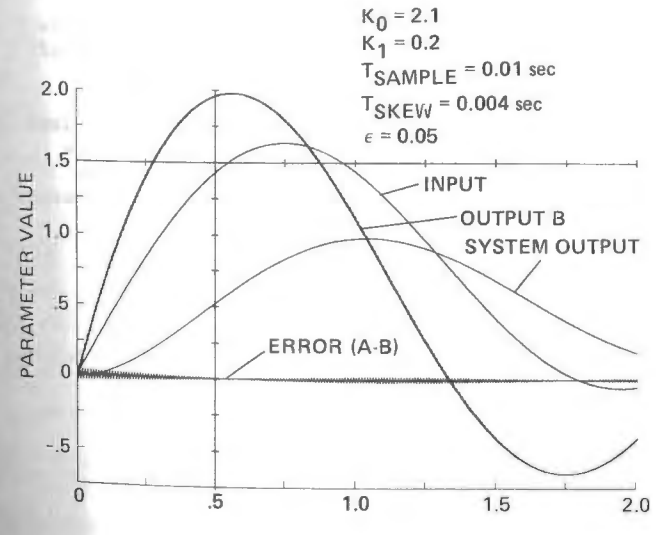


Fig. 8 Asymptotic system—sine response with fast sampling rate.

Control Time. The time at which state variables are sampled is of utmost importance to the amount of channel divergence which will be experienced. To a lesser extent, the relative variation in time-keeping between elements will also affect channel divergence. Obviously, if the redundant channels can be made to sample at nearly the same time, channel divergence will be created only as a result of the variations in the sampling interfaces discussed above. This can be accomplished without resort to cross communication between channels. Each redundant element can be given access to a redundant clock whose time data enters the system in exactly the same way as any sensed variable. If the redundant elements are programmed such that state variables are to be sampled at each 50 ms boundary and if the latency of the redundant time data is small, it can be shown that the sampling time of the redundant elements can be made to approximately converge. The coincidence of the sampling would depend solely on the uncertainty in each element of the actual value of time. Controlling time will eliminate the drift of time between computers for long-duration missions. The application programs would return to being frame-based. However, the redundant elements are still independent entities. Channel divergence is significantly decreased and the use of asymptotically stable controllers is still required.

Experimental Tests

A working example of a UFTCS system has been configured and tested at NASA Ames Research Center. It is a microprocessor-based, quadraplex, redundant control system configured with independently operated redundant elements. Sampling times were deliberately made highly asynchronous with cycle times of the application program being data driven and varying from 35 to 52 ms. Total real time control of all four axes of a UH-1H helicopter is provided both in simulation and flight test.

Manned Flight Simulations

Laboratory and manned simulation testing of the device have proven highly successful. Several hundred simulated "flight" hours have been logged to date. The four channels have demonstrated acceptable, votable, levels of channel divergence. Figure 9 shows an expanded view of the four redundant

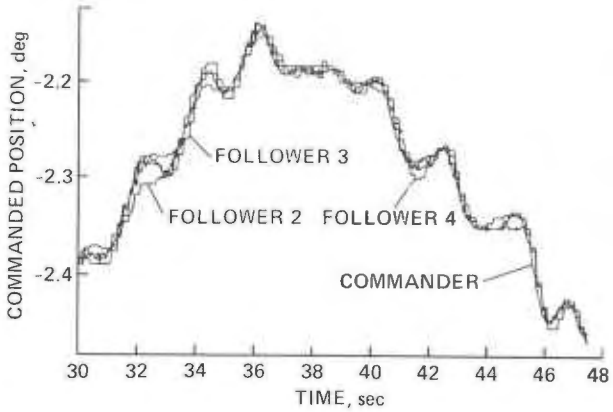


Fig. 9 UH-1H pedal position—midvalue select voter.

controls for the pedal axis during 18 sec of a 500 sec flight in the manned flight simulator. The voting algorithm employed was a version of midvalue select. The control theory employed was that described by Meyer and Cicolani in Ref. 6. As can be seen, the redundant control values generated remain well within 0.05° of each other. Over the entire flight, and other flights, the redundantly generated control values for each of the four axes remain within 0.1° of being consistent.

Calculated Reliability of the Test System

A detailed reliability analysis of the UFTCS structure was conducted by Curry/Frey of Search Technology and VanderVelde of MIT. The hardware developed for the test system was used as a model, and the study considered potential configurations combining quadruplet, pentad, and hexad members. Voting philosophies considered that simple voters would allow the system to fail to two operating, and "smart" voters could allow the system to fail to one remaining operative.

The results of the analysis show that the probability of system failure for the first 10 hr of a helicopter flight using a pentad UFTCS (excluding the unreliability of the actual sensors and the actuation hydraulics) with simple voters is 8.6×10^{-11}. The system is based on microcomputer technology with the total volume of a pentad UFTCS controller being less than a small suitcase, with the elements dispersed throughout the controlled environment. In the air-transport environment, under the same conditions, the probability of system failure is 2.9×10^{-11}. In space applications, such as the Space Station, the 2 week unattended probability of failure is 2.4×10^{-7}. As the system contains virtually no common elements and is not subject to latent common design failure, the delivered reliability of these systems should approach the calculated figures.

Conclusions

Fault-tolerant control systems can be configured from independent, autonomous, redundant elements. Made from dissimilar hardware, processing different versions of the application programs, and controlling system-fault status through the use of static redundancy management, these systems assume the property of ultrareliability. Systems so structured do not possess sources of latent common-design faults, and their demonstrated field reliability will approach predicted system reliability. Channel divergence normally encountered in fault-tolerant systems using independent redundant elements is caused by sampling and time differences between channels and the use of integral control theory. Based on experimental evidence, channel divergence can be decreased to levels which are readily voted through (1) the use of asymptotically stable control laws, (2) sampling rates which are adequately rapid relative to the frequency response of the total system, and (3) forcing sampling times to converge.

References

(1) W. Dunn, J. Johnson, and G. Meyer, "A Fault Tolerant Distributed Microcomputer Structure for Aircraft Navigation and Control," Fourteenth Asilomar Conf. Circ. Syst. Comp., IEEE Cat. No. 80CH1625-3, Nov. 1980

(2) E. Yeh, "Applied Computation Theory, Analysis and Modeling," Prentice Hall, pp. 352-359, 1976

(3) J. Wensley, L. Lamport, J. Goldburg, M. Green, K. Levitt, P. Melliar-Smith, R. Shostak, and C. Weinstock, "SIFT: Design and Analysis of a Fault Tolerant Computer for Aircraft Control," Proc. IEEE, Vol. 66, No. 10, pp. 1240-1255, Oct. 1978

(4) A. Hopkins, Jr., T. Smith, III, and J. Lala, "FTMP—A Highly Reliable Fault Tolerant Multiprocessor for Aircraft," Proc. IEEE, Vol. 66, No. 10, pp. 1221-1239, Oct. 1978

(5) G. Meyer, L. Hunt, and R. Su, "Design of a Helicopter Autopilot by Means of Linearizing Transforms," Advances Guid. Cont. Syst., Advisory Group for Aerospace Res. & Dev., pp. 4-1 to 4-11, AGARD-CP-321, 1982

(6) G. Meyer and L. Cicolani, "Application of Nonlinear System Inverses to Automatic Flight Control Design-System Concepts and Flight Evaluations," Theory Appl. Optimal Cont. Aerospace Syst., Advisory Group for Aerospace Res. & Dev., pp. 10-1 to 10-29, AGARD-AG-251, 1980

(7) R. Luppold, E. Gai, and B. Walker, "Effects of Redundancy Management on Reliability Modeling," Proc. 1984 Amer. Cont. Conf., pp. 1763-1770, June 1984

(8) D. Mackall, "AFTI-F16 Digital Flight Controls System Experience," First Annual NASA Aircraft Controls Workshop, NASA Langley Res. Cen., Oct. 1983

(9) S. Osder, "Generic Faults and Design Solutions for Flight-Critical Systems," In: Guidance and Control Conference, Coll. Tech. Papers (A82-38926 19018), New York, Amer. Inst. Aeron. and Astron., pp. 509-518, Aug. 1982

SESSION 9 SIGNAL PROCESSING

Chairmen:

Dr. Gerald Palatucci
Naval Air Development Center

Dr. Bjorn Bjerede
Linkabit Corp.

This session deals with both the algorithmic and computational aspects of communications signal processing, including techniques for filtering and frequency hopping.

SIGNAL PROCESSING USING DIGITAL LATTICE FILTERS

84-2651

Leonard Chin

Naval Air Development Center
Warminster, Pennsylvania 18974

Abstract

The "order recursive" feature of a digital lattice filter is discussed relative to the solution of a linear prediction problem using Levinson's algorithm in the determination of the order and coefficients of the autoregressive model. It is shown that the number of filter sections can be added or deleted without recomputing the coefficients of the remaining sections. Other benefits such as numerically stable, computationally efficient, etc. can also be derived from the lattice structure.

Introduction

The term "digital filter" is commonly used when referring to the counterpart of an analog filter having one of the ladder structures. In the past 20 years, many books and articles have been written on the subject of digital filters of the ladder form. However the interest in digital filters of the lattice form was not emerged until 1971 when Fettweis [1] introduced the "wave digital filter." Subsequently the study of properties of this filter was followed by investigators [2-11] who have shown that lattice digital filters have very low quantization noise and they are relatively insensitive to perturbations made to the multiplier coefficients of the filter. Currently the study of implementation of lattice filter was conducted by others [12-19] who have developed algorithms for many applications including transient analysis, spectral analysis, adaptive processing of time series, adaptive prediction and control, equalization of digital communication channels, etc.

Generally speaking, the use of lattice filters to process a certain class of stochastic signals (wide sense stationary) has a number of advantages over the ladder filters. The purpose of this paper is to demonstrate one of the many advantages. Namely the "order recursive" feature associated with Levinson's algorithm in the determination of the order and coefficients of the autoregressive model relative to the linear prediction problem.

The Linear Prediction Problem

Let x(k) be a digital signal that is predictable from linear combinations of past inputs and outputs of a system described as follows:

$$x(k) = -\sum_{i=1}^{N} a_i\, x(k-i) + GW(k) + \sum_{j=1}^{M} b_j\, w(k-j) \qquad (1)$$

where w(k) and x(k) are inputs and outputs respectively; a_i for $1 \le i \le N$, b_j for $1 \le j \le M$ and the gain G are parameters of the hypothesized system. Equation (1) can be specified in the frequency domain by taking the Z-transform on both sides of equation (1).

Define: $Y(z) = \sum_{n=-\infty}^{\infty} y(n) z^{-n}$; where Y(z) is the Z-transform of y(n).

$H(z) = \frac{X(z)}{W(z)}$; where H(z) is the transfer function of the system.

Then from equation (1):

$$H(z) = G\, \frac{1 + \sum_{j=1}^{M} b_j\, z^{-j}}{1 + \sum_{i=1}^{N} a_i\, z^{-i}} \qquad (2)$$

In equation (2), the roots of the numerator polynominal and the roots of the denominator polynominal are the zeros and poles of the model (equation (1)) respectively. The pole-zero model specified in equation (2) is called ARMA (autoregressive moving average) and the two special cases are:

1. MA (moving average) - all zero model
   $a_i = 0$ for $1 \le i \le N$

2. AR (autoregressive) - all pole model
   $b_j = 0$ for $1 \le j \le M$

For the purpose of demonstrating the unique properties of lattice filters relative to linear prediction problems, consider only special case 2, i.e., equation (1) is reduced to

$$x(k) + \sum_{i=1}^{N} a_i\, x(k-i) = Gw(k) \qquad (3)$$

Problem statement: Given measurement x(k), a stationary process, find $\hat{a}_i$ such that the mean square error (e) is minimized. e is defined as

$$e(k) = x(k) + \sum_{i=1}^{N} \hat{a}_i\, x(k-i) \qquad (4)$$

Applying the Orthogonality Principal [20] minimizing $E[e^2(k)]$ is equivalent to set

$$E\left[\left\{ x(k) + \sum_{i=1}^{N} \hat{a}_i\, x(k-i) \right\} x(k-j)\right] = 0 \qquad (5)$$

which yields

$$\sum_{i=1}^{N} \hat{a}_i R_x(j-i) = -R_x(j), \quad j=1,2,\ldots,N \tag{6}$$

in which $R_x(\cdot)$ is the autocorrelation function. Equation (6) is the well known Yule-Walker equation which can be written in matrix form:

$$\begin{bmatrix} R_x(0) & R_x(1) & \cdots & R_x(N-1) \\ R_x(1) & R_x(0) & \cdots & R_x(N-2) \\ \cdot & & \cdot & \cdot \\ \cdot & & \cdot & \cdot \\ \cdot & & \cdot & \cdot \\ R_x(N-1) & \cdots & & R_x(0) \end{bmatrix} \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \cdot \\ \cdot \\ \cdot \\ \hat{a}_N \end{bmatrix} = - \begin{bmatrix} R_x(1) \\ R_x(2) \\ \cdot \\ \cdot \\ \cdot \\ R_x(N) \end{bmatrix} \tag{7}$$

using short-hand notations, equation (7) can be written as

$$\hat{a}^N = -(R^N)^{-1} P^N \tag{8}$$

in which R^N is a Toeplitz matrix.

Mathematically the solution of equation (8) is straightforward. In fact $\hat{a}_i$ will form a stable set, i.e., the characteristic equation

$$1 + \sum_{i=1}^{N} \hat{a}_i z^{-i} = 0 \tag{9}$$

will have all zeros inside the unit circle in the z-plane. Computationally, $R_x(k)$ will be approximated by

$$R_x(k) \simeq \frac{1}{L} \sum_{n=0}^{L-|k|-1} x(n)\, x(n+k) \tag{10}$$

where L is the length of the data sequence. However, solving equation (8) in a straightforward manner will experience the following disadvantages.

1. A matrix inversion is required.
2. $R_x(j-i)$ must be computed in a batch.
3. N (the order of the model) must be known a priori.

The last disadvantage is most serious in terms of computation burden. Specifically in order to find N such that the mean square error (equation (4)) is minimum, equation (8) has to be solved repeatedly for various values of N. To this end, one is searching for a systematic and recursive method of determining N as well as estimating the coefficients (a_i) of the filter.

The Levinson Algorithm and The Digital Lattice Filter

The Levinson Algorithm was specially designed to solve equation (8) in such a manner that it will avoid all three disadvantages cited above. The purpose of this section is to show that the Levinson Algorithm can be conventionally implemented using digital filters of a lattice structure.

Assume m to be the order of the model, then

$$\hat{a}^m = (\hat{a}_1^m \;\; \hat{a}_2^m \;\; \cdots \;\; \hat{a}_m^m)^T \tag{11}$$

where the superscript m denotes the order of the model. In terms of the m^{th} order, equation (7) becomes

$$\begin{bmatrix} R_x(0) & \cdots & R_x(m-1) \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ R_x(m-1) & & R_x(0) \end{bmatrix} \begin{bmatrix} \hat{a}_1^m \\ \cdot \\ \cdot \\ \cdot \\ \hat{a}_m^m \end{bmatrix} = - \begin{bmatrix} R_x(1) \\ \cdot \\ \cdot \\ \cdot \\ R_x(m) \end{bmatrix} \tag{12A}$$

Since $[R^m]$ the correlation matrix is Toeplitz, equation (12A) can also be written as follows:

$$\begin{bmatrix} R_x(0) & \cdots & R_x(m-1) \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ R_x(m-1) & & R_x(0) \end{bmatrix} \begin{bmatrix} \hat{a}_m^m \\ \cdot \\ \cdot \\ \cdot \\ \hat{a}_1^m \end{bmatrix} = - \begin{bmatrix} R_x(m) \\ \cdot \\ \cdot \\ \cdot \\ R_x(1) \end{bmatrix} \tag{12B}$$

In order to establish a recursive relationship between $\hat{a}^m$ and $\hat{a}^{m+1}$, let m in equation (12A) increase by 1.

$$\left[\begin{array}{ccc|c} R_x(0) & \cdots & R_x(m-1) & R_x(m) \\ \cdot & & \cdot & \\ \cdot & & \cdot & \\ \cdot & & \cdot & \\ R_x(m-1) & & R_x(0) & R_x(1) \\ \hline R_x(m) & \cdots & R_x(1) & R_x(0) \end{array}\right] \left[\begin{array}{c} \hat{a}_1^{m+1} \\ \cdot \\ \cdot \\ \cdot \\ \hat{a}_m^{m+1} \\ \hline \hat{a}_{m+1}^{m+1} \end{array}\right] = - \left[\begin{array}{c} R_x(1) \\ \cdot \\ \cdot \\ \cdot \\ R_x(m) \\ \hline R_x(m+1) \end{array}\right] \tag{13}$$

The desired recursive relationship is obtained by rewriting equation (13) using the indicated partitions.

$$\begin{bmatrix} R_x(0) & \cdots & R_x(m-1) \\ \vdots & & \vdots \\ R_x(m-1) & & R_x(0) \end{bmatrix} \begin{bmatrix} \hat{a}_1^{m+1} \\ \vdots \\ \hat{a}_m^{m+1} \end{bmatrix} + \hat{a}_{m+1}^{m+1} \begin{bmatrix} R_x(m) \\ \vdots \\ R_x(1) \end{bmatrix} =$$

$$- \begin{bmatrix} R_x(1) \\ \vdots \\ R_x(m) \end{bmatrix} \qquad (14A)$$

$$\sum_{i=1}^{M} \hat{a}_i^{m+1} R_x(m+1-i) + \hat{a}_{m+1}^{m+1} R_x(0) = -R_x(m+1) \qquad (14B)$$

Premultiply euation (14A) by $[R^m]^{-1}$, then substitute equations (12A) and (12B) into it, yields the following result:

$$\begin{bmatrix} \hat{a}_1^{m+1} \\ \vdots \\ \hat{a}_m^{m+1} \end{bmatrix} - \hat{a}_{m+1}^{m+1} \begin{bmatrix} \hat{a}_m^m \\ \vdots \\ \hat{a}_1^m \end{bmatrix} = \begin{bmatrix} \hat{a}_1^m \\ \vdots \\ \hat{a}_m^m \end{bmatrix} \qquad (15)$$

Define $\hat{a}_o^m = \hat{a}_o^{m+1} = 1$ and $\hat{a}_{m+1}^{m+1} = p_{m+1}$, the following set of recursive equations obtained from equation (15) will be used to solve for $\hat{a}_i^{m+1}$.

$$\hat{a}_o^{m+1} = \hat{a}_o^m \qquad (16A)$$

$$\hat{a}_i^{m+1} = \hat{a}_i^m + p_{m+1} \hat{a}_{m+1-i}^m \;, \; i=1,2,\ldots,m \qquad (16B)$$

p_{m+1} is called the Partial Correlation Coefficients. In order to solve equation (16) recursively, these coefficients must be determined first. To this end, we take the digital filtering approach. Specifically let's multiply equation (16B) by z^{-i} (z^{-1} is a unit delay) and sum over index i.

$$\sum_{i=0}^{m+1} \hat{a}_i^{m+1} z^{-i} = \sum_{i=0}^{m} \hat{a}_i^m z^{-i} + \left[p_{m+1} z^{-i} \right].$$

$$\left[\sum_{i=1}^{m} \hat{a}_{m+1-i}^m z^{-(i-1)} + z^{-m} \right] \qquad (17)$$

Define:

$$F_m(z^{-1}) = \sum_{i=0}^{m} \hat{a}_i^m z^{-i} \qquad (18)$$

$$G_m(z^{-1}) = z^{-1} \left[\sum_{i=1}^{m} \hat{a}_{m+1-i}^m Z^{-(i-1)} + Z^{-m} \right] \qquad (19)$$

Equation (17) can be written as

$$F_{m+1}(z^{-1}) = F_m(z^{-1}) + p_{m+1} G_m(z^{-1}) \qquad (20)$$

Equation (20) is called the "forward equation," it alone is insufficient to solve for P_{m+1}. An additional relationship derived from equation (16B), replacing i by (m+1-i), is called the "backward equation":

$$z\, G_{m+1}(z^{-1}) = G_m(z^{-1}) + P_{m+1} F_m(z^{-1}) \qquad (21)$$

A lattice structure can be used to implement equations (20) and (21) which yields P_{m+1}. Then knowing p_{m+1}, we can use equation (16) to solve for $\hat{a}_i^{m+1}$. However in order to solve for p_{m+1}, $F_m(z^{-1})$ and $G_m(z^{-1})$ which are functions of $\hat{a}_i^m$, have to be known. A more practical way of computing p_{m+1} recursively without requirng the explicit form of F_m and G_m is to make use of the available data. Specifically the error associated with the estimate of the observed data.

Let the "forward" estimate be

$$\hat{x}(k) = - \sum_{i=1}^{m} \hat{g}_i^m x(k-i) \qquad (22)$$

Define:

$$e_f^m(k) = x(k) - \hat{x}(k) \qquad (23)$$

Combining equations (22) and (23) yields

$$e_f^m(k) = \sum_{i=0}^{m} \hat{g}_i^m \, x(k-i) \tag{24}$$

which is the error of the forward estimate. Similarly, let the "backward" estimate be

$$\hat{x}(k-m-1) = \sum_{i=0}^{m-1} \hat{h}_i^m \, x(k-m+i) \tag{25}$$

where

$$\hat{h}_i^m = \hat{a}_{m-i+1}^m \, , \; i = 1, 2, \ldots, m \tag{26}$$

Using a similar definition as equation (23), the error of the backward estimate is

$$e_b^m(k) = x(k-m) + \sum_{i=0}^{m-1} \hat{h}_i^m \, x(k-m+i) \tag{27}$$

Define:

$$E_f^m(z) = \sum_{k=0}^{m} e_f^m(k) \, z^{-k} \tag{28}$$

Combining equations (24), (18) and (28) yields

$$E_f^m(z) = F_n(z^{-1}) \, X(z) \tag{29}$$

where

$$X(z) = \sum_{i=0}^{m} x(i) \, z^{-i} \tag{30}$$

Similarly, define

$$E_b^m(z) = \sum_{k=0}^{m} e_b^m(k) \, z^{-k} \tag{31}$$

Combining equations (27), (19) and (31) yields

$$E_b^m(z) = z \, G_n(z^{-1}) \, X(z) \tag{32}$$

Study of equations (29) and (32) shows that $E_f^m(z)$ and $E_b^m(z)$ propagate through the lattice network in the same way as $F_m(z^{-1})$ and $G_m(z^{-1})$ given by equations (20) and (21). In order to show this, multiply equations (20) and (21) by X(z), make use of the results given by equations (29) and (32), we obtain the following pair of desirable equations.

$$e_f^{m+1}(k) = e_f^m(k) + p_{m+1} \, e_b^m(k-1) \tag{33A}$$

$$e_b^{m+1}(k) = e_b^m(k-1) + p_{m+1} \, e_f^m(k) \tag{33B}$$

which can be implemented using a lattice structue given in Figure 1.

It remains to show that p_{m+1} is computed based on the knowledge of the forward and backward errors. Specifically p_{m+1} is estimated by minimizing either the mean square value of $e_f(k)$ or $e_b(k)$. A straightforward minimization procedure using equations (33A) and (33B) yields

$$\hat{p}_{m+1}^f = - \frac{E\left\{ e_f^m(k) \, e_b^m(k-1) \right\}}{E\left\{ [\, e_b^m(k-1) \,]^2 \right\}} \tag{34A}$$

$$\hat{p}_{m+1}^b = - \frac{E\left\{ e_f^m(k) \, e_b^m(k-1) \right\}}{E\left\{ [\, e_f^m(k) \,]^2 \right\}} \tag{34B}$$

where $\hat{p}_{m+1}^f$ and $\hat{p}_{m+1}^b$ are the optimal estimates of the forward and backward partial corelation coefficients. Combining equations (34A) and (34B) yields

$$\hat{p}_{m+1} = \frac{-2E\left\{ e^m(k) \, e^m(k-1) \right\}}{E\left\{ [\, e_b^m(k-1) \,]^2 \right\} + E\left\{ [\, e_f^m(k) \,]^2 \right\}} \tag{35}$$

Equation (35) together with equations (16A) and (16B) constitute the Levinson algorithm with initial conditions given by equations (24) and (27). The iterative process continues until $(J_{m+1} - J_m) =$ constant, where J_m is the mean square error of $e_f^m(k)$ or $e_b^m(k)$ for k = 0, 1, ... L-1; L is the number of data points. When the residual error $(J_{m+1} - J_m)$ reaches a steady state value. N is determined to be equal to m.

Conclusion

It was shown in this paper that for a class of wide sense stationary signals, a digital lattice filter can be built, based on Levinson's algorithm, to solve a linear predictive problem; specifically determination of the order and coefficients of the AR model. As it can be seen from figure 1, the special feature of "order recursive" was demonstrated. That is, the number of filter sections can be added or deleted without recomputing the coefficients of the remaining sections. Another feature is that the computation of these coefficients associated with an NXN covariance matrix requires only N^2 operations. If direct matrix inversion methods are used, N^3 operations would be required. The computation is relatively insensitive to quantization errors as compared to the ladder filters because the coefficients are bounded by the magnitude of one. Also because the latice structure is a modular pipeline construction, it facilitates fixed point arithmetic implementation. These advantages, together with the convenience in filter initializing, make the digital lattice filter a very practical and efficient method for many applications.

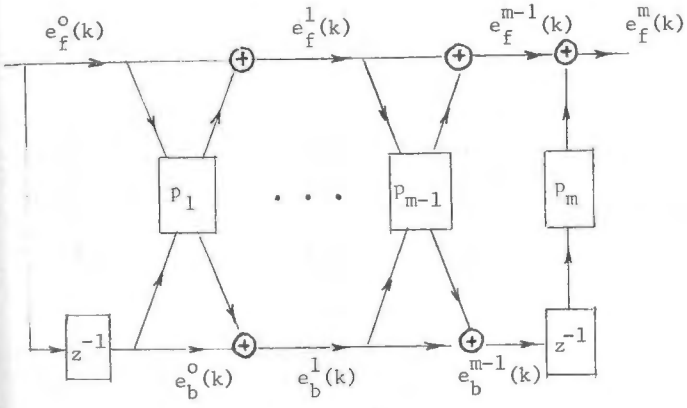


Figure 1. A Lattice Structure Implementation of the Levinson Algorithm

References

(1) Fetweis, A. "Some Principles of Designing Digital Fitlers Imitating Classical Filter Structures" IEEE Trans. Circuit Theory, Vol. CT-18, pp 314-316, Mar 1971.

(2) Gray, A. H. Jr., and Markel, J. D. "Digital Lattice and Ladder Fitler Synthesis" IEEE Trans. Audio Electro-acoustics, Vol. Au-21, No. 6, pp 491-500, 1973.

(3) Fettweis, A. "On Sensitivity and Roundoff Noise In Wave Digital Filters" IEEE Trans. Acoust. Speech Signal Process, Vol. ASSP-22, No. 5, pp 383-384, Oct 1974 .

(4) Fettweis, A., Levin, H., and Sedlmeyer, A., "Wave Digital Lattice Filters" Int. J. Circuit Theory Appl. Vol. 2, pp 203-211, 1974.

(5) Fettweis, A., Levin, H., and Sedlmeyer, A., "A Normalized Ditital Filter Structure" IEEE Trans. Acoust., Spech Signal Process., Vol. ASSP-23, No. 3, pp 268-277, 1975.

(6) Ku, N. H. and Ng, S. M. "Floating-Point Coefficient Sensitivity and Roundoff Noise of Recursive Digital Filters Realized in Ladder Structures" IEEE Trans. Circuits Syst., Vol. CAS-22, No. 12, pp. 927-936, Dec 1975.

(7) Markel, J. D. and Gray, A. H. Jr. "Roundoff Noise Characteristics of a Class of Orthogonal Polynominal Structues" IEEE Tans. Acoust., Speech Signal Process., Vol. ASSP-23, pp 473-486, 1975.

(8) Markel, J. D. and Gray, A. H. Jr. "Fixed-Point Implementation Algorithms for a Class of Orthogonal Polynominal Filter Structures" IEEE Trans. Acoust., Speech Signal Process., Vol. ASSP-23, pp 486-494, 1975.

(9) Wegener, W. "On the Design of Wave Digital Lattice Filters with Short Coefficient Word Length and Optimal Dynamic Range" IEEE Trans. Circuits. Syst., Vol. CAS-25, No. 12, p 1091, Dec 1978.

(10) Gray, A. H. Jr. "Passive Cascade Lattice Digital Filters" IEEE Trans. Circuits Syst., Vol. CAS-27, No. 5, pp 337-344, May 1980.

(11) Chu, P. L. and Messerschmitt, D. G. "Zero Sensitivity of the Digital Lattice Filter" Proc. IEEE Int. Conf. Acoust., Speech Signal Process., pp 89-93, 1980.

(12) VanBlaricum, M. L. and Mittra, R. "Problems and Solutions Associated with Prony's Method for Processing Transitent Data" IEEE Trans. Antennas Propagat., Vol. AP-26, pp 174-182, Jan 1978.

(13) Kay, S. M. and Marple, S. L. Jr., "Spectrum Analysis - A Modern Perspective" IEEE Proc. Vol. 69, No. 11, pp 1380-1419, Nov 1981.

(14) Friedlander, B. "Lattice Filters for Adaptive Processing" IEEE Proc. Vol. 70, No. 8, pp 829-867, Aug 1982.

(15) Friedlander, B. "Lattice Methods for Spectral Estimations" IEEE Proc. Vol 70, No. 9, pp 990-1017, Sep 1982.

(16) Friedlander, B. "A Lattice Algorithm for Factoring the Spectram of a Moving Average Process" IEEE Trans. Autom. Contr., Vol. AC-28, No. 11, pp 1051-1055, Nov 1983.

(17) Makhoul, J. and Viswanathan, R. "Adaptive Lattice Methods for Linear Predication" Proc. IEEE Int. Conf. Acoustc. Speech Signal Process., pp 83-86, 1978.

(18) Proukis, J. G. Digial Communications, McGraw Hill, New York, 1983.

(19) Goodwin, G. C. and Sin, K. S. Adaptive Filtering, Prediction and Control, Prentice Hall, New Jersey, 1984.

(20) Papoulis, A. Probabilty, Random Variables, and Stochastic Processes, 2nd edition, McGraw Hill, New York, 1984.

APPLICATION OF DIGITAL SIGNAL PROCESSING TO A LOW DATA RATE COMMUNICATIONS RECEIVER

Robert G. Henderson
Associate Department Head
Naval Systems Engineering Division
The MITRE Corporation
McLean, Virginia

Pierre Lafrance*
Group Leader
Naval Systems Engineering Division
The MITRE Corporation
McLean, Virginia

Abstract

This paper explores the application of digital signal processing techniques to replace and even improve receiver functions which are conventionally performed by analog circuitry. All signals downstream of the last IF stage are processed digitally. The IF processing system under consideration consists of several components. A pre-processor samples the 22 kilohertz wide IF. The resulting sequence is then digitally Hilbert transformed to produce the in-phase and quadrature components of an analytic signal. Several cascaded stages of filtering and decimation follow to produce a baseband signal with a bandwidth of 7 kilohertz and within which the information is known to reside, but with an a priori unknown doppler shift. The second stage of the system accomplishes further bandwidth compression by processing a spectral representation of the signal via a real time Fourier transform followed by decision logic. Performance is analyzed and design issues are discussed.

Introduction

This paper describes a preliminary design for a digital signal processor which will be used to enhance the performance of a particular low data rate communication receiver. The receiver in question must detect the transmission of a pulsed tone on one of two pseudo-random frequencies in the UHF band. The tone duration is 9 msec so that the optimal detector bandwidth should be approximately 111 Hz. However, the systmem must operate in an air-to-air environment involving high speed, high performance aircraft so that a doppler uncertainty in the transmitted frequency of +/- 1750 Hz must be dealt with. In addition, the system must operate using a host UHF receiver which has a final IF bandwidth of 22 KHz. Thus the objective of the proposed digital signal processor is to provide a synthesized 111 Hz matched filter which can be tuned within a 3500 Hz doppler uncertainty window. The expected processing gain achieved by using a synthesized matched filter is 15 dB with respect to using a 3500 Hz IF (analog or digital).

This paper will present the preliminary design considerations, and some results of a theoretical performance analysis.

Description of DSP

The DSP is shown schematically in Fig. 1. For the purpose of this discussion the DSP will be considered to be composed of the following processing blocks: analog to digital conversion (A/D) and basebanding block; decimation

* Member IEEE

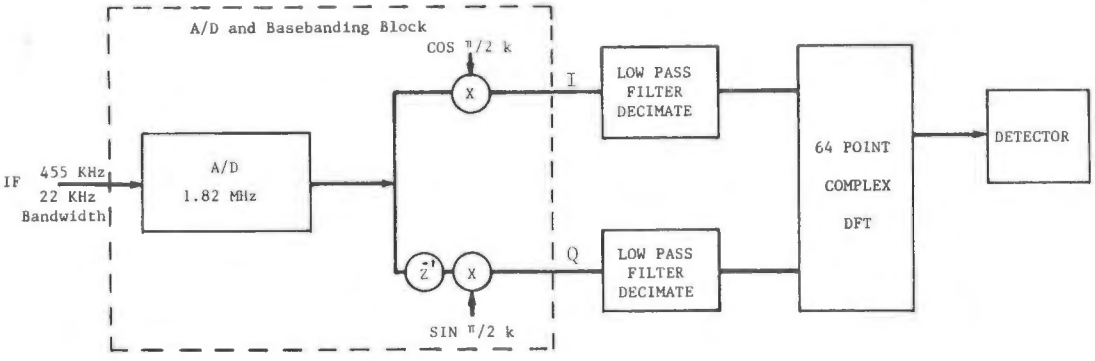


FIGURE 1

DSP PROCESSING BLOCKS

and filtering block; discrete Fourier transform (DFT) block; and the detection block. Each of these blocks and the considerations that led to their design will be discussed in turn.

A/D and Basebanding:

Several A/D schemes could be used to digitize the 455 KHz IF (22 KHz 3 dB Bandwidth) signal which must be processed. For reasons of processing speed and ease of implementation, a 1.82 MHz sampling rate was chosen. This is exactly four times the IF frequency and leads to considerable simplication in the translation to baseband of the digitized signal. An eight bit flash A/D converter will be used to provide 48 dB SNR at the output.

After digitization, quadrature demodulation is used to form inphase and quadrature phase channels at baseband (1). The inphase channel is obtained by multiplying each sample by a digital representation of the cosine of the IF frequency. However, since the IF frequency is exactly one quarter of the sampling frequency the cosine values are given by cos ($k\pi/2$) which can have the values of +1, 0, and -1 thus obviating the need for an actual multiplication. The quadrature phase channel is formed by first shifting the signal by $\pi/2$ and then multiplying by the sine of the IF frequency. Because of the selection of the sampling rate this is accomplished by a simple time delay and then multiplication by +1, 0, and -1.

Decimation and Filtering Block:

At this point the sampling rate is much higher than required by the signal which is contained within a 3.5 KHz bandwidth about the origin and a 3 dB noise bandwidth of 22 KHz. In addition of course, performing spectral analysis at this point would require a 16,000 point DFT! To make the DFT problem more manageable, decimation and filtering must first be carried out. Decimation will decrease the size requirement of the DFT while at the same time the filtering will reduce the noise bandwidth of the signal.

The decimation and filtering of the signal will be carried out in four steps, each step successively reducing the sample rate and the noise bandwidth. The process is done successively in order to reduce the number of multiplications required by the digital FIR filters. The process is shown in Fig. 2. The first step is to perform a decimation by 16, reducing the sampling rate to 113.75 KHz. A FIR low pass filter and decimation by 2 reduces the sampling rate to 56.87 KHz and the 50 dB noise bandwidth to approximately 28 KHz. The low pass filter and decimation operation is repeated twice more, as shown in Fig. 2 so that the final sampling rate is 7109 Hz, and the final noise bandwidth (50 dB) is 3.55 kHz. The final output from the decimation and filtering process consists of 64 samples.

Figure 2 also shows the filter order for each FIR and the resulting number of multiplies (2) that are required at each stage. Based on using a 200 nsec. multiplier-accumulator, the approximate processing times required at each filter stage are .8 msec, .8 msec and .2 msec respectively. Thus a total time of approximately 2 msec is required for the decimation and filtering block. If a single multiplier were used to perform all of the filtering in both the in-phase and

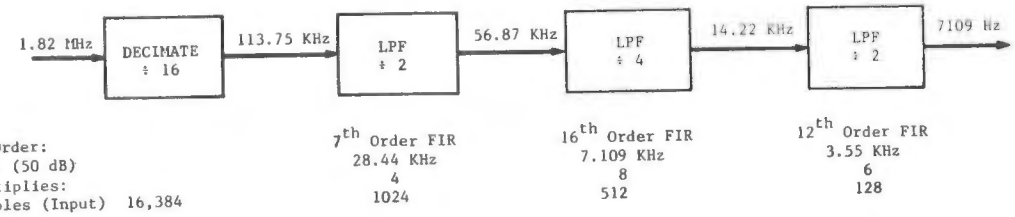


FIGURE 2

LOW PASS FILTER AND DECIMATE BLOCK

quadrature phase channels, approximately 4 msecs would be required.

Using an extended precision multiply/accumulator results in round-off noise of 53 dB below the signal after each multiply (3). With a total of 18 multiplies in the decimation and filtering process this amounts to a round-off SNR of about 40 dB.

DFT Block:

The decimated and filtered inphase and quadrature phase channels are input to a complex DFT for the purpose of providing a final filtering operation before detection of the desired signal. Since the goal is to produce as fine a resolution as possible, no windowing will be perfomed. In this case the resolution is given by the sample rate, Fs, divided by the number of samples or 111 Hz. The peak sidelobe in this case is -13 dB (4) so that two adjacent signals will only be resolved if their amplitudes differ by no more than 13 dB. Since the primary concern is to detect a single 111 Hz tone burst in the presence of noise the small amplitude difference is acceptable.

The hardware implementation of the DFT has not been decided as yet, however, special purpose hardware is available, such as the Texas Instruments TMS32010 which can perform a complex 64 point DFT in approximately .8 msec. (5).

Detection Block

The detection block will consist of a complex multiplication to square the outputs of the DFT and then decision logic to determine on which of two dwells (Mark or Space dwells) the 9 msec pulse was transmitted. Two decision logics have been considered: A single bit decision which is based on which of the two frequencies has the single largest output from the DFT; and a multibit technique which attempts to track the change in doppler offset and then make decisions based on examining only the appropriate DFT output for the Mark and Space frequencies. An analysis of the expected (theoretical) performance of these two approaches is outlined below and the results are compared to those of a true matched filter.

Analysis of Detection Alternatives

Two detection schemes are considered. The simplest involves a comparison of all squared DFT spectral amplitudes. Each of the Mark and Space dwells constitute M such amplitudes. A bit decision is made based on which (Mark or Space) dwell contains the largest spectral component. This approach has a relatively low implementation complexity.

An alternative approach, one which compensates for rather than tolerates doppler shifts, involves doppler acquisition and tracking. At each dwell, an estimate of the signal's doppler shift is generated and used to identify the DFT spectral components closest in frequency to this estimate. The selected Mark and Space spectral components are then processed in a binary detection scheme involving magnitude comparison. All other spectral components are ignored.

Both of the above approaches are discussed here. Their performances are compared to matched filtered non-coherent FSK for which the bit error rate is

$$P_e = \frac{1}{2} e^{-\gamma/2} \qquad (1)$$

where $\gamma = E_b/N_0$ is the bit energy

contrast ratio (6).

Multibit Detection Scheme

For FSK signals, either the mark or the space channel contains signal plus noise; the other channel contains noise only. Detection consists in determining which channel contains the signal. The noise only channel has a voltage r with Rayleigh statistics

$$P(r) = \frac{r}{N} e^{-r^2/2N} \qquad (2)$$

while the signal + noise channel voltage follows the Rician distribution

$$P(r) = \frac{r}{N} e^{-r^2/2N} e^{-A^2/2N} I_0(rA/N) \quad (3)$$

where N is the noise variance, A is signal amplitude, and I_0 is the modified Bessel function of the first kind.

A simple detection procedure consists in spectrally analyzing each of Mark and Space waveforms with an M-point complex discrete Fourier transform. Specifically, the I and Q sequences of figure 1 each contribute M (real) points to the DFT during a Mark (or Space) dwell. This implies M = TW, where T is the dwell duration and W is the doppler bandwidth to be analyzed. A Mark (or a Space) is declared according to which dwell produces the spectral component of largest magnitude.

Each of the M complex DFT outputs represents the spectral contribution in a bandwidth equal to the signal bandwidth. Prior to comparison, the square of the modulus for each spectral ouptut is computed by complex multiplication. This

transformation, $y \leftarrow r^2/2N^2$, yields a simple distribution for noise only

$$P(x) = e^{-x} \qquad (4)$$

and for signal plus noise

$$P(y) = e^{-(y+\gamma)} I_0 \sqrt{4\gamma y} . \qquad (5)$$

Assuming a Mark is transmitted, a bit detection error arises when the maximum squared modulus occurs as one of the Space spectral outputs. Of the M Mark outputs, one is y-distributed as per equation (5), and the remaining M-1 are x-distributed as per equation (4). All of Space outputs are x-distributed. An error occurs when one of the space x exceeds y and all of the M-1 mark x's. The corresponding probability is

$$P(N) = \int_0^\infty [P(y<x) \cdot P(x_1<x) \cdot P(x_2<x) \cdots P(x_{M-1}<x)] e^{-x} dx \qquad (6)$$

where

$$P(x_k<x) = \int_0^x e^{-t} dt = 1-e^{-x} \qquad (7)$$

and

$$P(y < x) = \int_0^x e^{-(y+\gamma)} I_0 \sqrt{4\gamma y}\, d\nu . \qquad (8)$$

Equation (6) readily reduces to

$$P(M) = e^{-\gamma} \sum_{k=0}^{M-1} (-)^k \binom{M-1}{k} \times \left[\frac{e^{\gamma/(k+2)}}{(k+1)\ (k+2)}\right] \qquad (9)$$

which in turn reduces to the familiar result (1) for M = 1. Equation (9) is the probability that one of the x exceeds y as well as (M-1) x variables. A wrong bit detection decision will be made if the maximum noise modulus belongs to any of the M Space outputs. Since there are M equally likely ways in which this can occur, each with probability P(2M-1), the resulting bit error rate is

$$P_b = M\ P\ (2M-1) . \qquad (10)$$

Results for M=1 (matched filter), 8, 16 and 32 are shown in figure 3. The performance degradation, compared to matched filtering, amounts to only 2 to 3 dB. Furthermore, the additional degradation incurred by doubling M rapidly decreases with increasing M.

Binary Detection Scheme

Assuming no acceleration, the effect of doppler is to displace spectral lines, thus introducing leakage in the measured spectrum. This has several effects:

1. decreases the magnitude of the signal's spectral response,
2. introduces sidelobes, and
3. changes the statistics of Mark spectral outputs to correlated Rician variables.

In the doppler adaptive approach, the DFT output believed to correspond to the doppler-shifted signal is computed from bit to bit. Comparison of the corresponding Mark and Space outputs is used to decide which signal was sent. Since all other 2M-2 outputs are ignored, the presence of sidelobes and the difficulties of correlated Rician variables are of no consequence. The only effect on detection is the effective decrease in signal power. The fraction of signal power remaining due to a shift of ν Hertz is

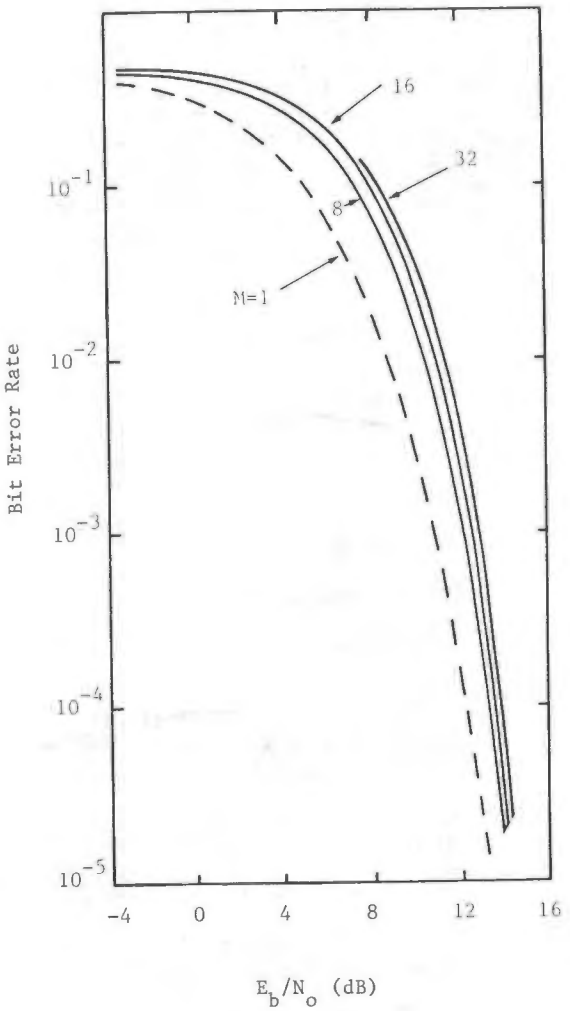


Figure 3: Multibit Detection Scheme Performance

$$S(\nu) = \frac{1}{N^2} \frac{\sin^2(\pi\nu)}{\sin^2(\pi\nu/M)} \qquad (11)$$

where M as before is the number of DFT bins used in the detection process.

The detection bit error rate is found by scaling γ in equation (1) by $S(\nu)$ and averaging over all ν. Since ν is uniformly distributed (all doppler shifts equally likely), the resulting bit error rate is

$$P_b(\gamma) = \int_0^{1/2} \exp\left[-\frac{1}{2}\gamma\, S(\nu)\right] d\nu \qquad (12)$$

This function is compared to matched filtering on figure 4. The resulting degradation amounts to only 2 dB at a bit error rate of 10^{-3}.

Conclusion

A digital signal processing technique has been designed which greatly improves the performance of a low data rate communications receiver. Based on this preliminary design it appears feasible to implement this scheme with a minimum of hardware (although the hardware actually implemented will be traded off against the software complexity). Work is ongoing to implement this design in the laboratory as a proof of concept. The actual hardware/software implementation and the results of performance testing will be published in a future paper.

References

(1) R. E. Crochiere, L. R. Rabiner, "Multirate Digital Signal Processing," Prentice-Hall, p.48, 1983.

(2) F. J. Taylor, "Digital Filter Design Handbook," Marcel Debber Inc., p. 323, 1983.

(3) L. R. Rabiner, B. Gold, "Theory and Application of Digital Signal Processing," Prentice-Hall, p. 328, 1975.

(4) F. J. Taylor, "Digital Filter Design Handbook," Marcel Debber Inc., p. 150, 1983.

(5) S. Magar, et al., "Signal-processing µC builds FFT-Based Spectrum Analyzer," Electronic Design, p. 149, August 19, 1982.

(6) M. Schwartz, W. R. Bennett and S. Stein, "Communication Systems and Techniques," McGraw-Hill, p. 297, 1966.

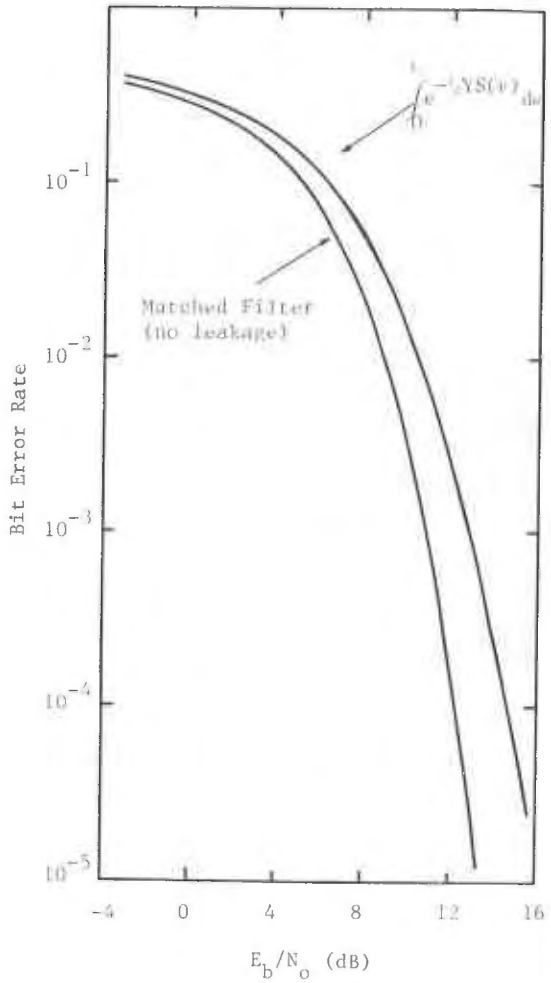


Figure 4: Effect of Spectral Leakage

THE IMPACT OF ADVANCED COMPUTATIONAL TECHNIQUES ON DIGITAL AVIONICS SIGNAL PROCESSOR PERFORMANCE

William M. Vojir
Grumman Aerospace Corporation
Bethpage, New York 11714

6th DIGITAL AVIONICS SYSTEMS CONFERENCE
December 3-6, 1984
Baltimore, Maryland

ABSTRACT

The implementation of new computational techniques in digital signal processors will have a major impact on avionics system design. Applications algorithms will require significantly fewer multiplications and benefit from increased accuracy. This will open up new possibilities for more complicated sensor data analysis. Novel computational techniques abound and inventors' claims are astounding. The evaluation, practical implementation, and integration of these techniques with new hardware technology is a significant undertaking. Modern computer aided engineering (CAE) tools are essential for design alternative analysis. Simulation of hardware implementations with local engineering workstation technology is feasible and eliminates expensive prototype development. One can design with VHSIC components that may not exist and optimize technology insertion advantages.

INTRODUCTION

Advances in signal processor performance have traditionally been linked to the integrated circuit revolution and architectural innovations. The industry and DoD are being driven to greater efforts in these areas by the spectacular computational requirements for such applications as synthetic aperture radar (SAR), image generation and enhancement, and adaptive nulling. The Very High Speed Integration Circuit (VHSIC) and Enhanced Modular Signal Processor (EMSP) programs are two familiar, realizable examples of this expanded emphasis. In addition, multisensor and multimode applications demand more processor flexibility that can only be achieved with increased programmability. Unfortunately, this feature has an adverse effect on throughput when the standard local arithmetic architectures are utilized.

The great breakthrough of the early 1960s was the Fast Fourier Transform (FFT) algorithm. It is recognized, that this algorithm was essential for the development of modern real-time signal processing applications. Twenty years later, the FFT butterfly is still the computational kernel of existing and future signal processor designs. New projects estimate their computational loading based on the use of the FFT algorithm; processors are then designed to meet these throughput requirements. Recently, a wide variety of theoretical results related to alternative computational algorithms was published. All these techniques rely heavily on mathematical concepts from number theory, combinatorial analysis, and abstract algebra. Therefore, little has been done with respect to practical hardware implementations [1].

The primary goal of most of these techniques is to reduce the multiplication burden of the standard processing functions (convolution and DFT). There are three fundamental approaches (approximation, rearrangement, and substitution) being utilized to achieve these results. Approximation methods are primarily used for generating special mathematical functions. Numerical analysis techniques modified for the binary domain have been applied to such functions as CORDIC and high performance complex magnitude circuits [2]. The emphasis of this paper will be on the rearrangement and substitution approaches.

At first glance, these techniques offer significant advantages in realtime avionics processing environments. However, each algorithm comes with unique overhead penalties. The practical implementation of a particular scheme must consider the tradeoff between overhead and performance gain with respect to the target processor architecture. The use of laboratory breadboards to evaluate these alternatives is time-consuming and expensive. The only realistic approach is to use sophisticated CAE techniques and hardware [2,3]. Modern engineering workstation (EWS) technology coupled with powerful computers is essential for the modeling, simulation, and performance comparisons of the alternative designs.

The Discrete Fourier Transform (DFT) has been the target of many new computational approaches. Most performance improvements for this function are based on rearrangements of the original equations. Application of the Winograd technique [4] and prime factor algorithm (PFA) [5] to the DFT is examined in the first section. The specific case of the 24-point transform is used to illustrate the effect of overhead and computational architectures. Implementation of such techniques will affect future signal processing systems design. More exotic algorithms and their possible impact on digital avionics design are discussed next. They are based upon replacing difficult operations with simple ones. Substituting shifts for multiplications is one attractive approach that must be carefully evaluated, since more tedious operations may be introduced. The third section discusses the possibility of entirely eliminating multiplications from certain filtering operations. Finally, the impact of advanced CAE techniques on signal processing system design is analyzed. Sophisticated technology and applications inevitably have led to the development of this industry. Powerful CAE tools (hardware and software) can be a significant advantage in integrating new techniques and technologies (VLSIC/VHSIC) into modern avionics systems.

DISCRETE FOURIER TRANSFORM

The DFT is probably calculated more than any other discrete function in digital avionics signal processing systems. No other calculation has received as much researchers' attention. The FFT

algorithm is the first practical application of a rearrangement scheme applied to the DFT; since 1963 it has had a tremendous impact on the aerospace industry. New signal processing systems are still designed around the FFT butterfly. A large segment of VHSIC technology is targeted for FFT implementation [6]. As a subject for technical journal articles, this algorithm ranks number one even today. However, recent research has led to two promising, similar approaches - the Winograd algorithm (WDFT) and PFA - that further reduce the computational burden.

The standard metric used to compare DFT algorithms is the number of complex multiplications required. Results for a given transform length, N, can be summarized as:

- Direct DFT - $O(N^2)$
- FFT - $O(N \log N)$
- WDFT/PFA - $O(N)$.

When it comes to practical implementation, however, there are other factors that also must be considered. The FFT is severely restrictive with respect to transform lengths, which are usually selected as powers of two for maximum efficiency. All rearrangement algorithms require data permutations. The FFT bit reversal requirement is really not a major obstacle. These two negative features are more than offset by the processing gain and the fact that the FFT algorithm is easily mechanized in terms of standard arithmetic kernels. Large transforms are computed using a simple data flow scheme. Variable transform lengths within the same processor are easily accommodated.

The situation with respect to the Winograd DFT is not so straightforward. The WDFT has a number of advantages that makes it a strong candidate for FFT replacement. It has excellent multiplication gain, but in addition, a larger set of transform lengths are available to the user. General complex multiplications are not required, only real or imaginary times complex. The WDFT expandability is based upon the nested multiply property. By cleverly replacing scalars with vectors and multiplies with short transforms, an iterative approach to longer transforms can be implemented. These advantages are gained at the cost of extra overhead, especially the input/output (I/O) and programmability areas. I/O data permutations (Winograd weave) are computed via the Chinese Remainder Theorem (CRT) and are a function of transform length. In addition, input and output permutations are different for the same length. Different transform lengths are not achieved through simple replication of a computational kernel and the data flow is not as orderly as for the FFT algorithm. The basic WDFT computational flow is shown in Fig. 1.

Figure 1 describes the data flow, but does not give any indication of control complexity. To illustrate the process, consider the example of a 24-point WDFT.

As the sampled data arrives to be processed, it is stored according to the Winograd weave addressing scheme shown in Table 1. This can be viewed as a sequence of address offsets from the location of the first sampled data point, an operation that effectively divides the data into three vectors of eight samples each, according to the data sample indices given in Table 2. Note that the 24 data samples are labeled 0 to 23. The next step is to create three auxiliary vectors:

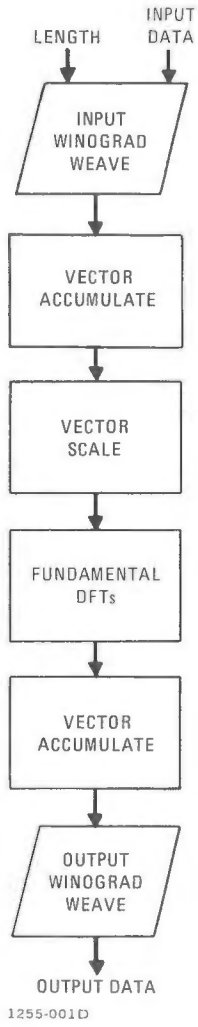


Fig. 1 WDTF Flow Chart

Table 1 Input 24-Point Winograd Weave Address Offsets

SAMPLE	ADDR	SAMPLE	ADDR	SAMPLE	ADDR
0	0	8	8	16	16
1	19	9	3	17	11
2	14	10	22	18	6
3	1	11	9	19	17
4	20	12	4	20	12
5	15	13	23	21	7
6	2	14	10	22	18
7	21	15	5	23	13

1255-002D

$$\bar{u}_0 = \bar{a}_0 + \bar{a}_1 + \bar{a}_2,\ \bar{u}_1 = \bar{a}_1 + \bar{a}_2,\ \bar{u}_2 = \bar{a}_2 - \bar{a}_1$$

with a simple vector add/subtract instruction. These vectors must then be scaled according to:

$$\bar{v}_0 = \bar{u}_0,\ \bar{v}_1 = -3/2\bar{u}_1,\ \bar{v}_2 = j\sqrt{3}/2\ \bar{u}_2.$$

Table 2 24-Point DFT Input Sample Locations

VECTOR COMPONENT	$\bar{a}_0$	$\bar{a}_1$	$\bar{a}_2$
0	0	8	16
1	3	11	19
2	6	14	22
3	9	17	1
4	12	20	4
5	15	23	7
6	18	2	10
7	21	5	13

1255-003D

Note that the scale factors are real or pure imaginary. At this point, three 8-point DFTs are executed on the scaled vectors to create:

$$\bar{M}_0 = DFT_8(\bar{v}_0),\ \bar{M}_1 = DFT_8(\bar{v}_1),\ \bar{M}_2 = DFT_8(\bar{v}_2).$$

One important feature of the algorithm at this stage is that these internal (or nested) DFTs can be executed with the same or different algorithm, an advantage for implementing this technique in signal processors with FFT-based arithmetic architectures. DFT outputs are then accumulated via:

$$\bar{A}_0 = \bar{M}_0,\ \bar{A}_1 = \bar{M}_0 + \bar{M}_1 + \bar{M}_2,\ \bar{A}_2 = \bar{M}_0 + \bar{M}_1 - \bar{M}_2.$$

The output data relationship to these vectors is given in Table 3. Table 4 lists the output sample address offsets required to unweave the data if the vectors $\bar{A}_0$, $\bar{A}_1$, and $\bar{A}_2$ are concatenated in order.

Table 3 24-Point DFT Output Sample Location

VECTOR COMPONENT	$\bar{A}_0$	$\bar{A}_1$	$\bar{A}_2$
0	0	16	8
1	9	1	17
2	18	10	2
3	3	19	11
4	12	4	20
5	21	13	5
6	6	22	14
7	15	7	23

1255-004D

It is instructive to compare the number of arithmetic operations for four possible 24-point DFT implementation alternatives. The full Winograd approach utilizes a nested 8-point DFT that requires four real multiplies and 52 real additions. The two vector accumulate stages are assumed to be executed in the most efficient manner and so require 48 real additions each. A hybrid approach using an 8-point FFT algorithm for the three nested 8-point DFTs is a realistic alternative to the full Winograd technique. An 8-point FFT requires 12

Table 4 Output 24-Point Winograd Unweave Address Offsets

SAMPLE	ADDR	SAMPLE	ADDR	SAMPLE	ADDR
0	0	8	16	16	8
1	9	9	1	17	17
2	18	10	10	18	2
3	3	11	19	19	11
4	12	12	4	20	20
5	21	13	13	21	5
6	6	14	22	22	14
7	15	15	7	23	23

1255-005D

radix-2 butterfly operations (each with four real multiplies and six real additions). The PFA is another method for executing DFTs using cyclic convolution and a multidimensional approach. It requires data permutations similar to the WDFT but does not have the nesting property. A 24-point DFT executed using the PFA requires eight 3-point DFTs followed by three 8-point DFTs. This illustrates the major difference between these two techniques - nested DFTs for the WDFT as opposed to DFTs for each prime factor of the transform length for the PFA. The fourth approach is to use a 32-point radix-2 FFT by appending eight zeros to the 24-point data set. Table 5 summarizes these results. The direct DFT is included for comparison. The operation column is the number of real arithmetic operations required and is just the sum of the first two columns. Assume a worst case situation (add time = multiply time) and define a relative efficiency factor as:

$$E = \frac{\text{FFT REAL ARITHMETIC OPERATIONS}}{\text{ALGORITHM REAL ARITHMETIC OPERATIONS}}$$

Table 5 Summary Comparison of 24-Point DFT Algorithms

ALGORITHM	REAL			EFFICIENCY
	MULTIPLI-CATIONS	ADDI-TIONS	OPERA-TIONS	
WINOGRAD/PFA	44	252	296	2.7
HYBRID WINOGRAD/FFT	160	312	472	1.7
FFT	320	480	800	1.0
DIRECT	1152	1104	2256	0.35

1255-006D

The values in the last column of Table 5 are the number of 24-point DFTs that can be calculated in the time required to calculate the same function with an FFT implementation. Although the efficiency numbers are impressive, the user must account for the additional overhead required by some of these new algorithms. Different arithmetic and control architectures may have a significant effect on the computational gains realized in any particular situation. Realization of a programmable transform length WDFT could be a significant signal processing technology breakthrough. However, more

work must be done to evaluate and tradeoff various alternatives based upon modern and future hardware and software capability. The only reasonable method for doing this is to use some of the new CAE technology, which is discussed in the last section.

EXOTIC SUBSTITUTION ALGORITHMS

This class of algorithms can be separated into two categories, transforms and alternate information representation. Exotic transforms, such as the number theoretic transform (NTT), the polynomial transform, and the Galois Field transform, are similar in structure to the DFT, but do not require general complex multiplication operations. If the ultimate reason for using a particular transform is to execute linear convolutions (FIR filters), then the transform variable domain is not being used. Typically in this situation, using the FFT algorithm has been less costly than direct implementation. However, because we are not interested in recovering frequency information, alternate transforms that satisfy the cyclic convolution property would give satisfactory results. All transforms mentioned have been designed to replace the general multiplication with some shifting operation. Although this is an extremely desirable substitution, the side effects have to be carefully evaluated before any implementation decision is made. On the other hand, if the data word lengths can be significantly reduced before doing any arithmetic, then these operations can be executed faster. The residue number system (RNS) representation is one method of accomplishing this that replaces long data word processing with several parallel short data word channels [7]. Because this is essentially a coding technique, it depends upon the CRT for decoding back into useable formats.

Specifically, the RNS technique is based on interpreting incoming data as integers and encoding them in terms of their residues with respect to a set of relatively prime moduli. The selected set of moduli, when multiplied together, determines the process's allowable dynamic range. All arithmetic operations are executed in the residue representation. Unfortunately, not all discrete functions fit within this restriction. Calculation of FIR filters is the best application; a practical approach to the implementation has been described [1]. FIR filter dynamic range is easy to estimate and a relatively simple analysis will determine the optimum moduli set. It is conceivable that this procedure can be done in real time to enable adaptive resource (channel) allocation. When this approach is considered for avionics system implementation, a residue representation frees the designer from normal arithmetic problems, i.e., overflow, rounding, and truncation. All results are exact until final decoding. At this point, required word lengths for CRT implementation can be quite long. For example, consider the moduli set (256, 15, 13, 11, 7) with a dynamic range of 3843840. With 8-bit data and coefficients, the longest FIR filter than can be executed without decoding ambiguity is of length 59. The shortest allowable filter is determined by the decoding speed, processing speed, and pipeline configuration. Final output data is 22 bits long, but the active decoder only has to process 14-bit data.

Transforms based on advanced algebraic concepts are derived for the express purpose of replacing multiplications with shifting operations while simultaneously satisfying the convolution theorem. NTTs have been discussed [8,9] that satisfy these criteria. When NTTs are considered for application to practical signal processing situations, a number of problems arise. The transform length and data word length are intimately related, because all arithmetic must be executed modulo and integer. Normally, modulo arithmetic requires a division operation or some equivalent procedure for calculating appropriate residues. Unless there is some easy way of calculating these residues, the advantages gained by the shifting operations would be lost in the modulo arithmetic. This disadvantage can be overcome by selecting the moduli as a Mersenne prime, i.e., a prime of the form 2^q-1. Arithmetic for the Mersenne Number Transform (MNT) is trivial - one's complement addition in a q-bit word and a q-bit rotation for multiplication by powers of two. Unfortunately, because q is a prime, there is no fast MNT similar to the FFT relationship to the DFT. Fermat Number Transforms (FNTs) are an attempt to generate a NTT with the required symmetry to facilitate a fast transform algorithm. Although this property was achieved, the simple arithmetic inherent with the MNT was lost.

The other exotic transforms have similar properties, each with its share of advantages and disadvantages. Application of these techniques depends on the practicality of implementation. The algorithms mentioned in this section have another advantage in addition to speed: the results are all exact. None of the accuracy problems inherent with normal arithmetic - truncation, roundoff, overflow - occur when using advanced algebraic techniques and modulo arithmetic. This feature may have significant advantages for future processing systems requiring high precision.

MULTIPLIER-LESS FILTERING

The typical response of a computer-aided filter design program to engineering specifications is a list of coefficients - the impulse response for FIR filters or the second order IIR filter section constants. While most design aids will attempt to minimize the filter length and some will include a wordlength sensitivity analysis, the recommended implementation will involve general arithmetic operations - multiplication, addition, and delay. If addition execution is significantly faster than that of a multiplication (which is the case when fixed point arithmetic is used), then filters without multiplications may have a speed advantage over the general case.

Specifically, restrict the allowable multiplications to include only shifts, i.e., multiplications by powers of two. The resulting filter lengths may be longer than those produced with the standard design approaches, but the implementation speeds could be considerably faster. Although the benefits sound attractive, the design process for a given set of engineering specifications is not that well defined. Eliminating multiplications is a severe restriction, so applications may be limited to certain classes of filters. A potential problem for the design engineer is the availability of useful CAE techniques for multiplier-less filter synthesis.

An interesting and fundamental approach to the design problem for linear phase FIR filters is described in Ref. 10. A set of basic multiplier-less filters are cataloged for use as building

blocks in the design process. Actually, the first six entries in the fundamental filter menu are special cases of:

$$H(z) = \sum_{n=0}^{N-1} (\pm z^{-p})^n$$

where the amplitude of the frequency response function is given in Table 6. The approach then, is to combine selected building blocks from the menu using the subfilter architecture technique [11], but without general multiplications. At present, a heuristic design procedure, based upon a knowledge of the zeroes and the effects of cascading or paralleling building blocks, can be used. Automating this procedure is possible, but obtaining an optimum design would require a very sophisticated CAE tool.

Table 6 General Multiplier-Less Filter Building Block Frequency Response Amplitude Function

FILTER LENGTH	NON ALTERNATING	ALTERNATING
EVEN	$\frac{\text{SIN}(\pi f N p)}{\text{SIN}(\pi f p)}$	$\frac{\text{SIN}(\pi f N p)}{\text{COS}(\pi f p)}$
ODD	$\frac{\text{SIN}(\pi f N p)}{\text{SIN}(\pi f p)}$	$\frac{\text{COS}(\pi f N p)}{\text{COS}(\pi f p)}$

1255-007D

The point is, that the task of the digital avionics systems engineer is becoming more complicated. With throughput at a premium in future systems, special filtering techniques such as described here must be considered. The ability to predict the future performance characteristics of avionics signal processors is no longer a straightforward extrapolation of previous implementations. A clever design team can squeeze more out of a particular hardware realization by substituting multiplier-less filters for the standard types. In fact, wherever a programmable processor is being utilized in present systems, it may be possible to increase performance by using this approach. The technique is attractive, since it does not require new mathematical functions (e.g., residue arithmetic) to be implemented in real time.

COMPUTER-AIDED ENGINEERING APPLICATION

The generic techniques described in this paper have not been seriously evaluated by the processor manufacturers for eventual implementation. Problems associated with carrying out detailed engineering investigations are formidable. The mathematical basis requires specialized knowledge of sophisticated theories not normally available in the average electronic design environment. A great deal of preliminary analysis must be performed before any practical design can be contemplated. All computer analysis programs have to be developed from scratch because no existing software libraries contain the required routines. The mathematician and the hardware engineer must work together to produce a feasible design. At this point, a prototype can be developed that will demonstrate the performance of the particular technique. This approach is very time-consuming, inflexible, and may not give a true evaluation of the final implementation. There has been no experience with these techniques, so one would expect many hardware iterations before finalizing any design. Because we are dealing with extremely complex implementations, this development method is not feasible. The approach to take is accurate simulation of the design utilizing computer resources.

Computer technology is not new to the aerospace industry. CAD/CAM systems have proved themselves in vehicle design and manufacturing processes over the past decade. However, it is only recently that an organized effort to bring the same benefits to the electronics industry has developed. Specifically, EWSs were designed for the integrated circuit market, but include a number of features that would make them useful in the systems integration environment. Those properties of EWSs that are especially valuable in this respect are: hierarchical representation, behavioral modeling, higher order language programming, inter resource communication, local simulators, local timing verification, and sophisticated graphics.

The engineer can create a design using the top-down approach which maintains system integrity, both vertically and horizontally. Behavioral modeling is a means of functionally implementing sophisticated circuitry without defining the component level representation. Complicated devices can be incorporated quickly, and various alternatives can be explored efficiently. A major feature of the EWS is the ability to locally simulate the operation of a design. Complicated results from sophisticated signal processing circuits cannot be verified by hand. The ability to write software to analyze the simulator results is an important property. This approach can be applied to any design level, and is only bounded by the physical limits of the computing resources.

Utilizing EWS technology is an ideal method for evaluating realizations of new computational techniques. Analysis, simulation, and timing verification can be integrated within a local computing environment. Rapid interaction with the results enables a flexible approach to optimizing designs and evaluating interrelationships with existing system blocks. If one wants to commit to actual hardware, the mechanisms are already in place. When VHSIC specifications are complete, these same tools can be used for technology insertion and applications experimentation. To illustrate this concept, consider the simple example of the common TRW multiply and multiply accumulate chips. A behavioral model is created on an appropriate EWS based on vendor specifications. Symbols and model (Fig. 2) are fused and stored in a universally accessible file for application in potential designs. For example, one can design a digital filter or a 3-point WDFT around the use of these chips and verify performance characteristics without building a hardware prototype.

CONCLUSION

Digital signal processing performance improvements are not limited to those arising from new hardware architectures and integrated circuit technologies. New computational techniques will have a major impact on avionics systems design. Signal processing intensive applications can benefit from using implementation algorithms with the minimum number of arithmetic operations. Several cases were described that illustrate their relative merits. The WDFT and PFA algorithms are

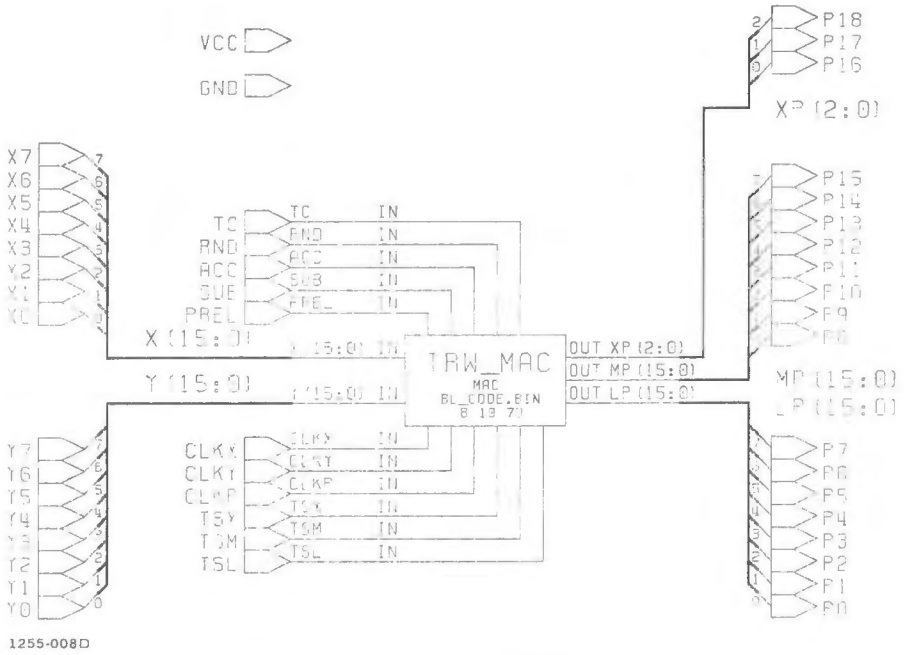


Fig. 2 Basic TRW Multiply Accumulate EWS Model

significantly less computationally-intensive than the FFT. The RNS representation and NTTs offer attractive approaches to the standard FIR filter execution. Multiplier-less filters is a less exotic alternative that may be easier to understand. In all cases, the theory is difficult and practical implementation in a flexible format must be carefully evaluated. CAE technology is a low cost, reasonable means for examining new designs and applications. Simulation of implementation schemes with local EWSs is feasible from the device to the system level. One can design with components (e.g., VHSIC) or subsystems that do not exist and optimize system level advantages. Because new computational techniques are independent of the high level macro-instructions for generating application algorithms, this approach to performance enhancement is essentially technology transparent.

ACKNOWLEDGEMENTS

The author would like to thank C. Jankowski and T. O'Hara for reviewing the manuscript.

REFERENCES

1. W. Vojir, "Hardware Design for New Digital Signal Processing Techniques," NAECON '80
2. C. Jankowski, "The Effects of Advanced Digital Signal Processing Concepts on VLSIC/VHSIC Design," NAECON '83.
3. A. Cosgrove, "Efficient Computer-Aided Engineering," 1982 National SWE Convention
4. S. Winograd, "On Computing The Discrete Fourier Transform," Mathematics of Computation, V. 32, No. 141, Jan 1978, pp 175-199
5. D. P. Kolba and T. W. Parks, "A Prime Factor FFT Algorithm Using High-Speed Convolution," IEEE Trans Acoust, Speech, Signal Processing, Vol ASSP-25, No. 4, pp 281-294
6. VHSIC Applications Workshop, Grumman Aerospace Corp, Bethpage, NY June 11-13, 1984
7. W. K. Jenkins and B. J. Leon, "The Use of Residue Number Systems in The Design of Finite Impulse Response Digital Filters," IEEE Trans Circuits and Systems, Vol CAS-24, No. 4, April 1977, pp 191-201
8. C. M. Rader, "Discrete Convolutions via Mersenne Transforms," IEEE Trans. Comput., Vol C-21, Dec. 1972, pp 1269-1273
9. R. C. Agarwal and C. S. Burrus, "Number Theoretic Transforms to Implement Fast Digital Convolution," Proc IEEE, Vol 63, Apr 1975, pp 550-560
10. R. C. Agarwal and R. Sudhakar, "Multiplier-less Design of FIR Filters," International Conference on Acoustics, Speech and Signal Processing, 1983
11. J. F. Kaiser and R. W. Hamming, "Sharpening The Response Of A Symmetric Nonrecursive Filter By Multiple Use Of The Same Filter," IEEE Trans on Acoustics, Speech, and Signal Processing, Vol ASSP-25, pp 415-422, Oct 1977

SELF ADAPTIVE FILTERING OF ENVIRONMENTAL NOISES FROM SPEECH*

D. Graupe**, J. Grosspietsch*** and S. Basseas***

ABSTRACT

This paper describes work on the design, testing and evaluation of a self-adaptive filter to filter environmental noises from speech when noise and speech parameters may arbitrarily vary (as they do) and are a-priori unknown and when no access is available to noise alone or to speech alone, but only the sum of speech plus noise is available. The above situations are typical to what exists in almost any military/navy situation, in aircraft, hangar, helicopter and even office or cafeteria.

The study was performed in two directions, one, a time-domain design and the other a frequency-domain design. Although these two designs are in theory directly inter-transformable, real-time realization presents computational-speed and other problems to the time-domain design that do not exist in the frequency-domain designs.

Performance tests on improvement via filtering in signal-to-noises-ratio (SNR) and in intelligibility (for mono-syllable standard word lists) in a wide range of noises, all point out to the superiority of the frequency-domain design, whose real-time on-line realization is also the simplest.

Indeed, performance tests, as presented in the paper, show SNR improvements via the frequency-domain design from 6.7 to 25.5 db for a multitude of noises throughout the spectrum, and where the input SNR (prior to filtering) varies from -5 to -20 db. The 10db or higher improvement was the more common one, even in cafeteria/cocktail situations.

Intelligibility-scores were equally impressive, as is tabulated, and reached, in cafeteria situations, an improvement from a 32% correct score prior to filtering to a 90% score after frequency-domain filtering, the average improvement being 71% (for a single-syllable words). Listening comfort, though not quantitatively measurable, was most impressive, this being a major aspect in fatigue when listening in a noisy environment. Ther performance of the resulting filter thus points to providing considerable intelligibility, SNR and listener-comfort improvements under the widest possible noise situations, when noise is a-priory unknown and inaccessible, as it is under real world conditions and especially in military/navy environments.

1. INTRODUCTION

This paper describes work on self-adaptive filtering of speech from enivronmental noises. This work, is an extension and rigorization of previous work at IntelliTech, Inc., on self-adaptive filtering of speech from noise when applied to hearing aids.

The present research has taken two parallel design directions, on based on a frequency-domain approach and one on a time-domain approach. Both designs have been subsequently tested for signal-to-noise ration (SNR) improvements and for intelligibility (with vs. without self-adaptive filtering) when using standard single-syllable unrelated word lists, considering a broad range of noise situations usually of high level noises, covering the frequency spectrum of interest in military application of up to 4KHz. The noises include also strong cafeteria/cocktail noises, and babble noise situations as well as various high-frequency and low-frequency noises that are common in aircraft and helicopter environments and in machine shops, hangar, etc.

2. METHOD

2.1 General Analysis

The self-adaptive filtering problem is that of filtering a signal s(t) from noise n(t) when access is available only to a measurement signal y(t) where

$$y(t) = s(t) + n(t) \quad ; \quad t = \text{time} \qquad (1)$$

and wnere no prior parameter information is available on either s(t) or n(t), see Fig. 4. The above problem can only be solved if some discriminant feature exists to facilitate discrimination and separation of (some) speech from noise. Fortunately, this situation exists in speech vs. most environmental noise situations, since speech phonemes, and hence speech parameters - spectral or time-domain (not necessarily its power), vary abruptly

* This work was performed at IntelliTech, Inc. This work was supported in part by NADC, Warminster, PA, under Contract N62269-83-R-0087.

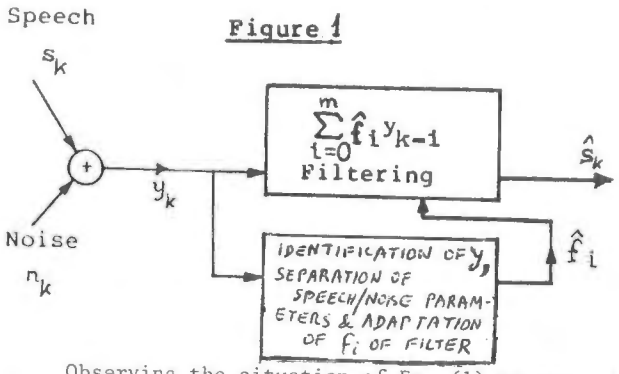
** Dept. of Electrical and Computer Engineering, Illinois Inst. of Technology and IntelliTech, Inc., Northfield, Il 60093

*** IntelliTech, Inc., Northfield, Il 60093

at every 5 to 100 milliseconds (and do so in a rather chaotic manner).

In contrast, the parameters of most environmental noises vary rather slowly over these time ranges. The latter is true even for noises such as cocktail party noise or babble noise (several persons speaking simultaneously), since an averaging effect on the various speech phonemes of the various speakers takes place, which is further assisted by room acoustics, to dampen the rate of change of the parameters of these noises. Whereas the degree of stationarity is not the only discriminant feature that can be used for discrimination as required in the above self-adaptive filtering problem [2, Ch. 11], it is adequate in speech/noise situations.

Figure 1



Observing the situation of Eq. (1), we can only identify a mathematical model for the measurement of y(t). A time domain such model for discrete time y_k ($k\Delta t=t$; k=0,1,2,...; t=time interval) may be given by a pure autoregressive (AR) time-series equation:

$$y_k = \sum_{i=1}^{n} a_i y_{k-i} + w_k \qquad (2)$$

w_k being inaccessible second-moment-ergodic white noise, s.t.

$$E[w_k w_p] = \begin{cases} W & \forall\ k=p \\ 0 & \forall\ k\neq p \end{cases} \qquad (3)$$

Of course, other models such as MA (moving-average), ARMA (mixed autoregressive-moving-average), AC (autocorrelation model), or frequency domain models, say, FT (Fourier transform) models, are also adequate. The parameters for the model for y_k, say the AR parameter a_i for the model of Eq. (2) are all that can be directly identified. However, for adequate filtering of s_k and/or n_k, such as the Wiener-Levinson filter [3] or the extended Kalman filter (noting the non-white noise character of n_k [2, Ch. 10], we require both the parameters of s_k and n_k. Yet, via Ch. 11 of [2], we can obtain the parameters of n_k given the parameters of y_k and of s_k. Furthermore, using an approach, as proposed by Tsypkin [4] or Widrow [5] (see Ch. 10 or [2]), we may design a semi-adaptive filter where a-priori knowledge of speech parameters alone or noise parameter alone is still required.

From the above it is obvious that the derivation of at least the parameter of speech or of noise is a pre-requisite to filtering of noise from speech. To derive such parameters we must first identify the parameters of a model for the measurement sequence $\{y_k\}$ which is the only accessible data sequence. Which of the various models one actually identifies, is mainly a matter of convenience and speed, since the various models are linearly intertransformable. In our designs we have identified an DFT (discrete FT) model in the frequency-domain and AR and AC models in the time domain, all of which are directly applicable to WF (Wiener Filter) algorithms.

Once the parameters of a model for y_k have been identified, the task of obtaining, from the latter parameters, models for s_k and/or for n_k must be solved, s_k and n_k denoting the speech and noise sequences, respectively. This separation of parameters is based on the property of speech mentioned earlier, that speech phonemes, i.e. speech parameters, vary in a rather chaotic manner every 5 to 200 milliseconds, whereas environmental noises are usually relatively stationary over these time intervals. Hence, if we identify a set of parameters for y_k and note the changes in the identified parameters over time, then, and assuming the identification to be sufficiently accurate, the variation in the values of these parameters is due to speech. If, say, the autocorrelation (AC) parameters $R_i(y)$ of y_k are considered, namely

$$R_i(y) \triangleq E[y_k y_{k-i}];\ i=0,1,2,\ldots m \qquad (4)$$

such that

$$\frac{1}{k}\sum_{j=0}^{k-1} y_j y_{j-i} \xrightarrow{p} R_i(y) \qquad (5)$$

$\xrightarrow{p}$ denoting convergence in probability when y_k is second moment ergodic [2], then one can show (see Table 1) that $R_i(s)$, i>0, i.e. R_i of $\{s_k\}$ vary in their sign from one speech phoneme to another. Furthermore, for unvoiced speech phonemes (phonemes of consonants), the average of $R_i(s)$ over several different phonemes is, due to this sign variation, close to zero. Furthermore, by [6, Ch.6], the power of unvoiced speech phonemes lies mostly above 1500 Hz, these phonemes being of considerably lower energy than those of voiced speech. Therefore, by Table 2, the average of $R_i(y)$ for $y_k=s_k+n_k$, over several unvoiced segments of s_k (these last each between some 5 to 50 millisec.), is more or less equal to $R_i(n)$, i.e. to R_i of the relatively far more stationary noise. Hence, to identify $R_i(n)$ of n_k, it approximately satisfies

$$\underset{k}{Ave}[R_i(y)] \sim R_i(n) \qquad i>0 \qquad (6)$$

$\underset{k}{Ave}[\ldots]$ denoting averaging of h unvoiced phonemes taken over an interval $T \gg \tau$, τ being the length of a phoneme. Noting that a speech phoneme usually lasts between 5 and 200 m.sec., we set T at 330 m.sec., a duration over which $R_i(n)$ is assumed to be (more or less) constant. Now, to determine that $R_i(y)$ of Eq. (6) are chosen from unvoiced phonemes, or, if possible (and even better) from speech-less, i.e. from "noise only" sub-intervals, each of a 7.5 to 15 m.sec. duration (a typical duration for an unvoiced phoneme), we only consider these τ where $R_0(y)$ is the lowest over the T milliseconds total interval. Observe that, by the low power of $R_i(n)$ in these h sub-intervals, any error in evaluating $R_i(n)$ according to Eq. (6) is very small.

Once $R_i(n)$, i>0, are available, we still do not have all parameters of s_k or even of n_k (specifically we do not have the $R_0(n)$ parameter). Hence, in order to derive the remaining parameters as requred if any WF (Wiener filter) or its equivalent

is to be employed, we employ the relation [2,Ch. 11]:

$$E[\rho_k^2] = \sum_{h=0}^{p+r} \eta_h^2 V + W$$
$$E[\rho_k \rho_{k-1}] = \sum_{h=0}^{p+r} \eta_h \eta_{h-1} V \qquad (7)$$
$$\vdots \qquad\qquad \vdots$$
$$E[\rho_k \rho_{k-p-r}] = \eta_{p+r} V$$

where W and V are the variances of w_k, v_k, respectively, which, in turn, are the white-noise residuals of linear time series model, for s_n and n_k respectively and where η_i, ρ_k relate to ARMA, MA and AR models for y_k, n_k and s_k respectively as in [2].

The AR, MA and ARMA parameters of Eq.(7) interrelate, via polynominal division [2, Ch. 8], whereas the R_i parameters and the AR parameters interrelate via the Yule-Walker equation [2, Ch. 2]. The latter derivations of Eq. (7) thus complete the identification of $R_i(n)$, $R_i(s)$, for setting the parameters f_i of a Wiener filter (WF) equation to obtain an LS (least-squares) filtered estimate $\hat{s}_k$ of s_k frp, y_k, s.t.

$$\hat{s}_k = \sum_{i=0}^{m} f_i y_{k-i} \qquad (8)$$

In our design, and since phonemes $\{s_k\}$ are to be short-lived, then in order to allow resetting of f_i according to their individual $R_i(s)$ values, we assume that $R_i(s)$ satisfy a long-term average flat spectrum. Indeed, by [6. Ch.6] we know that the long-term average spectrum of s_k is not flat but decreases with frequency. However, since the information content of speech increases with frequency [6, Ch. 6], the flat spectrum assumption appears justified. Hence f_i are chosen to satisfy [3]:

$$\min_{f_i} E[(s_k - \hat{s}_k)^2] \Rightarrow \frac{\partial}{\partial f_i} E[(s_k - \hat{s}_k)^2] = 0 \qquad (9\text{-a})$$

which yields:

$$\sum_{j=0}^{m} f_j E[y_k y_{k-j+j}] = E[s_k s_{k-i}]; \; i = 0, 1, \ldots m \qquad (9\text{-b})$$

as long as $E[s_k n_p] = 0 \; \forall k, p$ as is obviously the case. Alternatively, if the long-term flat spectrum assumption for s_k is dropped, a semi-adaptive version of Eq. (9), can be employed,

Table 1
Typical Autocorrelation Coefficients R_0 to R_5 for Certain Speech Phonemes
(Sampling Interval: p.1 m.sec.)

Phonemes	R_0	R_1	R_2	R_3	R_4	R_5
Voiced						
ah	1.33,-1	1.07,-1	4.53,-2	-2.44,-2	-8.08,-2	-1.01,-1
oo	5.34,-2	5.06,-2	4.36,-2	3.41,-2	2.24,-2	1.15,-2
ae	8.29,-2	5.42,-2	7.77,-3	-5.17,-3	1.68,-2	3.76,-2
ee	1.83,-1	1.96,-2	-7.06,-3	1.43,-2	4.11,-2	-8.19,-3
Unvoiced						
v	6.04,-2	1.68,-2	-2.28,-3	1.18,-2	-1.75,-3	-4.93,-3
z	7.34,-2	-3.60,-2	2.28,-2	-1.25,-2	2.51,-2	-1.20,-2
sh	1.15,-2	-4.96,-2	-2.33,-2	5.22,-2	-2.25,-2	5.19,-3
j	4.05,-3	2.21,-3	2.15,-3	2.03,-3	3.09,-3	2.17,-3
r	9.17,-3	8.31,-3	6.22,-3	3.56,-3	9.61,-4	-1.12,-3

Comment: 6.04,-2 means 6.04×10^{-2}

Table 2
Change in R_i (Autocorrelation Coefficients) Over Several 7.5 m.sec Segments of a Speech Sentence Imbedded in Near-Stationary Noise vs. R_i for Same Noise Without Speech
(Sampling Interval: 0.1 m.sec., data length: 7.5 m.sec.)
noise: 2.4 - 3.0 KHz (filtered white noise)

	R_0	R_1	R_2	R_3	R_4	R_5
(a)						
Sentence &	1.34,-1*	-1.25,-2	-1.24,-1	3.52,-2	1.04,-1	-4.91,-2
Noise at	1.10,-1	-3.39,-3	-8.41,-2	2.02,-2	6.02,-2	-1.36,-2
an unvoiced	9.00,-2	2.11,-3	-7.90,-2	4.36,-3	6.99,-2	-2.69,-3
speech segment	1.26,-1	1.25,-3	-9.77,-2	1.55,-2	9.05,-2	-1.06,-2
Ave. R for (a)	1.15,-1	-3.13,-3	-9.62,-2	2.13,-2	8.05,-2	-1.90,-2
(b)						
Sentence &	5.03,-1	1.18,-1	-1.86,-1	9.67,-2	1.66,-1	-1.09,-1
Noise at a	4.01,-1	1.82,-1	1.58,-3	7.16,-2	4.58,-1	-1.26,-1
voiced speech segment						
(c)						
Noise Alone:	1.22,-1	-7.49,-3	-1.07,-1	2.20,-2	8.48,-2	-2.51,-2
Same noise as in (a)	9.06,-2	-7.75,-3	-7.73,-2	2.26,-2	5.89,-2	-2.49,-2
Ave. R for (c)	1.06,-1	-7.62,-3	-9.22,-2	2.15,-2	7.19,-2	-2.50,-2

*5.02,-2 means 5.02×10^{-2}

namely [2, Ch. 11]:

$$\hat{\underline{f}}_k = \hat{\underline{f}}_{k-1} + \gamma_k \left(y_k - \hat{\underline{f}}_{k-1}^T \underline{Y}_k \right) \underline{Y}_k - \underline{R}_{nn} \qquad (10\text{-a})$$

where $\hat{\underline{f}}_k$ is the k'th-stage estimate of the vector:

$$\hat{\underline{f}}_k \triangleq \left[\hat{f}_0 \cdots \hat{f}_m \right]^T \qquad (10\text{-b})$$

T denoting transposition, $\underline{Y}_k$ and $\underline{R}_{nn}$ being vectors satisfying

$$\underline{Y}_k \triangleq \left[y_k \cdots y_{k-m} \right]^T ; \; \underline{R}_{nn} \triangleq \left[R_n(0) \cdots R_n(m) \right]^T \qquad (10\text{-c})$$

2.2 The Time-Domain Filter (TDAF)

By the derivation above, the set of equations (1), (5), (6), (76), (8) and either (9) or (10), is the complete set of equations to perform adaptive filtering, where Eq. (1) describes the measurement model, where Eqs. (6) and (7) yield

the noise parameters over the appropriate sub-intervals τ as above, where either Eq. (9) or (10) yields the WF f_i parameters from the $R_i(n)$ parameters thus obtained, and where Eq. (8) is the subsequent Wiener filtering operation to yield the filtered estimate $\hat{s}_k$ of s_k as required.

In that filtering algorithm we bunch together the h=4 minimal sub-intervals, as discussed in Sect. 2.1. For speeding-up of computation, minimal $R_o(y)$ are used to approximate $R_o(n)$. Computational results of comparing $R_o(n)$ and $R_o(y)$ at these minima, for cases of speech plus noise vs. cases where only noise is present in y_k, show this approximation to be valid (see Table 2). In this design, the order m + 1 = 8 was selected via performance testing as in Table 6 of Sect. 3.3 below and via comparing f_i parameters of Eq. (8) for different m, as in Table 3. The sampling rate is 10,000/sec.

2.3 The Frequency-Domain Filter (FDAF)

Since the time-domain and the frequency-domain models for s_k and n_k are linearly inter-transformable via the Fourier or the inverse-Fourier transform, it is evident that the analysis of Sect. 2.1 above holds also in the frequency domain. In that case, y_k is identified in terms of its spectral components, which in discrete frequency form are $Y(\omega_i)$, ω_i denoting a discrete frequency (angular frequency), and $Y(\omega_i)$ representing y_k at ω_i. The WF eq. (8) now becomes:

$$\hat{s}_k = \sum_{i=1}^{M} F_i Y_k(\omega_i) \qquad (11)$$

F_i denoting the frequency domain WF parameters, $\hat{s}_k$ being the FDAF estimate of s_k and M, being the FDAF filter's order.

The WF parameters F_i are obtained, in parallelity to f_i of Eq. (8), via an identification procedure that is also parallel to that of the time domain case, and where $R_i(y)$ are replaced by their spectral equivalents.

Table 3
Comparison of Filter Parameters of eq. (8) for different filter orders (typical date sets)

Comment: 4.913 -1 means 4.913×10^{-1}

Filter order m + 1:	4	6	8	10
Filter Parameter				
f_o	4.913,-1	5.363,-1	5.369,-1	5.410,-1
f_1	-7.608,-2	-4.594,-2	-4.608,-2	-4.634,-2
f_2	2.362,-1	1.897,-1	1.859,-1	1.864,-1
f_3	-1.102,-1	-9.901,-2	-1.035,-1	1.047,-1
f_4		-1.545,-1	-1.594,-1	-1.732,-1
f_5		-7.926,-3	-1.357,-3	-4.283,-3
f_6			-5.790,-3	1.249,-2
f_7			1.792,-2	8.452,-3
f_8				4.485,-2
f_9				-1.703,-2
Observe the closeness of f_0 to f_4 for m + 1 = 6, 8, 10 (not: 4)				

Hence, in parallelity to Eqs. (7) of the time-domain design, $N(\omega_i)$ determines F_i of Eq. (11) via minimizing the variance as in Eq. (9-a) with respect to F_i (rather than with respect to f_i in the time-domain case), to yield:

$$F_i = \varphi_i[N(\omega_i)] \qquad (12)$$

where the functional relation φ_i is such that the higher $N(\omega_i)$, the lower is F_i. However, by the non-flat distributions of speech power vs. frequency and of speech intelligibility vs. frequency [6, Ch.6] - see table 4, appropriate biasing is included in φ_i.

The exact φ_i have been experimentally fitted via testing SNR (signal-to-noise ratio) improvement and via intelligibility experiments, to optimize SNR and intelligibility performance in a heuristic manner, noting that no exact theory exists for this purpose. Biases, due to the distributions of Table 4, have also been heuristically incorporated in the time-domain design, in parallelity to the frequency-domain case. The FDAF filter's order M or Eq. (11) was chosen as 8 following SNR and intelligibility tests of the kind discussed in Sect. 3 below:

Table 4
Distribution of Speech Power and Intelligibility vs. Frequency*

Frequency Hz	Speech Power: % Lying in Freq. Range	Speech Intelligibility: % Lying in Freq. Range
62-125	5	1
125-250	13	1
250-500	42	3
500-1000	35	35
1000-2000	3	35
2000-4000	1	13
4000-8000	1	12
almost all voiced-speech's energy at	50% of power lies at below 450 Hz	50% of unintelligibility at above 1400 Hz
almost all unvoiced-speech's energy at above 1500 Hz	Peak power at 500 Hz	Peak intelligibility at 1000 Hz

*According to Ch. 6 of [6]

3. PERFORMANCE TESTING: PROCEDURES AND RESULTS

3.1 Testing procedures - SNR

To obtain a concrete and reproducible measure of performance of the various designs of the self-adaptive filter above, and to evaluate the effects of certain design parameters and to set them for best performance, we compared changes in SNR (signal-to-noise ratio) for situations of speech (signal) imbedded in noise, when measured directly (unfiltered) and when passed through the version of the filter that is teted.

To get a good understanding of the filter's performance, these tests were repeated using noises at difference frequencies within the range of speech frequencies of interest. In this manner, one can evaluate the performance of the filter throughout the spectrum. Furthermore, since certain very unpleasant noises cover a range of frequencies and have somewhat non-stationary properties, but are extremely common, such as

babble noises (several persons speaking simultaneously) or cocktail/cafeteria noises, these have been also tested for SNR improvement via filtering. We comment that although these noises are rather close to speech (especially babble noise), a parameter-averaging effect takes place with regard to these noises' parameters, which is due to the fact that several phonemes of different parameters are uttered simultaneously, this averaging being aided by room acoustics. Hence, our basic noise-stationarity assumptions of Sect. 2 do still hold, at least partly, such that improvement is expected and was indeed achieved via our design.

In all cases, no prior knowledge of parameters was assumed. When passing the $y_k = s_k+n_k$ signal through the filter, we did however have the computer store and record the filter's parameter settings. Subsequently, and to obtain an accurate SNR measurement in the "filtered" mode, we forced the filter to be frozen at the above stored parameters while we fed speech alone through the filter thus set, and then we fed noise alone through it. We recorded both pure (speech) signal RMS (root-mean-square) values at filter's input and at its output and the pure noise RMS such values, to derive the SNR ratio at input (prior to filtering) and at output (after filtering).

The above tests were also repeated at different input-SNR ratios for the same type of noise, since the biasing of the filter and its gain policy setting (Eq. (12), Sect. 3.2).

3.2 Testing Procedures - Intelligibility

A somewhat less repeatable but nonetheless important and indicative performance test to evaluate the filter is the intelligibility test. Here one tests intelligibility scores for sequences of words imbedded in noise when filtered vs. when not filtered. To obtain a fairly dependable such evalution, we used sequences of standard single-syllable word lists (NU 6, Northwestern University) which are widely used and widely accepted word lists [1]. These lists span the speech spectrum. Also by their single-syllable feature, there is no indication in one phoneme on the previous or next one. Hence, the listener cannot guess the word if a phoneme is lost. Again no word is related to the other in any content-sense or sentence-sense.

Our scores were of an examined volunteer correctly writing down the word from the list as he hears it. Any error was counted as wrong. Hence, writing "hurl" for "girl," or "sheep" for "get" both receive a zero score. However, for better evaluating performance, we also observed the number of phonemes which were wrongly identified. Here, "hurl" for "girl" is a one-phoneme error, while "sheep" for "get" is a three-phoneme error. Obviously, in conversation situations, only rarely will one-phoneme errors affect sentence understanding or even word understanding. One would never mistake "The girl is 5 years old" for "the hurl is 5 years old."

3.3 Performance Results - SNR

The SNR (signal-to-noise-ratio) tests described in Section 3.1 above, lead to the results as summarized in Table 5 below:

Table 5
IMPROVEMENT IN SIGNAL-TO-NOISE-RATIO (in dB)* via the various filter designs

	Improvement via Filter in dB*				
	TDAF**	FDAF**		HAF**	
Type of Noise		Low Noise	High Noise	Low Noise	High Noise
250- 400 Hz	+9.4	+20.6	+25.5	+15.1	
1200-1700 Hz	+6.1	+22.1	+25.5	- 3.8	
2400-3000 Hz	+9.0	+25.5		+10.35	
Babble	+9.3	+8.4	+6.7	+ 6.3	+ 2.6
Cafeteria	+7.6	+12.2	+16.2	+11.6	+12.7

*Signal-to-noise ratio (SNR) in dB is a logarithmic scale of $20 \cdot \log_{10}$(signal/noise). Hence a signal-to-noise ratio of 1 is an SNR of 0 dB. A ratio of 2 is an SNR of 6 dB, a ratio of 3 is SNR of 10 dB and a ratio of 10 is 20 dB, etc.
**TDAF: time-domain adaptive filter; FDAF: frequency-domain adaptive filter; HAF: Hearing-aid adaptive filter.

In Table 5 comparison is made between the TDAF and the FDAF of Section 2, and whose design is given in App. A and B, respectively, the RDAF employing an 8 parameters Wiener filter, i.e. m+1=8 in Eq. (8) of Sect. 2.1, and the FDAF similarly employing q=8 in Eq. (11) of Sect. 2.2. Also, comparison is made with a hearing-aid-applied adaptive filter (HAF), which is a constrained version of the present designs, where only q=3 parameters are employed, (up to 5 for some noise situations), and which was further simplified to allow its realization in a below 250 microwatt, single-CMOS chip employing a 1.1 to 1.4 VDC supply (conventional hearing-aid battery), for use in conventional behind-the-ear or inside-the-ear hearing aids.

Table 5 clearly illustrates the superior performance of the FDAF version in all situations. Observe that the accuracy of the measurements is ±1dB. Hence, even in the babble-noise situation, where the FDAF is the weakest, as can be expected (see our discussion on that noise in Sect. 3.2 above), it performs almost as well as the second best design for that noise, namely the TDAF design. In other situations, all of which are of importance to the navy, probably as much or more than babble noise, the FDAF outperforms the HAF in every category, as could be expected, by anything from 1 dB to 29 dB. Finally, it improves on "no-filtering" by 6.7 to 25.5 dB, considering all the noises tested.

Table 6
SNR Improvement for TDAF of Various Orders
Noise 1700-2400 Hz/speech & noise test
SNR at input (unfiltered): -11.8 dB

Model order m+1	SNR improvement via filtering (dB)
4	5.8
6	6.3
8	5.8
10	1.8

Table 6 is enclosed below to justify our

choice of order of m+1=8 for the TDAF design. Indeed, if the same best performance is achieved for several orders, the lowest of these orders should be chosen. However, for better resolution in cases of very narrow-band noises, we chose m+1=8 rather than 6. In practice such very narrow band environmental noises are rare. If they occur and a somewhat wider band is thus affected by filtering, intelligibility is hardly affected, as is illustrated in Table 7 of Sect. 3.4 below. Hence a choice of m+1=8 is considered as a good compromise, to adequately be valid even if noise is very narrow band in one or several spectral regions.

3.4 Performance Results - Intelligibility Scores

The intelligibility test resuts, performed as described in Sect. 3.2, are summarized in Table 7, when using mono-syllabic NU-6 (Northwestern Univ.), word-lists #1 to #4, Forms A to D.

Table 7
FDAF: Intelligibility Test Scores:
NU Mono-syllabic Word List

Type of Noise	FDAF Unfil*	FDAF Fil**	HAF(1) Unfil	HAF(1) Fil.
250- 400 Hz	28%	72%	44.2%	73.6% (2)
1200-1700 Hz	36%	48%	52 %	64 % (3)
	16%	36%	72 %	72 % (4)
2400-3000 Hz	68%	74%	92 %	100 % (5)
	24%	36%	40 %	58.4% (6)
Babble	26%	30%	36.8%	56 %
Cafeteria	32%	90%	38.4%	68.4%

(% of fully correct word recognition)

*Unfiltered **Filtered

(1) For people with normal hearing, results from [1] on same HAF
(2) Test for 600-800 Hz noise
(3) Tests by R. LaRose, on same HAF, Noise of 1200-1400 Hz
(4) Test by R. LaRose, on same HAF
(5) Test by R. Larose, on same HAF
(6) Test for 1700-2400 Hz

At no-noise, the score with the same word lists (FDAF) was not 100% but only 90%, since, when using this monosyllabic words, tape, in 10% of the words a 1-phoneme error occurred when even when no noise was present. The filter is, of course, completely transparent at no noise conditions, such that speech is unaltered by the filter at no-noise. Hence, the latter no-noise score was independent of the filter being on or its being by-passed.

For correct interpretation of Table 7, all scores for the FDAF must thus be multiplied by a correction factor of 1.11 (i.e., for cafeteria noise, the score was 35% unfiltered and 100% filtered). A similar correction, by of 1.035 is required for the HAF filter, since the HAF tests were done with a higher quality play-back system. Note that no such intelligibility tests were possible with the TDAF design, since this system is not running in real time, as was discussed in Sect. 2 above.

The intelligibility test results above indicate again that the FDAF outperforms the HAF design in all categories but for babble noise. We comment that in the case of babble noise using the FDAF, 56% of words were recognized with either no error or with one phoneme being wrong (say, "nag" vs. "nap"), whereas without filter, only 42% were similarly recognized in the babble noise (FDAF) test tabulated in Table 7. It is important to observe that there is no exact correspondence betwen SNR improvement and intelligibility improvement, though they go in the same direction. When comparing Table 5 (SNR performance) with Table 7 (intelligibility scores), we note that FDAF outscored HAF in all noises in Table 5, and it outscored HAF in all but babble noise, in intelligibility scores. Furthermore, whereas FDAF yielded improvements vs. no-filtering at all, in every category both in Table 5 and 7 (SNR and intelligibility) HAF yielded improvement in every intelligibility score category in Table 7, but was slightly inferior (3.8 dB) in the 1200-1700 Hz SNR test, though yielding improvement in every other SNR test. Interestingly, HAF yielded improved intelligibility even in the 1200-1700 Hz range.

4. Conclusions

In conclusion we observe that the present FDAF design is beyond doubt the best of the designs considered, both with respect to its improvement in SNR which is outstanding in every category (from an improvement by 25.5 dB in SNR vs. no filtering, for both 1200-1200 Hz and 2400-3000 Hz noises, to a 6.7 to 8.4 dB improvement for the most difficult noise to filter, i.e., babble noise-see Table 5 of Sect. 3.3), and with respect to its improvement in single-phoneme word-list intelligibility scores (up to an improvement for Cafeteria noise from 32% unfiltered to 90% filtered, noting that at no-noise conditions intelligibility for same list is only 90%!) The TDAF also outperformed, on the average, the simplified HAF design in SNR tests. However, the FDAF design is the obvious choice, outscored the simplified HAF, by improving on the HAF performance by an additional 29 dB for 1200-1700 Hz (though even for this noise, the HAF's intelligibility scores show doubling the score vs. "no-filtering"), by an additional 10 dB for 250-400 Hz noise and 2400-3000 Hz noise, and by an additional 2 to 4 dB for babble noise.

Regarding the intelligibility tests with single-syllable words, it is obvious that any of the intelligibility-score during normal conversation, or even with single short sentences, would be far superior when the equivalent score (same conditions otherwise) with single-syllable word list that are also unrelated in content.

We also comment that, though not quantitatively measurable, all tests clearly pointed to a very considerable improvement in listener's comfort while using the filter when in a noisy environment, relative to the unfiltered condition. This listener-comfort is a most important aspect in listener-fatigue under noise, which is of great relevance to military situations requiring high degrees of concentration of pilot, etc.

The FDAF design tested was a real-time on-line design, which lends itself to a portable (flyable on board of a helicopter or fighter plane) realization.

5. References

(1) Stein, L. and Dempsey-Hart, D. "Listener-Assessed Intelligibility of a Hearing-Aid Self-Adaptive Noise Filter," Ear and Hearing Jour. (Jour. of the Amer. Audiologic. Soc), July 1984.

(2) Graupe, D., Time Series Analysis, Identification and Adaptive Filtering, Krieger Publishing Co., Malabar, FL, 1984

(3) Levinson, N., "The Wiener RMS Error Criterion in Filter Design and Prediction," App. B in: Wiener, N.: Time Series, MIT Press, Cambridge, Mass, 1949.

(4) Tsypkin, Y. Z., Foundations of the Theory of Learning, Academic Press, New York 1973.

(5) Widrow B., Glover, T. R., McCool, T. M., Kaunitz, J., Williams, C. S., Hearn, R. H., Zeidler, T. R., Doug. E., and Goodlin, R. C.: "Adaptive Noise Cancelling, Principles and Applications," Proc. IEEE, Vol. 63, pp. 1692-1716, Dec. 1975.

(6) Hodgkin, W. R. and Skinner, R. W.: Hearing Aids Assessment and Use in Audiological Rehabilitation, Williams & Wilkins Publ Co., Baltimore, 1977.

84-2655

A FREQUENCY HOPPED DEMAND ASSIGNED MULTIPLE ACCESS SCHEME

Kuriacose Joseph and Stephen S. Rappaport

State University of New York at Stony Brook
Department of Electrical Engineering
Stony Brook, New York 11794

Abstract

A DAMA scheme is presented in which users gain access to a CDMA message channel via randomly selected Frequency Hopped Multiple Access (FHMA) request channels. The number of message channels that can be assigned is limited to a maximum value in order to limit error probabilities due to noise and co-channel interference. The problem of synchronization of Frequency Hopped signals is considered and a theoretical performance analysis of the scheme is presented. User generated interference with white gaussian noise background is considered. Measures of the FHDAMA scheme performance evaluated were the maximum number of message channels, the message channel occupancies, the bandwidth utilization and the probability of channel acquisition.

I. Introduction

Frequency-Hopped communication systems are spread spectrum systems in which the spreading of the spectrum is achieved by hopping the frequency of the carrier signal at regular intervals in a predetermined hopping pattern. These waveforms have some properties which make them useful in multiple access systems. The inherent frequency diversity makes these systems resistant to frequency selective fading and also to tone jamming.

Coding schemes for Frequency Hopped Spread Spectrum Multiple access (FHSSMA) communications have received some attention in the literature [1,2,3]. Most codes are selected so that a given frequency occurs not more than once in a pattern of hops. Applications of FHMA schemes to mobile communication have also been presented. In [4] two coding schemes were presented in which data was transmitted using predetermined hop patterns. An analysis was carried out at high Signal-to-Noise ratios for bit error probabilities when a certain number of users were accessing the channels. An analysis of one of the coding schemes of [4] in a Raleigh fading environment at low signal to noise ratios was carried out in [5]. In the coding scheme presented in [6] data was transmitted by biphase modulation of the carrier by the digital signal. A comparison of the FHSS scheme with FM/channel reuse schemes in a cellular environment was also done.

In both [4] and [5] the analysis of the schemes were carried out assuming a certain fixed number of users on the channels. Synchronization between the transmitted frequency hopped patterns and the pattern detectors at the receivers was assumed. In an environment where users are randomly accessing the channels, synchronization is an important aspect of the detection and decoding problem. The number of users at any time on the channels is also a random quantity. In this paper a DAMA scheme is presented that uses FH channels for both request and message transmission. The random nature of the channel use is considered and a solution to the synchronization problem suggested.

*The research reported in this paper was supported in part by the U.S. National Science Foundation under Grant ECS 83-05642.

The structure of the overall scheme is similar to that of [8]. Requests for message channels are made over collision type request channels. If these requests are received successfully at a central processor and if message channels are available, a message channel assignment is made. Request channels presented in previous literature [9,10] were simple Aloha channels in which each request occupied a separate portion of the total bandwidth. A request was lost if there was an overlap in time between the transmissions of two requests using the same channel. When FHMA request channels are used there need not necessarily be a loss of information when two requests overlap.

Some problems arise due to the nature of the coding used in [4,5,6]. Since frequencies over the whole bandwidth are shared by all users, certain identifiable frequency hop patterns arise due to the transmissions of different users. This could lead to errors in decoding. It is seen intuitively and has been shown in [4,5] that errors of this origin increase with the number of users. Hence it may be desirable to limit the number of users at any time so that the probability of this kind of error is limited to a predetermined value. Demand assignment of the message channels is suggested.

In section II a brief discussion of the method of coding is given. In III a solution to the synchronization problem is presented and an analysis of the flow of requests in the system carried out. In section IV, the receiver at the central processor is described and in V an analysis of the overall scheme is carried out.

II. Coding Scheme

In the paper by Einarsson [4], a method of coding has been presented in which the interference between users of two different channels is minimized when these users are synchronized to each other at the chip or frequency hop level. This coding has been used in our DAMA scheme. A more detailed explanation of the coding can be found in [4].

Using this code, Q channels are generated from Q carrier frequencies. Each of the Q channels contains Q different hop patterns called symbols which can be used to transmit $\log_2 Q$ bits of data. The number of hops in each pattern, L, is the same for all channels and is predetermined to be anything from 2 to Q-1. The code is illustrated by an example with Q=5 and L=3. If the frequencies of the carriers are numbered 0 to 4, the 5 channels can be represented by the sequences of hops shown below.

Channel	Sequences Used
1	(000),(124),(245),(312),(431)
2	(111),(230),(304),(423),(042)
3	(222),(341),(410),(034),(103)
4	(333),(402),(021),(140),(214)
5	(444),(013),(132),(201),(320)

A frame is defined as the time period for which a symbol is transmitted. It is seen that the frequencies of the carriers transmitted on two different

channels coincide in only one chip of a given frame. This is called the one-coincidence property. It is also seen that the symbols of any one channel are cyclic or shortened cyclic shifts of each other. Hence two users of the same channel, out of frame sync with each other could transmit the same carrier frequency in anything from 1 to L hops of a frame. Since the coincidences between users of the same channel depend on the data transmitted, there may not be a loss of information in the event of a collision between them. This property of the channels could be used to an advantage when sending requests. Since requests occur at random times and involve the selection of random request channels, the possibility exists that two or more requests are sent on the same channel and that the times of transmission of these requests overlap. If the above method of coding is used, this need not necessarily lead to a loss of the transmitted information. Hence FH request channels are used in this scheme.

In our implementation, the frequencies of the request channel cover a bandwidth different from that covered by the frequencies of the message channel. Hence there is no interference between the two.

III. Synchronization and Flow of Requests in the Scheme

Most literature on FHMA mobile communication has not dealt with the crucial aspect of frame synchronization in randomly accessed systems.

In this scheme we assign one of the Q symbols in a channel as the "channel recognition symbol". This symbol is used only to signal start of transmission using a certain channel. No data is transmitted through this symbol and frame synchronization of the symbols that are transmitted subsequently is based on its detection. Information is transmitted using the Q-1 symbols that are left in a channel. For the given example, the symbols 000, 111, 222, 333 and 444 could serve as the channel recognition symbols of the respective channels. The number of bits of data transmitted per symbol now is $\log_2(Q-1)$.

Let ℓ_1 and ℓ_2 be the number of bits of information per symbol on the request and message channels respectively and L_1 and L_2 denote the number of hops per symbol on these channels.

The coding scheme used in the design of the channels demands that the number of frequencies used must be a prime number or a power of a prime number. If ℓ_1 and ℓ_2 are parameters that are given, Q_1 and Q_2, the number of frequencies in the request channels and message channels respectively are chosen to be the smallest integers that satisfy the conditions

$$Q_1 \geq 2^{\ell_1} + 1 \text{ and } Q_2 \geq 2^{\ell_2} + 1, \qquad (1) \text{ and } (2)$$

and the condition of being prime or a power of a prime number. In the further analysis of the scheme we approximate (1) and (2) by the equalities.

The structure of the overall DAMA scheme is shown in Fig. 1. It represents the flow of requests and the assignment of message channels in the system channel assignment are made at a central processor that receives the requests from all the users simultaneously. A user wishing to gain access to a message channel begins by selecting a request channel at random and then transmitting the "channel recognition symbol" of that channel. This is followed by the transmission of the information that the central processor needs in order to assign a message channel. This is done by transmitting K symbols of information in the K succeeding frames where

$$K = \begin{cases} \left\lfloor \frac{N}{\ell_1} \right\rfloor + 1 \text{ if } N/\ell_1 \text{ is not an integer} \\ \frac{N}{\ell_1} \text{ if } N/\ell_1 \text{ is an integer.} \end{cases} \qquad (3)$$

N being the length of the request in bits of data. $\lfloor\ \rfloor$ is a sumbol representing the nearest integer less than the given real number.

The data transmitted on the request channel may include the user's identification, the intended final destination of the message and the length of the message to be transmitted. N was taken to be a constant in our analysis.

If a request is decoded successfully at the central processor and a message channel is available, an assignment is made. This message channel assignment is then broadcast over a single reverse broadcast channel to which all users listen. It is assumed that the probability of the user receiving its assignment over this broadcast channel is one. If all the message channels are being used no channel assignment is possible and no broadcast is made. The user then retransmits its request after a random waiting period.

The arrival times of new requests for message channels are assumed to be poisson distributed with λ_T being the average arrival rate of new requests in arrivals/sec.

If the delay in the retransmissions of unsuccessful requests are random and sufficiently long, the total arrival rate of requests at the central processor can also be considered poisson distributed. Then if λ_γ is the total average arrival rate of requests at the central processor in arrivals/sec, it can be shown that

$$\lambda_T = \lambda_\gamma \cdot P_{det} \cdot P_{nblo} . \qquad (4)$$

where P_{det} is the probability of successfully decoding a request and P_{nblo} is the probability of there being a message channel available.

At equilibrium, the rate at which new requests are made should be the rate same as the rate at which message channels are assigned.

Then occupancy of the message channels is given by

$$\rho_m = (\lambda_T/R_2)(N_m/C_m), \qquad (5)$$

where R_2 is the rate of data transmission on the message channels and N_m is the average length of a message in bits. C_m is the total number of message channels available.

From (5) total traffic in the message channels is given by

$$a_T = (\lambda_T/R_2) \cdot N_m . \qquad (6)$$

Using the Erlang B formula

$$P_{nblo} = 1 - B(C_m, a_o) , \qquad (7)$$

where

$$B(s,a) = (a^s/s!) / \sum_{k=0}^{s} a^k/k! \qquad (8)$$

and a_o is the traffic that impinges on the central processor

$$a_T = a_o(1 - B(C_m, a_o)) = a_o \cdot P_{nblo} \qquad (9)$$

Using (4), (6) and (9),

$$a_o = (\lambda_\gamma/R_2)\cdot P_{det}\cdot N_m \qquad (10)$$

C_m, the number of message channels, is chosen so that the bit error probability in these channels is limited to a certain value.

A measure of performance of a DAMA scheme is the overall bandwidth utilization S, which is given by,

S = (Occupancy of the message channels). (Fraction of the total bandwidth assigned to these message channels) (11)

$$S = \rho_m(1 - \nu) \qquad (12)$$

where ν is the fraction of the bandwidth assigned to request channels. The minimum request and message channel bandwidths required are given by $(Q_1\cdot R_1)$ and $(Q_2\cdot R_2)$ where R_i is the rate of data transmission in request channels.

If the data burst rates are the same in both the request and message channels, $R_1 = R_2$ (13)

and

$$(1-\nu) = Q_2/(Q_2+Q_1) \ . \qquad (14)$$

Using the approximations for Q_1 and Q_2 and (5), (12) and (14),

$$S = (\lambda_T/R_2)(N_m/C_m)[(2^{\ell_2}+1)/((2^{\ell_2}+1)+(2^{\ell_1}+1))] \qquad (15)$$

Another measure of performance of the DAMA scheme is P_{cha}, the probability of a user getting a message channel when a request is made. The expected number of transmissions before a message channel when a request is made. The expected number of transmissions before a message channel is assigned, N_T, given by the expression

$$N_T = 1/P_{cha} \ . \qquad (16)$$

$$P_{cha} = P_{det}\cdot P_{nblo} \ . \qquad (17)$$

P_{det} depends upon the structure of the receiver used at the central processor to decode requests and the protocols used in this decoding. Once this and the other parameters are calculated, the overall system can be analyzed.

IV. Detection and Decoding of Requests

The central processor receives requests from all potential users of the message channels. These requests are also corrupted by noise in the system which is assumed to be white gaussian. The problem at the receiver is to decode the symbols belonging to the different channels on a frame by frame basis.

The first stage in decoding a request is to detect the "channel recognition symbol" that is transmitted and associate it with the correct channel. In our scheme this is accomplished by a bank of noncoherent matched filters or correlators matched to this symbol. An example of a correlator bank matched to the pattern 000 is shown in Fig. 2.

Once the start of a request transmission is detected, a tracker is initiated which decodes the data transmitted in the K succeeding frames on this channel. This tracker decodes the request on a frame by frame basis and the overall performance of the receiver depends upon the approach used for decoding and also the protocols involved. Using noncoherent matched filters or correlators, a record is made of the frequencies of all those carriers used in the request channels that are detected during each chip of a given frame. A symbol belonging to the channel is now detected by the tracker if the pattern of all the frequencies corresponding to that symbol is detected in the frame.

Cochannel interference and noise could cause a symbol to be detected in a channel even if it was not transmitted by the user of interest. Let P_{fsy} denote this probability. A request transmission is assumed to be missed if the "channel recognition symbol" is not detected or if more than one symbol belonging to the given channel is detected in any one of the K frames of the request.

Requests that are made using different channels engage different trackers. Overlapping requests on the same channel, out of frame synchronization with each other, engage different trackers also. In the case where more than one request is detected using the same channel, in frame sync, and overlapping in transmission times, tracking of all these requests is done using the same tracker from the start of the first detected request to the end of the K frames of the last detected request. A timing diagram for the case of two such users is shown in Fig. 3. Each of the overlapping requests can be detected by decoding the K symbols detected from the start of that request transmission on the tracker. Using this arrangement it is seen that the maximum number of trackers needed to detect all possible requests in the receiver is $(2^{\ell_1} + 1)\cdot L_1$.

The thresholds for the noncoherent matched filters used in these trackers are chosen so that the probability of detection of a tone transmission approaches unity. The probability of not detecting a transmitted symbol tends to zero. This reduces the probability of missing a request transmission to the probability of detecting one or more false symbols belonging to the particular channel in a given frame due to noise and cochannel interference.

The probability of a symbol not being falsely detected is $(1-P_{fsy})$. When a symbol is being transmitted in a frame, the probability that no other symbol belonging to the same channel is detected in this frame is given by $(1-P_{fsy})^{2^{\ell_1}-1}$ since there are $(2^{\ell_1}-1)$ symbols left in the channel. Then P_{det} is the probability that no false symbol is detected in any of the k request frames.

$$P_{det} = [(1 - P_{fsy})^{2^{\ell_1}}]^K \qquad (18)$$

If two or more users access the same request channel within K frames of each other, in frame synchronization, more than one symbol belonging to that channel is detected in one or more frames of each request. Hence all overlapping requests are lost. This factor is accounted for in P_{fsy}.

A false alarm is the detection of a request transmission at the central processor when none is actually transmitted. In the occurrence of such an event, if the decoded request information makes sense, a message channel could be allocated to a non-existant user.

Prob. of getting exactly one false symbol belonging to a channel in a frame in the absence of an actual transmission in that channel is given by the expression $2^{\ell_1}\cdot P_{fsy}(1 - P_{fsy})^{2^{\ell_1}-1}$. The probability of getting exactly one false symbol in K successive

frames is then $[2^{\ell_1} \cdot P_{fsy} \cdot (1-P_{fsy})^{2^{\ell_1}-1}]^K$.

If P_{fs} is the probability of detecting a "channel recognition symbol" when one is not transmitted, P_{fa}, the probability of a false alarm is given by

$$P_{fa} = P_{fs} \cdot [2^{\ell_1} \cdot P_{fsy} \cdot (1-P_{fsy})^{2^{\ell_1}-1}]^K . \quad (19)$$

For the protocols used, it was found that P_{fa} was very small at the traffic levels of interest. When P_{fsy} is small at the higher signal to noise ratios, P_{fa} is obviously reduced. At lower signal to noise ratios, even if P_{fsy} is higher, the probability of getting exactly one false symbol in a frame is small and the probability of this occurrence for K succession frames is significantly lower. Even when false requests are detected, there are parity checks for the overall request data transmitted. The probability of these matching is very small for false requests. Hence the effects of false alarms on the overall system for the protocol chosen is found to be negligible.

Derivation of expressions for P_{fsy} and P_{fs}

Using the definition of arrival rates introduced earlier, average arrival rate of requests at the central processor per hop interval is

$$(\lambda_\gamma/R_1) \cdot (\ell_1/L_1) . \quad (20)$$

False symbols can be detected in a frame due to other users of the same channel who are in frame sync with our frame of reference. This could be caused by requests made in any one of the K preceeding frames, in frame sync with the frame of reference and using the same channel.

Using (20), the average arrival rate of such requests then is

$$K \cdot (\lambda_\gamma/R_1) \cdot (\ell_1/L_1)/(2^{\ell_1} + 1) . \quad (21)$$

The average arrival rate of requests that form a particular symbol in a given frame,

$$\lambda_x = K \cdot (\lambda_\gamma/R_1) \cdot (\ell_1/L_1)/((2^{\ell_1}+1) \cdot 2^{\ell_1}) \quad (22)$$

Using the poisson distribution, the probability of getting a particular symbol in the frame due to these requests is P_0 where

$$P_0 = 1 - e^{-\lambda_x} \quad (23)$$

The average arrival rate of all other requests over channels other than the one containing the symbol of interest and who could cause some frequency to be detected in a particular slot is given by the expression

$$(K+1) \cdot \ell_1 \cdot 2^{\ell_1} (\lambda_\gamma/R_1)/(2^{\ell_1}+1) . \quad (24)$$

using the same arguments as those used for (21).

Using the same approach as for (22) average arrival rate of these users who could cause a particular frequency out of $(2^{\ell_1}+1)$ possible frequencies to be detected in a particular chip

$$\lambda_y = (\frac{\lambda_\gamma}{R_1}) \cdot \ell_1 \cdot (K+1) \cdot \frac{2^{\ell_1}}{(2^{\ell_1}+1)} \cdot \frac{1}{(2^{\ell_1}+1)} . \quad (25)$$

$$P_{fsy} = P_0 + (1-P_0) \cdot P' \quad (26)$$

with P_0 given by (23)

P' is the probability of the false symbol being found in a frame but not due to other requests on the same channel, in frame synchronization with the frame of interest. Let P_1 denote the probability of a particular symbol being formed due to a combination of users of the same channel, one chip out of frame sync with the frame of reference and other users of the request channel as shown in Fig. 4.

Since symbols of a particular channel are cyclic or shortened cyclic shifts of each other, in the worst case, the requests on the same channel, one chip out of frame sync could transmit tones corresponding to the first (L_1-1) frequencies of the required symbol in the frame of reference. Using reasoning similar to that used for (23), this probability is $(1-e^{-\lambda_x})$.

The Lth frequency in the frame could be formed either by same channel requests whose frames begin at the 2nd, 3rd,.....,(L-1)th or Lth chip of our frame of reference, by requests on all other channels (which contribute to λ_y) or by the false detection of a frequency transmission due to noise present which is assumed to be white gaussian with one sided spectral density η.

In our analysis we assume that the detection and tracking of the tones transmitted in both the request and message channels is done using noncoherent matched filters. In order to ensure that a transmitted tone is nearly always detected, the thresholds for the filters are chosen so that the miss probability of the weakest possible user whose tone is to be detected in 10^{-5}. Then, using standard analysis techniques

$$10^{-5} = \int_0^b x e^{-(x^2+a^2)/2} dx \quad (27)$$

with $a=(2 \cdot (E_b/\eta) \cdot (\ell_1/L_1))^{1/2}$ where E_b/η is the signal-to-noise ratio of the weakest signal to be detected in energy per bit of data per hertz. b is the threshold of the matched filter and is the only unknown in (27). Using the value of b solved for in this equation, the probability of a false exceedence of a threshold, P_{fchip}, is given by

$$P_{fchip} = \int_b^\infty x e^{-x^2/2} dx \quad (28)$$

It can be shown that

$$P_1 = (1-e^{-\lambda_x})[(1-e^{-\lambda_x(L_1-1)}) + e^{-\lambda_x(L_1-1)} \cdot ((1-e^{-\lambda_y}) + e^{-\lambda_y} \cdot P_{fchip})] \quad (29)$$

Examining (26) further

$$P_{fsy} = P_0 + (1-P_0)[P_1 + (1-P_1)P''] \quad (30)$$

where P'' is the probability of the symbol being formed in a frame but not due to requests on the same channel in frame sync or one chip out of frame sync with the frame of reference.

Define P_i as the probability of the symbol being formed by same channel users i chips out of frame sync with the frame of reference. It has been shown that

$$P_{fsy} = P_0 + \sum_{j=0}^{L_1} P_j \left[\prod_{i=0}^{j-1} (1-P_i) \right] . \quad (31)$$

$$P_i = x_i [(1-e^{-\lambda_x (L_1-i)}) + e^{-\lambda_x (L_1-i)} (1-e^{-\lambda_y}) \cdot \left(\sum_{j=1}^{i-1} xy^{(j-1)+y^{(i-1)}} \right)] \quad (32)$$

with $x=1-e^{-\lambda_x}$ and $y=e^{-\lambda_x}((1-e^{-\lambda_y})+e^{-\lambda_y}P_{fchip})$. $x_i=x$ except when $i=L_1$ for which value $x_i=1$.

Similarly,

$$P_{fs} = P_{1s} + \sum_{j=2}^{L_1} P_{js} \left[\prod_{i=1}^{j-1} (1-P_{is}) \right] . \quad (33)$$

$$P_{is} = (1-e^{-\lambda_1}[(1-e^{-\lambda_1 (L_1-i)}) + e^{-\lambda_1 (L_1-i)} (1-e^{-\lambda_2}) \cdot \left(\sum_{j=1}^{i-1} x_s y_s^{j-1+y_s^{(i-1)}} \right)] \quad (34)$$

with $x_s=(1-e^{-\lambda_1})$, $y_s=e^{-\lambda_1}((1-e^{-\lambda_2})+e^{-\lambda_2}P_{fchip})$ and $\lambda_1=(\lambda_\gamma/R_1)(\ell_1/L_1)/(2^{\ell_1}+1)$ and $\lambda_2=(\lambda_\gamma/R_1)((k+1)\cdot\ell_1 \cdot 2^{\ell_1}+k\ell_1)/(2^{\ell_1}+1)^2$.

If as a design parameter we specify that the hop rates in the request and message channels are the same

$$L_1/\ell_1 = L_2/\ell_2 \quad (35)$$

Under this condition, the request and message channel frequency detector thresholds can also be chosen to be the same. Probabilities of false threshold exceedences are also the same on both channels.

Then, $P_b(e)$, the bit error probability in the message channels, using the detection algorithm of [5] is given by

$$P_b(e) = (2^{(\ell_2-1)}/(2^{\ell_2}-1))[1 - \sum_{j=0}^{(2^{\ell_2}-1)} (P_m(j)/(j+1))] \quad (36)$$

with $P_m(j) = \binom{2^{\ell_2}-1}{j} z^j (1-z)^{(2^{\ell_2}-1-j)}$

$z=(P_y+(1-P_y)P_{fchip})^{L_2}$ where $P_y=(1-(1-(1/(2^{\ell_2}+1)))^M)$

and M is the number of users on the message channels.

V. Analysis of the Overall Scheme

The aim of the design was to allocate frequency hopped message channels as efficiently as possible. Efficiency, depending on the system objectives could be measured by ρ_m, S, or P_{cha}. The parameters of the design to be decided on are l_1, l_2, L_1 and L_2. These are calculated using (35) and where applicable, the equation based on the bandwidth(W) and date rate(R) constraints.

$$W/R = (2^{\ell_1}+1)L_1/\ell_1 + (2^{\ell_2}+1)L_2/\ell_2 \quad (37)$$

N, N_m and E_b/η are characteristics of the communication system which are given.

For the above parameters, the maximum number of message channels, C_m, is chosen so that the worst cases $P_b(e)$ as given from (36) is less than 10^{-3}. The measures of performance were calculated assuming that the request strategy was working at the level where its throughput λ_{sys} was maximum.

$$\lambda_{sys} = \max_{\lambda_\gamma} (\lambda_T/R_1) \quad (38)$$

This optimum exists due to the nature of the request channels in which the number of requests that are lost due to collisions increases with the traffic levels. In the initial study ρ_m, S and P_{cha} were plotted as functions of L_1 for different values of l_1 and l_2 using (35) and other equations. Values of N=128 and N_m=2048 were used and the analysis carried out for different signal-to-noise ratios. This initial study concerned itself with an understanding of the variation in the performance with the design parameters. The results of this study can be found in [12]. Here, we have also presented an example of the design of a FHDAMA communication system which is designed to support a certain maximum number of users under certain conditions.

Figures 5 to 9 show the results of a study where bandwidth and data rate constraints of W=20 Mhz and R=32 kbps were used. The operating conditions correspond to a system with no external noise, N=128 and N_m=2048. Fig. 5 shows that the number of possible users increases with l_2 conditional to the changes in the hopping rates. For a given l_2, a system with l_1=8 has a lower hopping rate in the message channel than a system with l_1=7 and as a consequence, supports less simultaneously transmitting users. Calculations are based on (35), (36) and (37). From Fig. 6 it is seen that for the smaller values of l_2 the carried traffic is limited by the number of message channels whereas for the higher values of l_2 the limitation is due to the throughput of the request channels. The carried traffic could also decrease for higher values of l_2 due to lower hopping rates. Figures 7,8 and 9 show the variation of S, ρ_m and P_{cha} respectively with l_1 and l_2. These can be analysed based on the results in Figures 5 and 6.

To design a practical scheme a decision is first made on the maximum number of active users that need to be supported simultaneously and the traffic levels that are of interest. Combinations of l_2 and l_2 that satisfy this requirement are found and depending on the other parameters of interest to be optimized one of these combinations is selected for design.

Conclusion

A DAMA scheme using FHMA channels for request and message channels is presented. A method of synchronization of random requests at a receiver was introduced. The analysis was carried out for the case where several geographically separated users were communication or attempting to communicate with a central facility. Errors due to white gaussian noise and interference from other users were considered.

The protocols of the scheme were very stringent and although this reduced the probability of the detection of requests, it also reduced the level of false alarms. It is suggested that where error correction coding is possible, the protocols could be eased and probabilityes of detection improved.

References

1. C.R. Cahn, "Spread Spectrum Applications and State-of-the-Art Equipments," in Spread Spectrum Communications, AGARD Lecture Series No. 58, AD-766 916, 1973.
2. G. Solomon, "Optimal Frequency Hopping Sequences for Multiple-Access," Proceedings of the 1973 Symposium on Spread Spectrum Communications, Vol. 1, AD-915 952, pp. 33-35.
3. D.V. Sarwate and M.B. Pursley, "Hopping Patterns for Frequency-Hopped Multiple-Access Communications," Proceedings of the International Conference on Communications, 1978, pp. 7.7: 1-3.
4. G. Einarsson, "Address Assignment for a Time-Frequency-Coded Spread-Spectrum System," B.S. T.J., Vol. 59, No. 7, pp. 1241-1254, September 1980.
5. D.J. Goodman, P.S. Henry and V.K. Prabhu, "Frequency-Hopped Multilevel FSK for Mobile Radio," B.S.T.J., Vol. 59, No. 7, pp. 1257-1275, September 1980.
6. G.R. Cooper and R.W. Nettleton, "A Spread-Spectrum Technique for High-Capacity Mobile Communications," IEEE Transactions on Vehicular Technology, Vol. VT-27, No. 4, pp. 264-275, November 1975.
7. V.H. MacDonald, "The Cellular Concept," B.S. T.J., Vol. 58, No. 1, January 1979, pp. 15-41.
8. S.S. Rappaport, "Traffic Capacity of DAMA Systems Using Collision Type Request Channels," National Telecommunication Conference, Vol. 3, pp. 68:1, 1-6, December 1977.
9. D. Raychaudhuri, "Multiple Access Communications Using Tree Codes," Ph.D. Dissertation, S.U.N.Y. at Stony Brook, pp. 75-93, December 1978.
10. G.L. Choudhury and S.S. Rappaport, "Cellular Communication Schemes Using Generalized Fixed Channel Assignment and Collision Type Request Channels," IEEE Transactions on Vehicular Technology, Vol. VT-31, No. 2, pp. 53-65, May 1982.
11. A.D. Whalen, "Detection of Signals in Noise," Academic Press, New York, 1971.
12. K. Joseph and S.S. Rappaport, "A Frequency Hopped Demand Assigned Multiple Access Scheme for Mobile Communication," Technical Report, S.U.N.Y. at Stony Brook, Department of Electrical Engineering, available from authors on request.

FIG 1 FLOW OF REQUESTS THROUGH THE FH DAMA SYSTEM

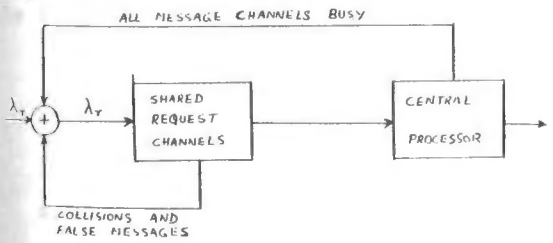


FIG 2 A BANK OF NONCOHERENT MATCHED FILTERS MATCHED TO THE FREQUENCY PATTERN 000 AND USED FOR THE DETECTION OF THE 'CHANNEL RECOGNITION SYMBOL'

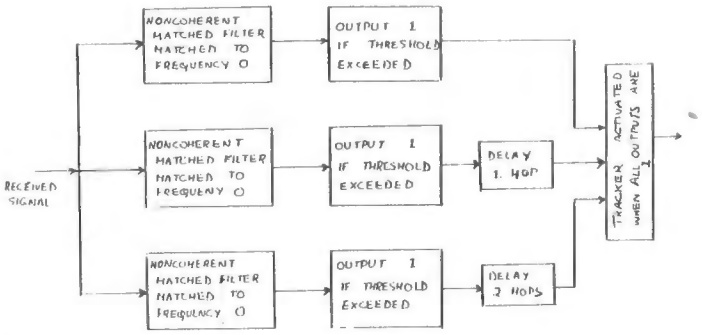


FIG 3 TIMING DIAGRAM FOR TRACKER USE WHEN TWO REQUESTS ON THE SAME CHANNEL OVERLAP IN TIME, IN FRAME SYNCHRONIZATION WITH EACH OTHER

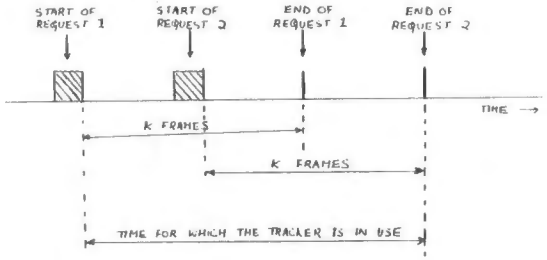
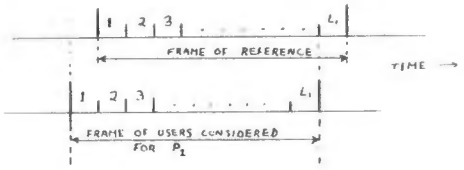


FIG 4 TIMING DIAGRAM FOR THE RELATIVE PHASE OF INTERFERING USERS CONSIDERED FOR P_1 WITH RESPECT TO THE FRAME OF REFERENCE



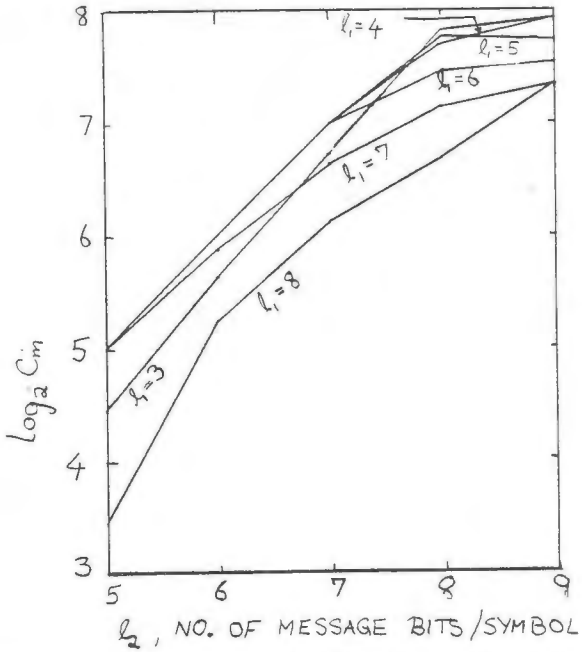


FIG. 5 MAXIMUM NUMBER OF MESSAGE CHANNELS WHEN $W=20MH_3$ AND $R=20K$ bps

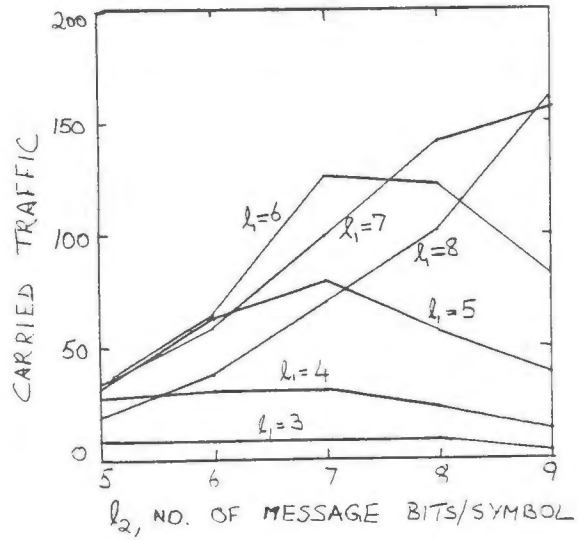


FIG. 6 MAXIMUM CARRIED TRAFFIC WHEN $W=20MH_3$ AND $R=20$ kbps

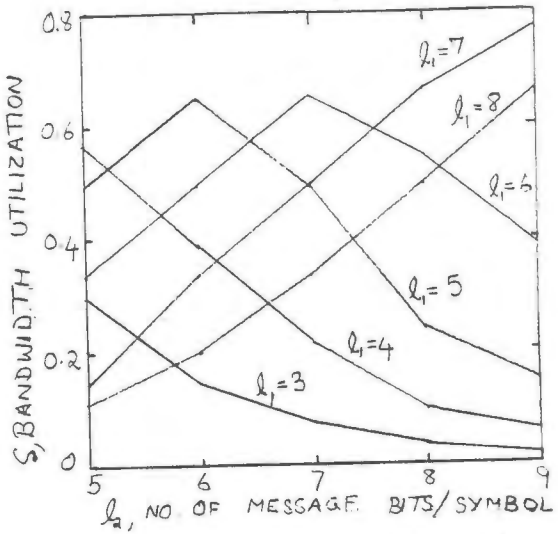


FIG. 7 MAXIMUM BANDWIDTH UTILIZATION FOR GIVEN VALUES OF l_1 AND l_2

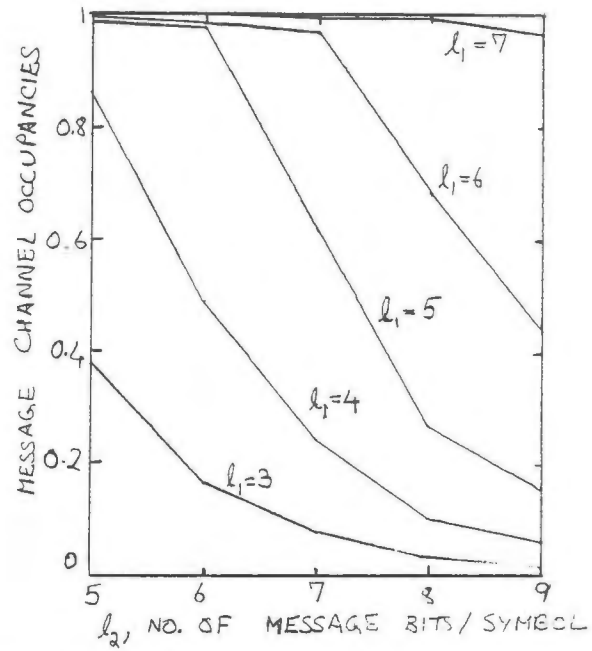


FIG 8. MAXIMUM MESSAGE CHANNEL OCCUPANCIES FOR GIVEN l_1 AND l_2.

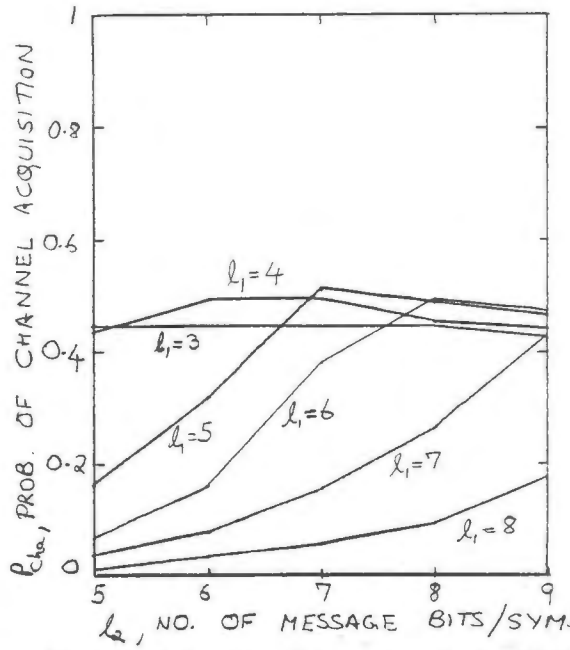


FIG 9 PROB. OF CHANNEL ACQUISITION FOR MAXIMUM THROUGHPUT OF REQUESTS

ENHANCED TCAS II SIGNAL PROCESSOR DEVELOPMENT

B. James Lyons
Margaret A. Martin

Allied Bendix Aerospace
Bendix Communications Division
Baltimore, Maryland 21204

Abstract

A methodology for computer simulation aided hardware design has been developed and was used successfully in the design and systems integration of the Enhanced Traffic Alert and Collision Avoidance System (TCAS II). Both hardware (low-level) simulation and statistical function (high-level) simulation were used to guide and finalize the hardware design. The low-level simulation (LLS) was needed to guide the detailed hardware design of the signal processor. However, the LLS was three orders of magnitude too slow to test the integration and functioning of the signal processor with the total Enhanced TCAS II system. A high-level simulation (HLS), which provided the same statistical response as the low-level simulation, was written and tested. The HLS did not provide the same response as the LLS to a given ATCRBS or Mode S reply signal, but the reply response (range, angle, altitude, amplitude measurements) to a statistically significant set of replies was comparable for the two simulations.

Three parallel efforts directed toward development of the LLS, the HLS, and hardware design and test were coordinated to yield the finalized TCAS signal processor design. Hardware test results show that the signal processing and decoding performance was predicted by the LLS. Flight tests results with the collision avoidance system show that the HLS functioned to predict the system performance.

Design Methodology

The signal processor development leading to the tested hardware was accomplished by following twelve ordered steps. There was, of course, iteration of these steps so that the results from the low-level simulation, the high-level simulation, or the hardware design could cause changes in previously completed steps. This paper consists of a brief discussion of the procedure and results of the following 12 development steps:

1. Formulate Functional Block Diagrams
2. Develop LLS Specification Flow Diagrams
3. Write LLS Programs
4. Run LLS Program and Obtain Preliminary Signal Processor Evaluation
5. Develop Preliminary Hardware Design
6. Finalize LLS Program and Obtain Signal Processor Performance Evaluation
7. Write HLS Specification and Generate HLS Statistical Tables Using LLS
8. Write HLS Program
9. Integrate HLS Program into TCAS II System Simulation and Obtain Test Results
10. Finalize Hardware Design
11. Fabricate and Test Signal Processor
12. Flight Test Signal Processor in TCAS II Sytem

Formulate Functional Block Diagrams

As an aid to understanding the functional interface of the signal processor in the TCAS II system, Figure 1 is provided showing the total system block diagram. The interrogator processor for handling Mode S and ATCRBS signals is represented by block number 3 of Figure 1. Figure 2 is the signal processor block diagram, which was initially configured from FAA requirements on TCAS operation. The video quantizer and digitizer form our best estimate of the leading edge position of all true reply pulses even when they are overlapped by other pulses. The Mode S and ATCRBS processors decode the replies (DABS and the more recent Mode S nomenclature are used interchangeably in this paper). Azimuth, range, amplitude, and data from the decoder are passed through a buffer to an 8086 microprocessor. Data to/from the system central computer is passed through the IEEE 796 multibus.

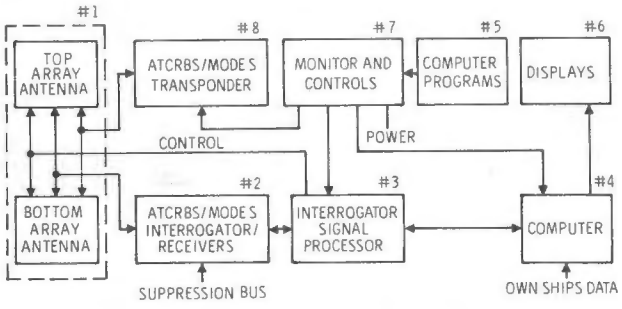


Figure 1. TCAS II System Block Diagram

LLS Specification Flow Diagrams

A low-level simulation (LLS) specification was generated from the block diagram and FAA requirements. The LLS is a computer simulation of the hardware functions to be designed. This simulation is self-sufficient in that it includes subroutines for RF signal generation, IF filtering, the log receiver, and monopulse measurement, as well as the functions in the signal processor. Hence, the LLS was used to aid the monopulse receiver design, as well as signal processing. As an example of the type of information provided by the LLS specification, the video quantizer part of the total specification is included in Figure 3. Footnotes on the dynamic minimum triggering level (DMTL) function are also shown for ATCRBS and DABS. Figure 3 is one of five sheets of specification flow diagrams generated during the signal processor development.

Write LLS Programs

The LLS specification flow diagrams (Figure 3) are modularized into functional blocks that "group" related logical operations. Each block, or module, corresponds to at least one top-level subroutine within the main calling program. The subroutines were executed iteratively, with each iteration corresponding to one delta t of simulation time.

Figure 4 shows the block diagram resulting from the modularization of the ATCRBS/DABS LLS specification flow diagrams. The following gives a brief description of the functions included in each of the blocks in Figure 4.

SIGNAL + NOISE: Generate all ATCRBS and DABS signals, with noise, for one delta t increment.

BUTTERWORTH FILTER: Filter the signal received from both sum and difference beams.

MONOPULSE PROCESSING: Obtain azimuth monopulse measurement for one delta t and pass measurement through delay line to maintain synchronization with digitized data.

LOG RECEIVER: Pass signal sample through math model of log receiver.

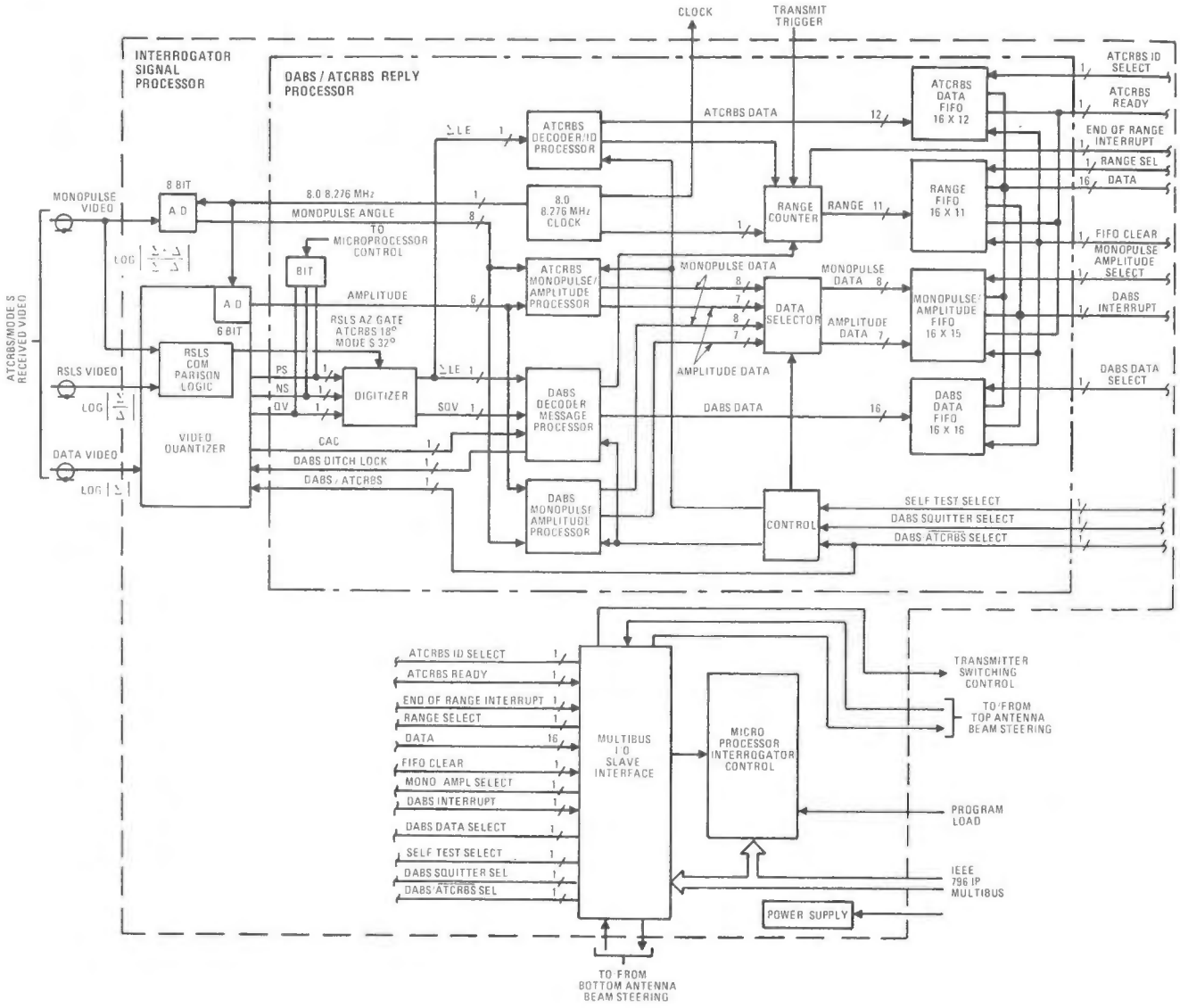
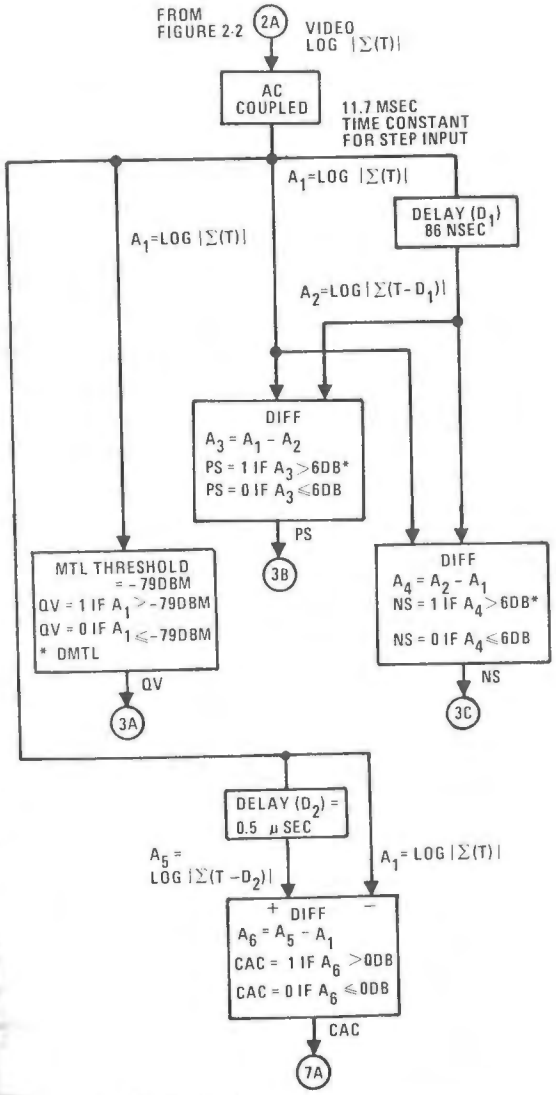


Figure 2. Interrogator Signal Processor

AMPLITUDE PROCESSING: Obtain amplitude measurement and pass measurement through delay line to maintain synchronization with digitized data.

COUPLING CAPACITOR: Pass signal sample through math model of coupling capacitor.

VIDEO DELAY: Pass signal sample through 86-ns video delay line.



* DMTL (ATCRBS)
IF A_1 >(MTL THRESHOLD) + (DITCHLOCK LEVEL)
SET:
$MTL_{NEW} = MTL_{OLD} + (A_1 - (MTL_{OLD} + DITCH\ LOCK\ LEVEL)|$
HOLD MTL_{NEW} FOR 22μSEC UNLESS
A_1 MTL_{NEW} + (DITCH LOCK LEVEL) IS REQUIRED
IN THIS CASE RESET MTL_{NEW} AS BEFORE
AND RENEW LOCK FOR 22μSEC.
DITCHLOCK LEVEL = 12DB

* DMTL (DABS)
DITCH LOCK LEVEL = 6DB
LOCK 3μSEC ON SINGLE PULSE
SET DITCHLOCK ON P4 PULSE OF MODE S
DECODE – DITCHLOCK 60μSEC
FOR PREAMBLE DECODE

Figure 3. Video Quantizer Flow Diagram

VIDEO PULSE QUANTIZER: Determine PS, NS, QV, and CAC pulse samples.

DIGITIZER: Determine actual leading and trailing edges, SQV, and insert ELE's as required.

ATCRBS DECODER: Pass ATCRBS signal leading edge pulses through 101.5-micros delay line; check for garble types 1, 2, and 3 in the before (B) and after (A) sections of the delay line; get "phantom" status. Get validated amplitude and azimuth data.

MODE S PREAMBLE DECODER: Pass MODE S leading edge pulses into preamble detector, and upon detection enable, synchronize CAC, amplitude and azimuth data.

MODE S DECODER: Receive synchronized CAC, amplitude, and azimuth data; validate CAC pulses; determine amplitude and azimuth values; decode text and address bits.

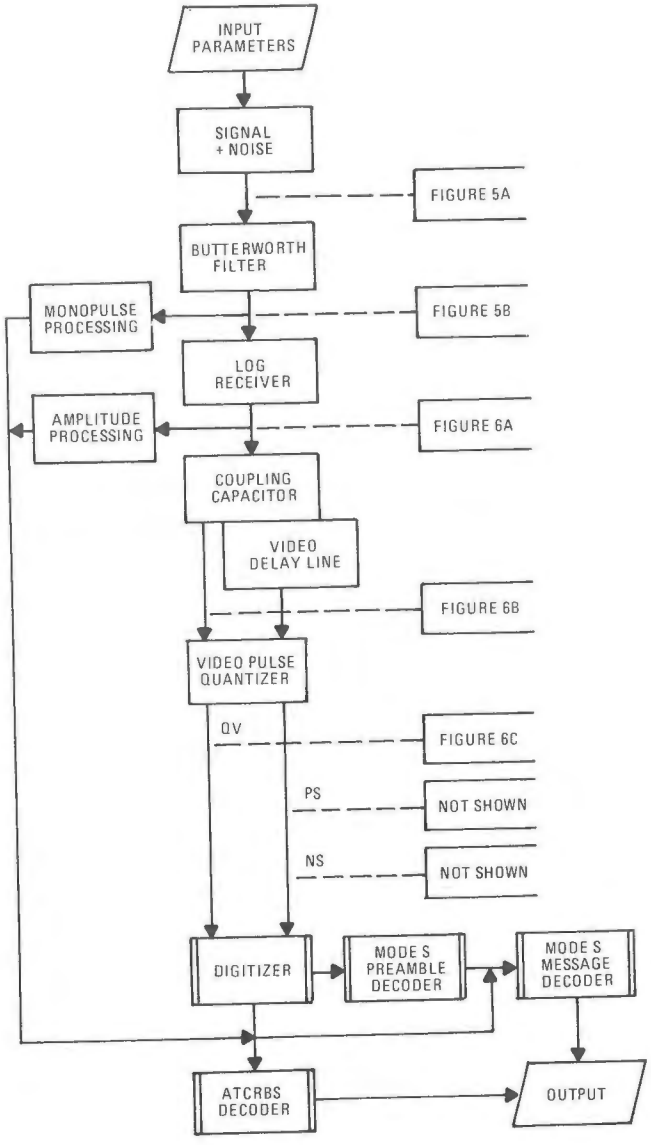


Figure 4. Low-Level Simulation Block Diagram

Test LLS Program

Upon completion of the coding and testing of all LLS modules, they were linked in the main LLS program for testing of the total signal processor LLS program. The first concern in testing the LLS program was to detect any problems in the timing and interactions between the modules. Included in the tests performed during these preliminary stages is the generation of a single pulse and the study of processing results as that pulse is passed through the modules of the simulation.

A single pulse, with 75 nanoseconds 10 to 90 percent rise time and 135 nanoseconds 90 to 10 percent fall time and a 3 dB width of 0.45 microseconds, was passed through the system and the processing results after each stage of Figure 4 were recorded.

The first stage output, corresponding to an input pulse amplitude of -50 dBm, is shown in Figure 5A. The pulse amplitude is -39 dBm, with noise set to -96 dBm after the Butterworth filter. Figure 5B shows the output of the Butterworth filter after receiving the signal plus noise. The filter is a four-pole Butterworth with a 3-dB bandwidth of 8 MHz.

The log amplifier output, the video pulse quantizer (VPQ) input, and part of the VPQ output are shown in Figures 6A thru 6C.

Develop Preliminary Hardware Design

The LLS specifications were used to guide the preliminary hardware paper design, as well as guide the programming of the low-level simulation. Figure 7 shows the preliminary ATCRBS decoder design. The function of the decoder is to detect replying ATCRBS transponder bracket pulse pairs (20.3-microsecond separation) and to decode the data pulses between the bracket pulses. Bracket decoding occurs at the shift register taps marked F1 and F2. Outside the F1 and F2 taps are taps B1 thru B28 and A1 thru A28. The function of the outside taps is to detect garble and phantoms. A subject reply with bracket pulses at F1 and F2 is said to be garbled if an overlapping reply is detected at a spacing that could cause the data pulses to be erroneously decoded. Another function of the shift register taps outside F1 and F2 is to determine if the subject reply with bracket pulses at F1 and F2 is false (phantom). The phantom bracket decode may be caused by two closely spaced replies.

Finalization of LLS and Performance Evaluation

The performance evaluation of the signal processor using the LLS provided results which were used to modify the LLS specification and program. Also contributing to the finalization of the LLS was feedback from the preliminary hardware design of the signal processor. Figure 8 shows the LLS sensitivity curve for detection of F1 and F2 pulse pairs of an ATCRBS reply (ATCRBS Bracket Detection). The detection threshold was set at -79 dBm to obtain the test results illustrated in Figure 8.

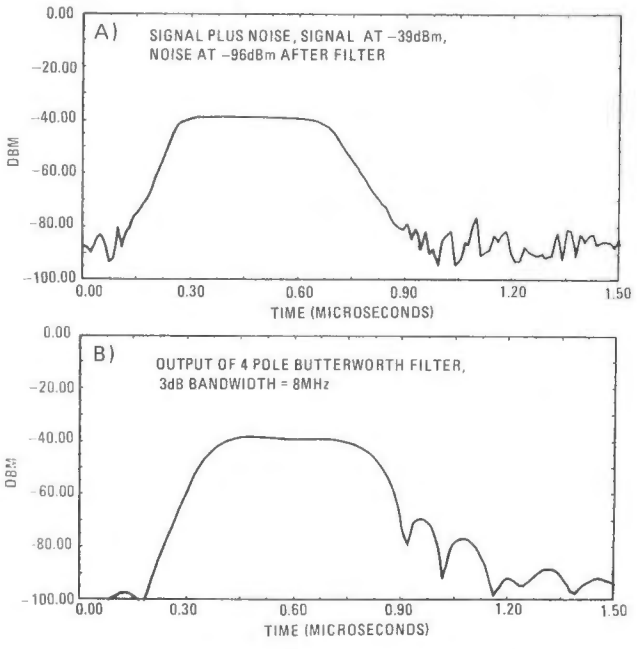


Figure 5. Signal Generator (Without and With Noise) and Filter Output

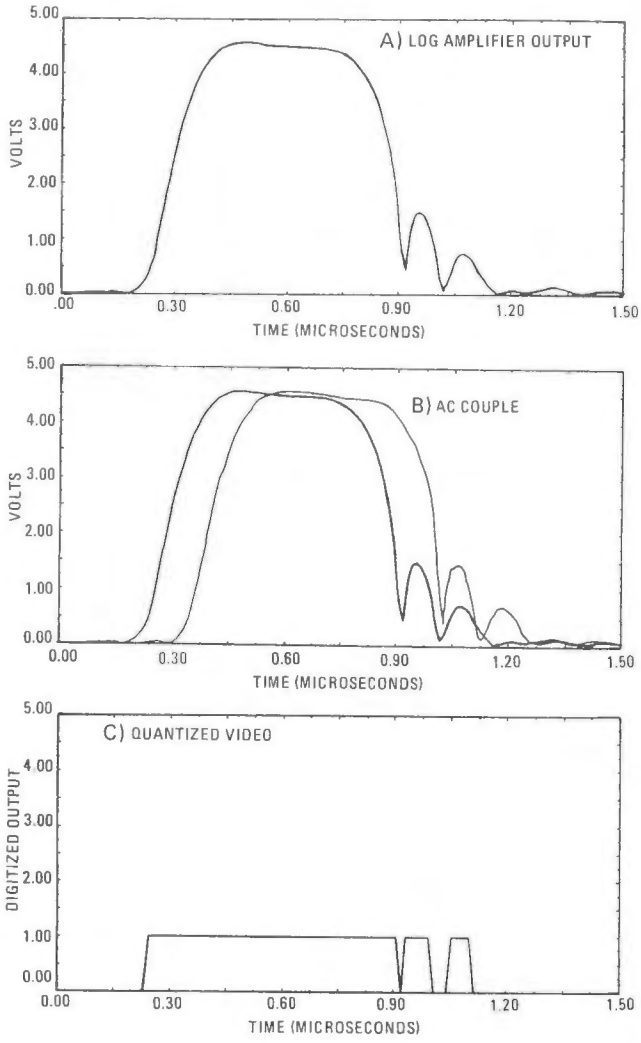


Figure 6. Log Amplifier, Delayed and Undelayed AC Couple, and Quantized Video Output

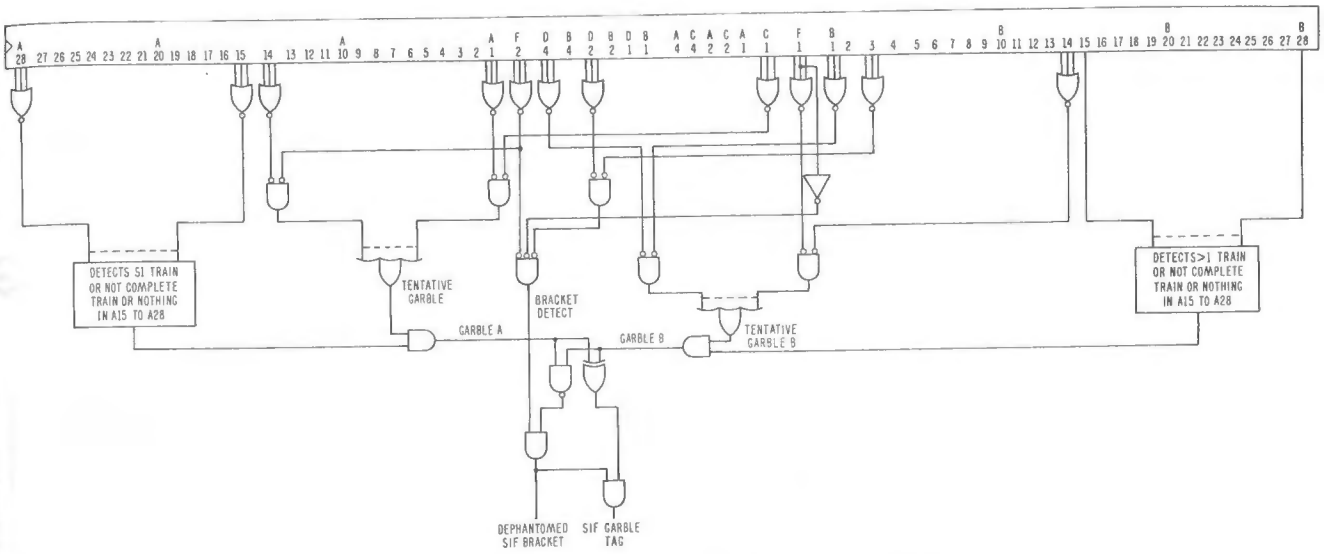


Figure 7. Preliminary ATCRBS Decoder Design

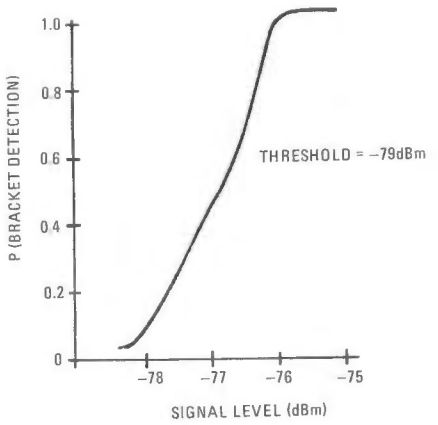


Figure 8. Probability of ATCRBS Bracket Detection for -79 dBm Threshold

The 2-dB increase in signal level required to change the probability of bracket detection from 10 to 90 percent is characteristic of high quality reply receivers and decoders. The slope of this curve is primarily dependent upon noise interference with the signal detection.

Write HLS Specification Flow Charts and Generate HLS Statistical Parameters Using LLS

The LLS was found to be three orders of magnitude too slow to function with the total TCAS II system simulation. This was because every second of data simulated through the signal processor required processing of 83 million samples of the reply signals. In order to test the collision avoidance function of TCAS II through the Los Angeles environmental model, it became necessary to write a second program for the signal processor. This program was called the High Level Simulation (HLS). The HLS responded statistically the same as the LLS. The results for a given reply of the LLS and the HLS were not the same, but over a sufficiently large set of replies the response of both programs was the same.

Specification flow charts for the HLS were written. These flow charts showed the form of the statistical functions that were required to complete the HLS. The LLS was used to generate the actual statistical function. Variables and ranges of parameters were included in the specification, but statistical tables defining the detection performance for a given level of reply in a given density of interfering replies was provided by the LLS. Statistical results for the HLS were collected in computer data base lookup tables. The data base format, in terms of tables of matrices necessary to run the HLS, was defined. Each table corresponds to a particular HLS decision box and gives the answer to this decision box as a function of the reply environment. For example, the ATCRBS reply environment is defined in terms of: signal amplitude case, number of replies on the 101.5-microsecond ATCRBS decoder shift register delay line at the time of the subject decode, number of simultaneous replies (overlapping, including subject), and the amplitude spread between the strongest overlapping signal and the subject signal.

A sample of one of these decision box tables is shown in Table 1. Table 1 gives the initial portion of the decision box tables used to decide probability of valid bracket detection for a signal whose amplitude lies between -50 and -70 dBm (called the -60 dBm signal center case). The first matrix appearing in this table is labeled "# on line=1". This means that at the time of the subject decode, the number of replies on the 101.5-microsecond shift register delay line was one, namely, the subject. Therefore, there could not have been any overlapping signals, and so the data in columns 2 thru 7, which corresponds to one to six signals overlapping the subject, are filled with zeros. Column 1 corresponds to one simultaneous signal, which means a clear subject. Data only appears in the first row of this column because a clear signal has no overlapping signal, and so the amplitude spread between the subject and strongest overlapping reply corresponds to the weakest dB index (-15, -10). The probability of valid bracket detection is 0.96 for this subject environment.

TABLE 1. -60 dB SIGNAL CENTER: PROBABILITY OF VALID BRACKET DETECTION

ON LINE= 1

	1	2	3	4	5	6	7
[-15,-10)	0 96	0.00	0	00	0	00	0 00
[-10, -5)	0 00	0 00	0	00	0	00	0 00
[-5, 0)	0 00	0 00	0	00	0	00	0 00
[0, 5)	0 00	0 00	0	00	0	00	0 00
[5, 10)	0 00	0 00	0	00	0	00	0 00
[10, 15)	0 00	0 00	0	00	0	00	0 00

ON LINE= 2

	1	2	3	4	5	6	7
[-15,-10)	0 95	0 94	0	00	0	00	0 00
[-10, -5)	0 00	0 87	0	00	0	00	0 00
[-5, 0)	0 00	0 95	0	00	0	00	0 00
[0, 5)	0 00	0 89	0	00	0	00	0 00
[5, 10)	0 00	0 86	0	00	0	00	0 00
[10, 15)	0 00	0 23	0	00	0	00	0 00

ON LINE= 3

	1	2	3	4	5	6	7
[-15,-10)	0 91	0 94	0	75	0	00	0 00
[-10, -5)	0 00	0 94	1	00	0	00	0 00
[-5, 0)	0 00	0 94	0	86	0	00	0 00
[0, 5)	0 00	0 93	0	83	0	00	0 00
[5, 10)	0 00	0 83	0	88	0	00	0 00
[10, 15)	0 00	0 27	0	22	0	00	0 00

ON LINE= 4

	1	2	3	4	5	6	7
[-15,-10)	0 90	0 99	0	90	0	60	0 00
[-10, -5)	0 00	0 93	0	91	0	75	0 00
[-5, 0)	0 00	0 90	0	94	1	00	0 00
[0, 5)	0 00	0 87	0	88	0	95	0 00
[5, 10)	0 00	0 86	0	72	0	33	0 00
[10, 15)	0 00	0 21	0	29	0	17	0 00

ON LINE= 5

	1	2	3	4	5	6	7
[-15,-10)	0 88	0 96	0	95	1	00	0 50
[-10, -5)	0 00	0 93	0	91	0	88	0 60
[-5, 0)	0 00	0 89	0	92	0	81	0 88
[0, 5)	0 00	0 86	0	83	0	83	0 89
[5, 10)	0 00	0.77	0	69	0	67	0 56
[10, 15)	0 00	0 13	0	20	0	20	0 12

ON LINE= 6

	1	2	3	4	5	6	7
[-15,-10)	0 84	0 94	0	93	1	00	1 00
[-10, -5)	0 00	0 97	0	93	0	97	1 00
[-5, 0)	0 00	0 89	0	92	0	80	0 89
[0, 5)	0 00	0 78	0	83	0	86	0 69
[5, 10)	0 00	0 76	0	69	0	57	0 76
[10, 15)	0 00	0 21	0	20	0	17	0 10

ON LINE= 7

	1	2	3	4	5	6	7
[-15,-10)	0 87	0 93	0	94	1	00	0 67
[-10, -5)	0 00	0.89	0	93	1	00	0 92
[-5, 0)	0 00	0 84	0	83	0	86	0 91
[0, 5)	0 00	0 90	0	82	0	75	0 69
[5, 10)	0.00	0.64	0	66	0	51	0 58
[10, 15)	0.00	0 19	0	15	0	18	0 14

Write HLS Program

Figure 9 shows the first of three sheets of the HLS program flow diagrams. The total program provided results 1000 times faster than the LLS. Basically, the program read the true state of the environment and reacted statistically the same as the LLS by referring to lookup tables containing statistics that the LLS would display for a particular environmental configuration of reply data. The actual response was a random selection of responses from the LLS generated distribution.

Integrate HLS Program into TCAS II System Simulation and Obtain Test Results

The HLS program was integrated with the rest of the TCAS II system simulation and allowed running of programs essentially in real time. That is, 15 minutes of simulation time corresponded roughly to 15 minutes flight of targets in the Los Angeles area.

Figure 10 shows 77 targets simulated. The coverage display is a circle of radius equal to 10 nmi. Our aircraft is at the center with heading toward the top of the circle. The velocity vector (speed and direction) for each aircraft is shown by the stick on each triangle or square. The length of each stick is equal to the difference between present position and the position 25 seconds in the future. There are 33 targets replying with ATCRBS replies and 44 replying in Mode S in this simulation. A collision is predicted in this case and the advisory of own aircraft is to descend.

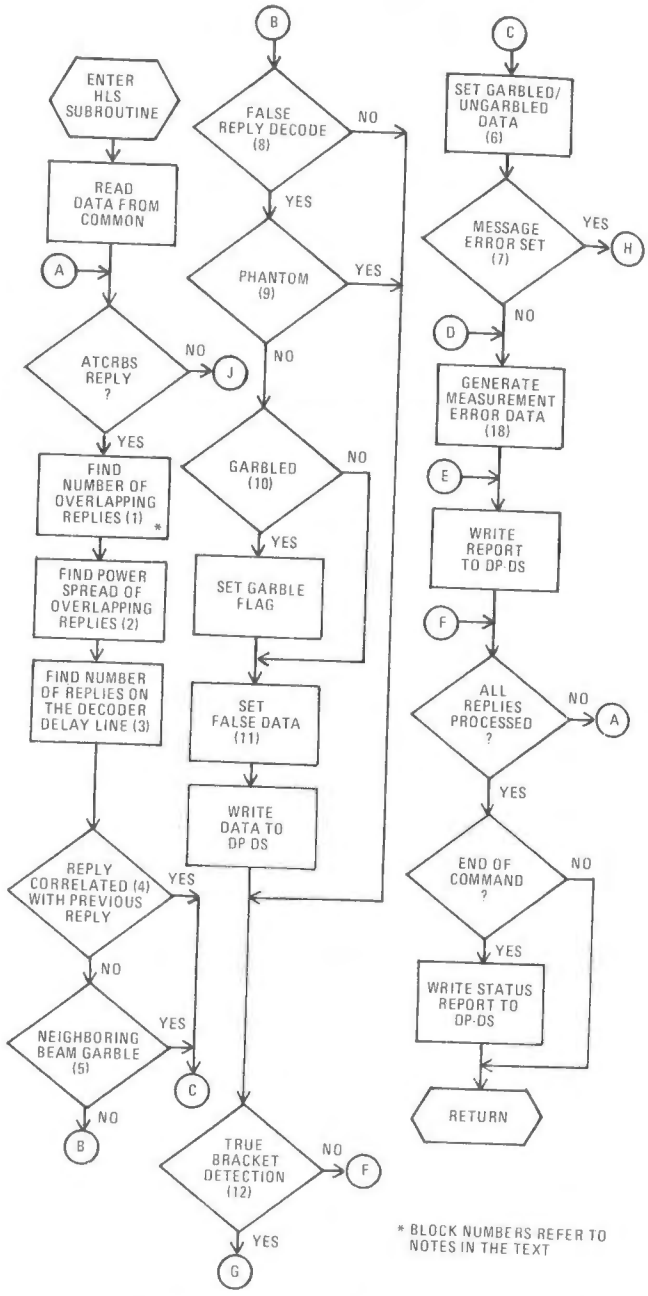


Figure 9. Functional Flow of HLS (Sheet 1 of 3)

Finalize Hardware Design

Finalization of hardware design consisted of reviewing the high- and low-level simulation results as well as the preliminary hardware design before progressing through the following hardware design steps:

1. Incorporate changes due to LLS results.
2. Incorporate changes due to HLS results.
3. Finalize design schematics for hardware.
4. Flow chart and program interrogator microprocessor software.

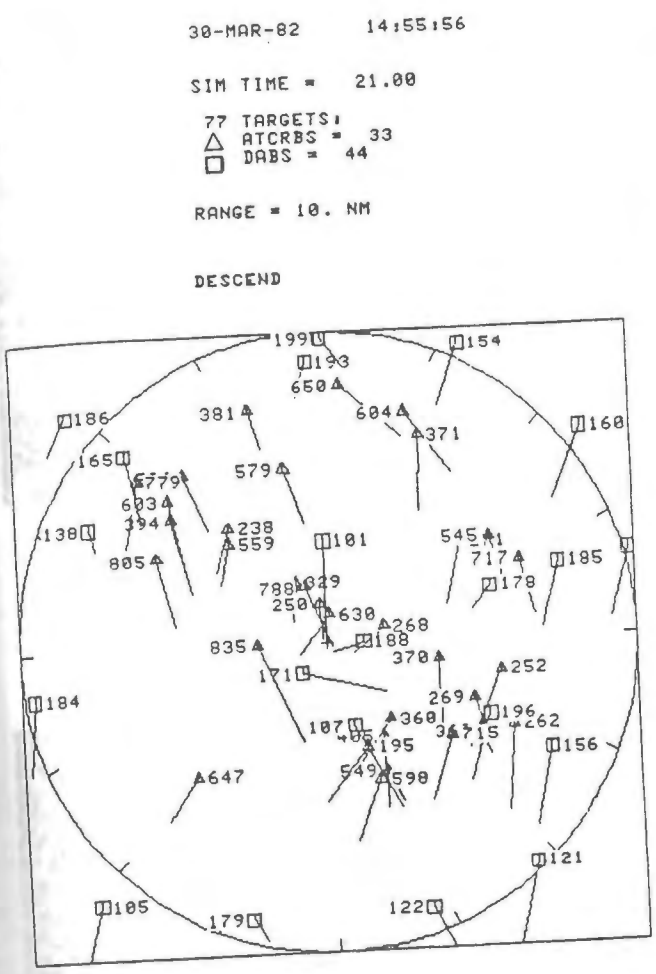


Figure 10. TCAS II System Simulation Display

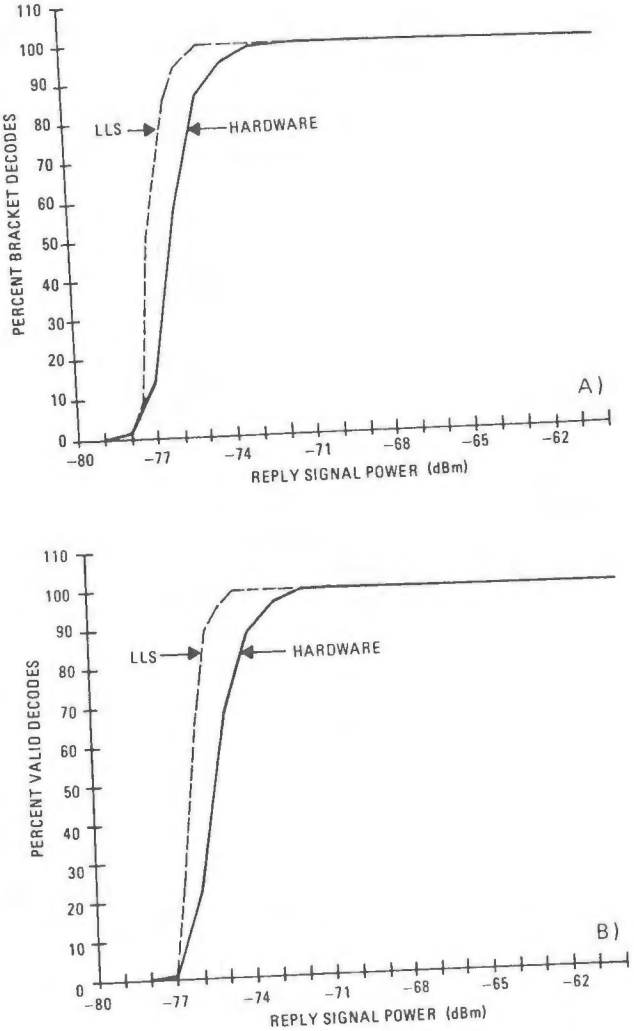


Figure 11. Comparison of Hardware and LLS Sensitivity Curves

Fabricate and Test Signal Processor

Fabrication and test of the signal processor resulted in data that allowed comparison of the signal processor results and the LLS results. A sample of those results are presented in Figures 11 and 12. Figure 11 essentially shows agreement between hardware and LLS decoder sensitivity curves for bracket decode and valid data decodes. The slope difference on the curves is attributed to differences between the noise characteristics for the LLS and the hardware. Figure 12 shows the valid decode performance for two closely spaced replying transponders. The two replies have bracket pulses overlapped by about half of a pulse width. When the amplitude of the two replies is equal, the system will correctly decode 75 percent of the replies as indicated by both the LLS and the hardware test results. These results exceed the initial design goal of 50 percent for this case. When the subject reply is 10 dB greater in amplitude than the interfering reply, valid decodes for the subject reply approach 100 percent. For the interference 10 dB greater than the subject reply, valid decoding of the subject reply drops to about 25 percent as expected.

Flight Testing of Signal Processor in TCAS II System

Figure 13 shows the video display of the TCAS II in the FAA Boeing 707 test aircraft during flight tests. Figure 13A shows a pilots view of the display. A magnified view of the display is shown in Figure 13B with a display radius of 5 nmi. In this case, the pilot is being alerted to an aircraft approaching from the port side and 300 ft above own aircraft. This is a near collision case where own aircraft is being advised to turn right.

Conclusion

The application of this design/simulation methodology has resulted in a viable Traffic Alert and Collision Avoidance System which has demonstrated, in flight, the performance that was predicted in earlier simulations. These simulations provided a valuable design function and directly influenced the hardware/software implementation.

SOFTWARE CONTROLLED DISPLAY/PUSHBUTTON

Dean A. Nicholson
MICRO SWITCH
Division of Honeywell

Robert J. Spiger
Boeing Aerospace Company

ABSTRACT

Virtually every aircraft being designed today includes some form of software-controllable man/machine interface. Many of the reasons are obvious, such as, reduced space and weight requirements, reduced training and errors, and easier reconfiguration. The purpose of this paper is to discuss one of the newest entries into the field of software-controllable man/machine interfaces. This device is commonly called the programmable pushbutton switch. This type of pushbutton includes a display which is also software-controllable so that it is truly a means of two-way communication between the operator and the system. Both the function of the switch and the message on the display can be changed at any time by the software of the host system. The display is a dot matrix of 16 x 35 dots (560 total). It is fully addressable so that any display can be generated which can be drawn on this size matrix. The display technology is LEDs and the switch tecnology is Hall effect. This paper discusses the particular advantages of this type of control and how it may be best used in the avionics environment. This programmable pushbutton has application in the cockpit, at other crewstations, in electronic surveillance, in fire control, and in automatic test equipment. The programmable pushbutton can be combined with other types of man/machine interface to provide a truly "user-friendly" control panel that is also "designer-friendly".

INTRODUCTION

As avionics systems become more complex, the operator/system communicatons become more critical. Modern equipment can handle massive amounts of information at lightening speeds. The interface with the operator increasingly becomes the area to focus attention on if further communications improvements are to be made.(1) Operating modern equipment requires more skill and results in expensive training. The cost of an operator error also increases as the cost of equipment increases.

The first portion of this paper discusses the design of two devices intended to improve the operator/system interface: 1) the programmable display pushbutton provides user-friendly communication for the operator; 2) the logic and refresh control unit provides a convenient, practical method for interfacing the programmable display pushbutton into the system. The two devices together, as shown in Figure 1, provide an operator/system interface which helps reduce operator training, reduces operator error and makes the total system more efficient.

The second portion of the paper discusses system considerations using the programmable display/pushbutton and a particular application of these devices to a multifunction keyboard.

PROGRAMMABLE DISPLAY PUSHBUTTON

The programmable display pushbutton is a multifunction, interactive device for man/machine interface. This device is a combination pushbutton switch and dot matrix display for two-way communication between operator and system.

The switching portion of the device is a solid state Hall effect switch. It provides an open emitter, current sourcing output. The switch operates on 5VDC. The switch is operated by pushing on the pushbar at the bottom of the display or on the display surface itself. The operating force is one to three pounds on the pushbar. It is higher on the display, since the display pivots at the top. Tactile operation is provided by a spring beam mechanism. The tactile operation is designed for applications requiring emphasis on data entry accuracy.(2)

The display portion of the device is a fully addressable dot matrix of discrete LEDs. The matrix is made up of 16 rows by 35 columns. The 560 pixels can be lit in any combination to form alphanumerics and/or graphics. The display operates on 5VDC. The matrix is multiplexed, one row on at a time. This allows the 560 pixels to be individually controlled with just five logic signal lines. To accomplish this, decoding and drive electronics are provided in the switch. Communication between the programmable display pushbutton and a host is via an eight pin connector. TTL level logic signals control the display. The electronics are constructed as multilayer hybrids. A band-pass filter is used to enhance the display. The filter also includes glare-resistant, scratch-resistant, fingerprint-resistant coatings. A mask is used above the LEDs and below the filter to enhance contrast ratio and keep the pixels clear and sharp.

The display and pushbutton are combined in a sturdy plastic housing designed to withstand shock, vibration, and temperature extremes. The package is front-of-panel sealed against liquid spills. Sealing is accomplished with a flat elastomer seal at the switch-panel juncture and a flexible seal around the movable display.

The programmable display pushbuttons are individually mounted in rectangular panel cutouts. Panels can be configured with any desired number of switches. The programmable display pushbutton can be used with or without the logic and refresh control unit.

LOGIC AND REFRESH CONTROL UNIT

A logic and refresh control unit was developed to facilitate interfacing the programmable display pushbutton with the host, and also to provide several desired functions. One control unit can interface up to four programmable display pushbuttons. Interface to the switches is via the same eight pin connector discussed previously. Interface to the host is via a fifteen pin connector. Figure 2 illustrates how these devices fit into a total system. Communication to and from the host is with asynchronous serial signals compatible with RS422 or RS232 interfaces. Communication is in 8 bit bytes with one start bit and one stop bit. Bit seven is odd parity on transmissions from the control unit. There is no parity bit on transmission from the host.

The logic and refresh control unit is a single printed circuit board including a microprocessor, an EPROM, a refresh controller, four RAMs, and various other devices. The control unit has the capability of performing many functions.

ASCII Conversion - The control unit can convert commands for an ASCII subset of 64 characters into the signals required by the programmable display pushbutton to generate those characters. The control unit can generate these characters in two sizes for two rows of up to six characters or one row of up to three characters.

Bit/Pattern Mapping - The control unit decodes graphics commands to draw parallel, vertical and horizontal lines, or light individual pixels.

Brightness Control - The control unit can vary brightness of the displays as commanded by the host. The display brightness is controlled by varying the duty cycle of the LEDs. The control unit can select 36 different duty cycles.

Display Blinking - When requested by the host, the control unit can blink a display at 1.5 Hz. The blinking is accomplished by switching the duty cycle signal for that programmable display pushbutton. The duty cycle signal blanks the display when the signal level is high.

Self-Test - Upon command from the host, the control unit will perform a self-test of many of its own components and the outputs of the switches. It sends to the host a self-test pass message, or a self-test fail message containing a failure description code.

Display Refreshing - The control unit refreshes each display at over 500 Hz. This refresh rate is designed to overcome the trait of refreshed displays to break up visually when viewed under high vibration. When a display message is received by the control unit, the microprocessor verifies the message and converts it to the proper signals to generate that display. The signals are passed to the refresh controller. The refresh controller is a custom designed bipolar IC. The refresh controller then sends the signals to the RAM for that particular display. When the message is complete, the refresh controller is switched to the refresh mode and the displays are turned on.

When a programmable display pushbutton is actuated, the control unit will transmit a "Switch Depressed" message to the host. This message includes a code identifying which switch was depressed.

An Interrupt Request Output signal is available from the control unit no more than five milliseconds after a switch is actuated. This can be used as a cue to the host to abort messages being transmitted by the host. The control unit will not transmit a "Switch Depressed" message when it is receiving a message from the host.

Other Design Features - The control unit can communicate at four baud rates. The desired rate is chosen by moving a jumper on the PC board. The four rates are 2.4K, 4.8K, 9.6K and 19.2K baud.

All messages from the host are verified by a sumcheck. If the sumcheck is not correct, or the message is invalid for another reason, the control unit will send a "Retry" message to the host. Any display currently in RAM will not be destroyed until a complete valid message has been received.

The displays can be enabled (refresh mode) by either an End of Message byte or an End of Transmission byte. The choice is made by moving a jumper on the PC board. The End of Transmission enable prevents the control unit from displaying half a display in the event the host is interrupted between messages.

The control unit's serial output can be inhibited by a Select In signal from the host. This allows the host to monitor several different control units selectively. This feature is also chosen by a jumper on the PC board.

The four programmable display pushbuttons are individually fused and buffered on the control unit so that a catastrophic failure of one will not affect the others. The microprocessor, EPROM, and refresh controller are common to all the programmable display pushbuttons.

In the event of a loss of power, the RAMs on the control unit will lose the information for the current display. When the power is restored, the control unit will send an "Acknowledge" message to the host. This can alert the host to resend the same displays or send new ones.

MULTIFUNCTION KEYBOARD

The concept of a software reconfigurable display on a keyboard has been under study for some time.(4) The programmable display pushbutton is one of the first components to make such a keyboard possible using discrete switches. Using these devices, a multifunction keyboard (MFK) has been developed which satisfies several important goals for avionics and CCCI system applications. These include:

Host Independence - Avionics and information systems often employ a central host computer for data processing and communication with sensors and controls. In numerous cases, technology limitations at the time of design and/or system growth have produced a current situation in which

the host has little additional memory available and a software configuration which is difficult to modify. Figure 3 shows a block diagram of a host and MFK configuration. In this MFK design, the keyboard switches are combined with an auxiliary thin-film electroluminescent (TFEL) scratchpad display which displays operator entries, menus, and responses from the host. Minimal host interaction and processing are part of the MFK design. MFK software and firmware store the switch and display legends, handle the logic of successive legends and displays upon switch activation, and process the I/O with the host. Figure 4 shows an example of the hierarchial logic system often used in an MFK for function access. To minimize host loading, only the final switch activation would cause a transmission to the host. The legend changes and command logic are part of the MFK software and firmware.

Reconfigurability - The long life of current airframes generally implies a considerable number of updates and/or revisions in the aircraft systems.

This is particularly true for weapons, sensor and communications systems in military aircraft. The MFK design provides a distinct separation between the background operating system and the data base which defines the legends displayed, and the commands sent, as a result of operator switch action. Consequently, the switch functions and legends can be easily changed to reflect revisions or replacements within the original system.

Fault Tolerance - An obvious potential problem in an MFK design is the impact on the system of a switch failure. The effect can be a loss of access to a whole branch of the system. In a dedicated switch array, the impact is more likely to be limited to a single function. The MFK design alleviates this problem by monitoring the keyboard for indications of a continuous switch closure. If observed, that switch is judged faulty and the data base is automatically reconfigured to bypass the row of switches associated with that logic and refresh control unit. A second page of legends is formed, if necessary, to provide a display of all the original legends. Similarly, a manual operator test permits the checkout of proper activation and display on each switch. During this test, a switch which does not close is detected visually and by MFK software and a reconfiguration to bypass the switch is implemented. These design features provide a much higher degree of reliability to the MFK. It should be noted that the discrete programmable display pushbuttons permit this type of a bypass, whereas a single output device (e.g., a resistive touch screen) would not.

Response Time - A noticeable delay in the time required to update the MFK legends requires the operator to consciously wait between steps. This delay can be annoying in many cases and detrimental in systems where a rapid response is important. As a result, a keyboard update time of 200ms or less was adopted as a design goal for the MFK. This response time has proven to be satisfactorily rapid for tests conducted with the MFK to date.

In Figure 5 a picture of the MFK in operation is shown. The keyboard array comprises five rows of four switches. Each row is controlled by an individual logic and refresh control unit. The TFEL display shows a menu of options selectable from the keyboard. Note the two font sizes for keyboard legends. The level selector switch permits the operator to move back to the previous page of legends or to return to the top level of the MFK data base. The photodiode in the center of the panel monitors ambient light and adjusts luminance levels on the keyboard LEDs to programmed values. Variations within a predetermined range of acceptability are set with the intensity control at the right of the panel.

CONCLUSION

The programmable display pushbutton and the logic and refresh control unit provide a significant advance in operator/system communication. These devices are state-of-the-art technologies combined and packaged in a way to provide major new capabilities. They give the system designer more freedom to create. By combining these devices with microprocessor controllers and software to form keyboards and displays such as the MFK, a reduction in hardware and space and an improvement in the "user friendliness" of avionics and information systems can be achieved.

REFERENCES

(1) Johnson, Roger "Centralized Control Concepts, An Evolution of Operator Interfacing In Process Control" Proceedings of Advanced Control Conference, 1980.

(2) Taylor, R.M.; Berman, J.V.F., "Aircraft Keyboard Ergonomics: A Review", RAF Institute of Aviation Medicine, Farnborough, UK, 1983.

(3) Hatfield, Jack J.; Robertson, James B.; Batson, Vernon M. "Advanced Crew Station Concepts, Displays, and Input/Output Technology For Civil Aircraft of the Future". CH1518-0/79/000.0187, IEE, 1979.

(4) Graham, D.K., "Logic & Design Procedure Development for Multifunction Mode Switching Controls", JANAIR Report 730501, 1973.

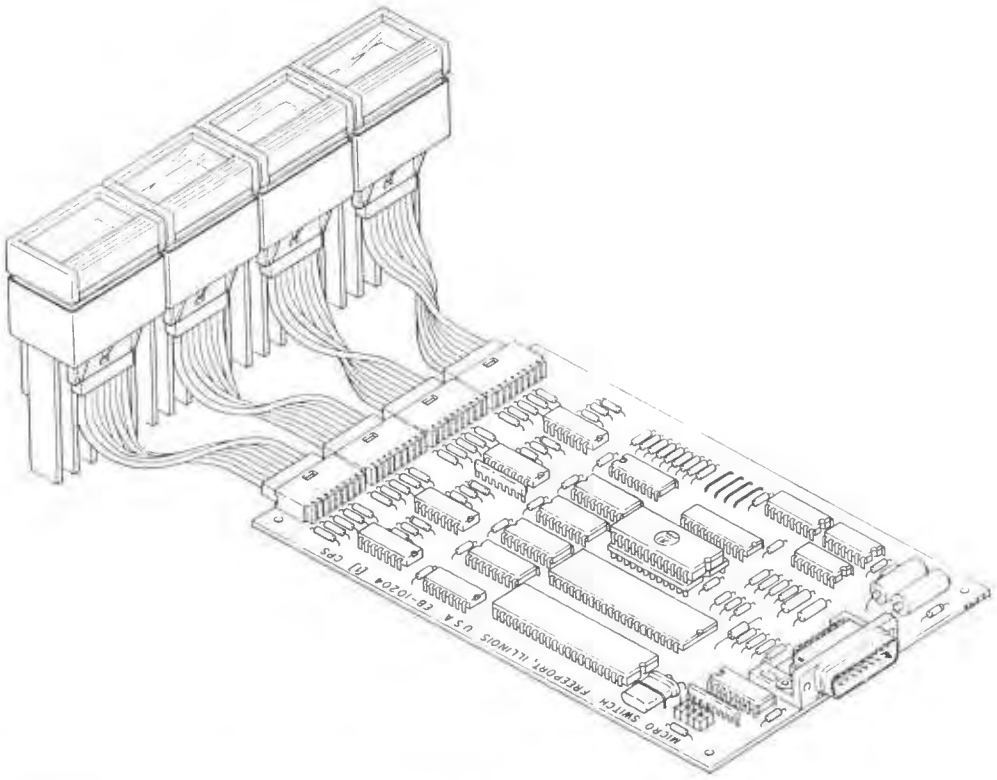


FIGURE 1 - FOUR SWITCHES AND ONE CONTROL UNIT

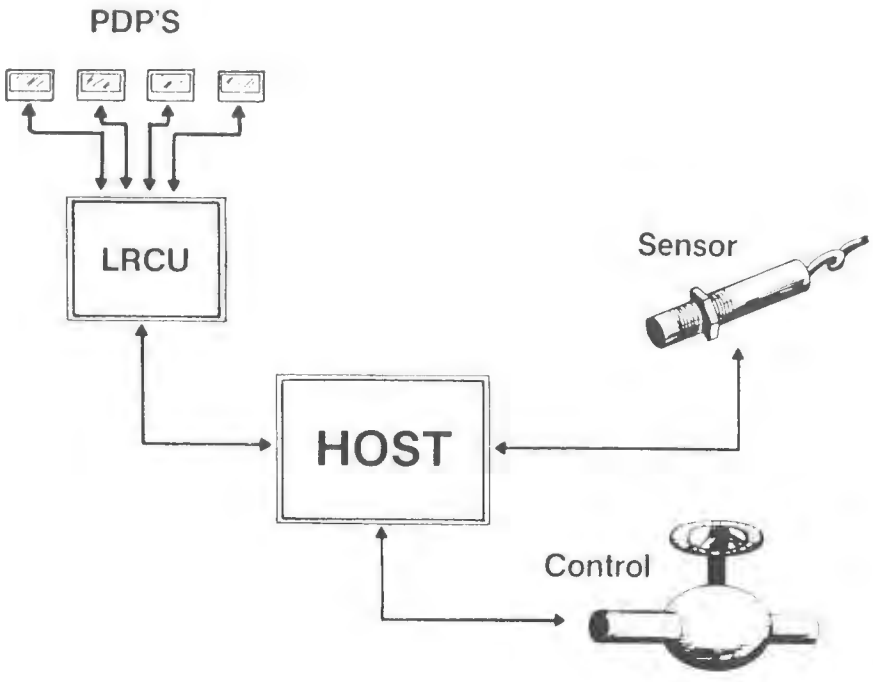


FIGURE 2 - SWITCHES AND CONTROL UNIT IN A HOST SYSTEM

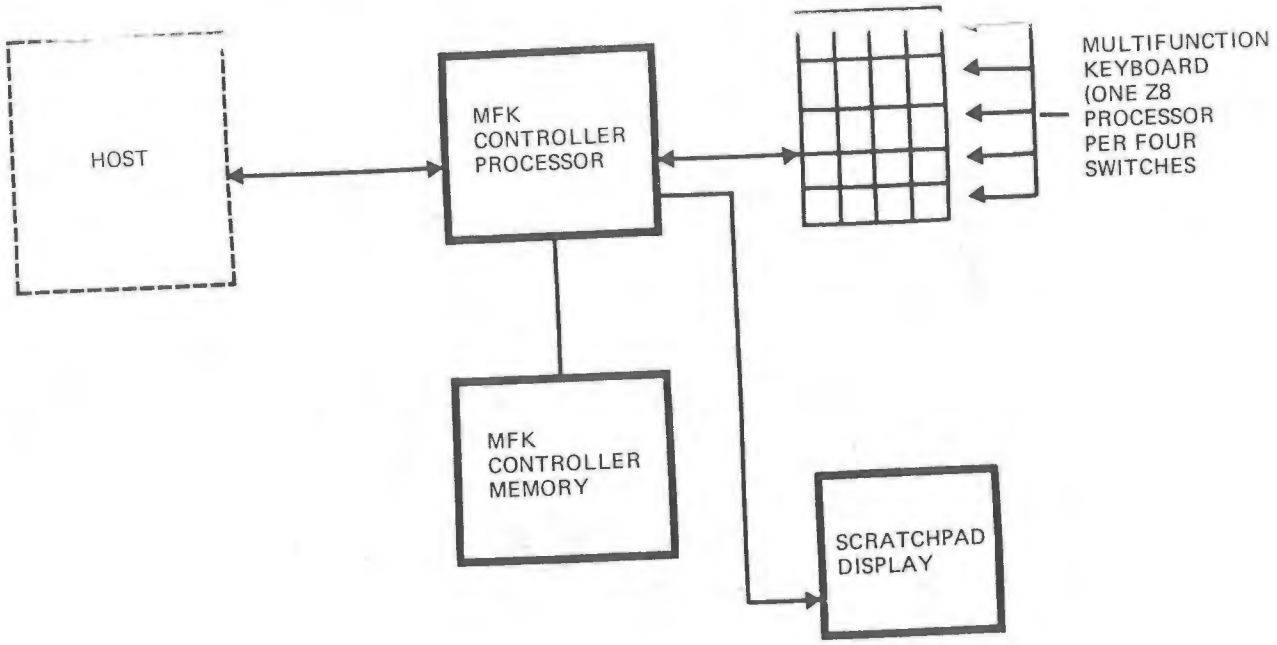


FIGURE 3 - CONFIGURATION OF AN MFK USED WITH A HOST COMPUTER

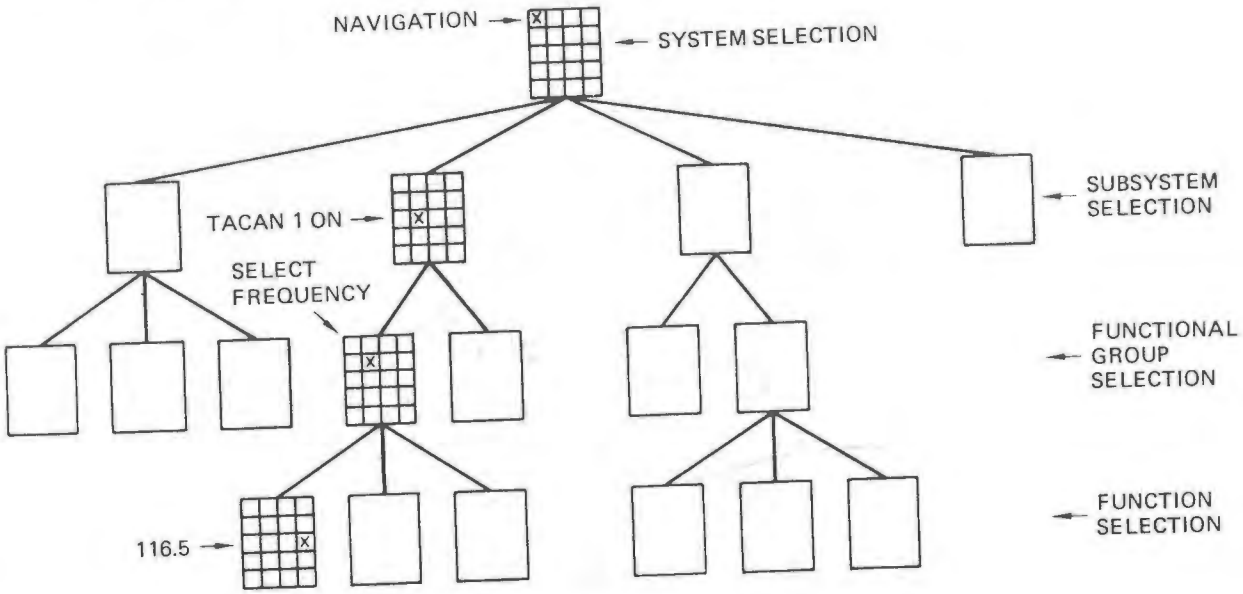


FIGURE 4 - EXAMPLE OF A LOGIC TREE HIERARCHY USED WITH THE MFK DATA BASE

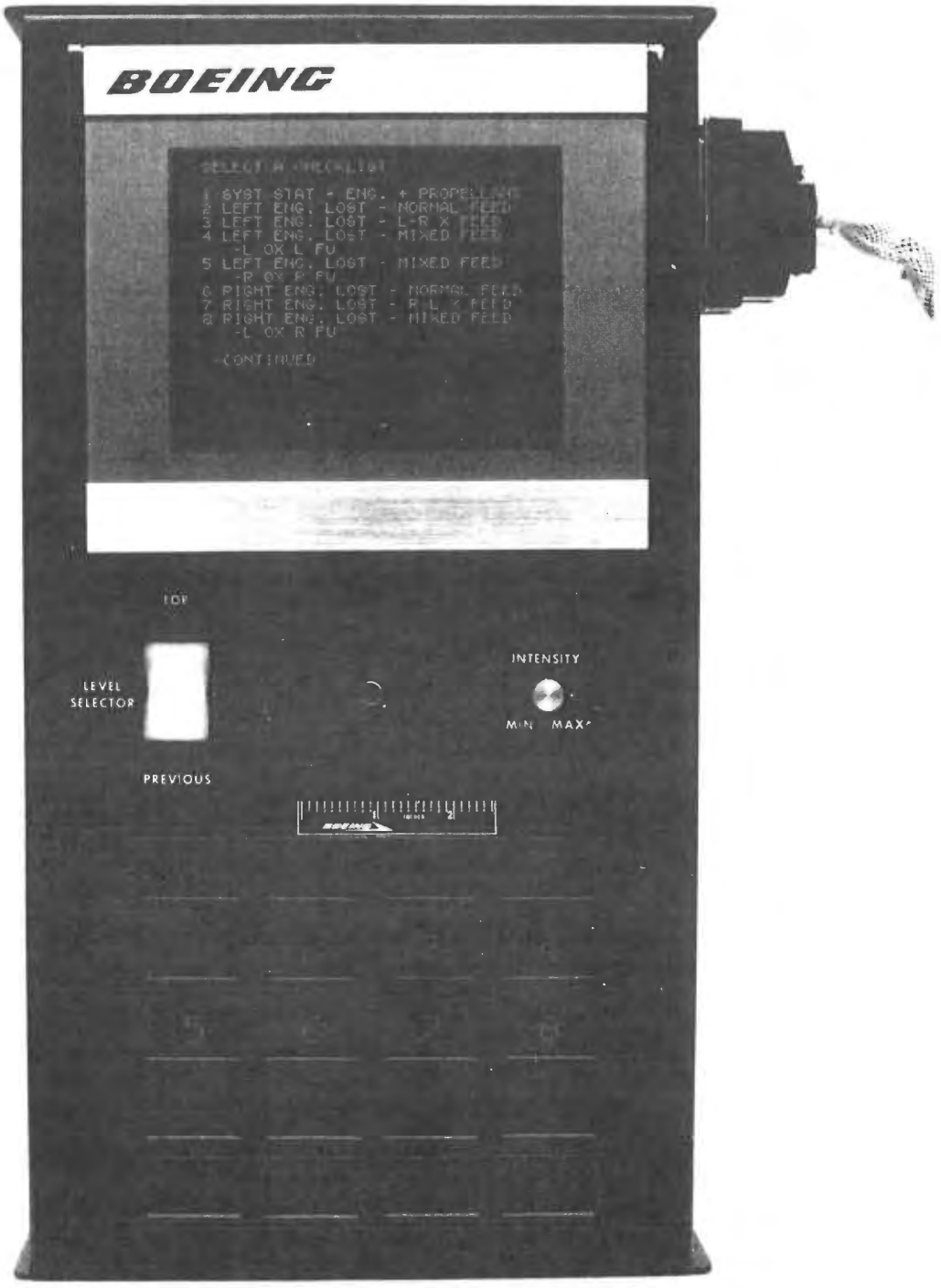


FIGURE 5 – MFK KEYBOARD

SHOWING TFEL PANEL & SWITCH ARRAY

RECENT ADVANCES IN ELECTROLUMINESCENT DISPLAYS APPLICABLE TO FUTURE CREW-STATION INTERFACES

M. ROBERT MILLER
ELLIOTT SCHLAM
US ARMY ERADCOM
FORT MONMOUTH NJ

JAMES B. ROBERTSON
JACK J. HATFIELD
NASA-LANGLEY RESEARCH CENTER
HAMPTON VA

ABSTRACT

Matrix addressed thin-film electroluminescent (TFEL) display technology shows great promise for ultimately replacing the cathode ray tube (CRT) as the primary, integrated, pictorial display device in flight decks of future aircraft and spacecraft. Among the advantages of the TFEL display are: minimal depth behind the panel, low power consumption, high environmental tolerance, no geometric distortion, uniform edge-to-edge resolution, direct digital signal compatibility, high reliability, stable phosphor brightness during long term operation and graceful degradation. In addition, in the light of recent developments, TFEL display technology has the potential for full color. Recent advances in the development of TFEL displays include: the integration of a 240 x 320 element panel with a high performance raster graphics generator for the display of integrated, pictorial flight displays, the incorporation of a similar panel in an advanced simulated navigation control/display unit, the television video operation of a 512 x 640 element panel, the development of black (light absorbing) layers for contrast enhancement, the development of red, green and blue EL phosphors and establishment of plant production lines for large area TFEL panels. This paper will outline programs in which this technology is being incorporated in cockpit display applications.

INTRODUCTION

Flat panel displays have the potential for vastly improving many commercial and military systems. Avionics is a particularly exacting area where flat panel displays could make a big impact in freeing 10-12 inches or more of instrument panel depth for other needs. There are a number of candidate flat panel display technologies, the three most likely candidates being plasma, liquid crystal and thin film electroluminescence, although some might argue for other candidates such as vacuum fluorescent. Because of the reputation that plasma has for substantial power consumption and weight, it is generally not considered to be a good candidate for avionics application. On the surface, liquid crystal would be considered to be an excellent candidate because of its low power consumption, however with further analysis, this simplicity breaks down. There is still very great difficulty in multiplexing the 200-500 rows needed for a graphics quality display, the viewing angle is very limited, and the temperature sensitivity requires heaters which tend to circumvent the apparent power advantage. Thin film electroluminescent displays on the other hand are readily multiplexed, consume very moderate amounts of power, are temperature insensitive, are legible in the high ambient light encountered in aircraft, and can show live video in addition to graphics. We have chosen this technology to pursue a variety of applications and are evaluating its specific application to avionics. The TFEL video graphic exerciser described in this paper was designed to meet a wide variety of application's tests in that it has a display head which is less than 1-inch thick which is tethered by a single 8-foot long flat cable to its small controller box, and is therefore easily physically configurable for a variety of installations. The software and pictures presented on the display is just a small subset of what can be done with such technology.

TFEL TECHNOLOGY STATUS AND DEVELOPMENT

The major thrust of display development effort at the Army ERADCOM ET&D Laboratory has been applied to the thin film electroluminescent (TFEL) technology because of advantages with regard to weight, power, legibility and ruggedness inherent in this approach. TFEL panels are made up of a number of thin film layers deposited on a glass substrate. The panel layers are all thin (on the order of 1,000-3,000 angstroms) which produces a display thickness determined by the glass substrate. Light is generated when a potential of approximately 200 volts is applied between the rows and columns in a multiplexed manner. Grey shades are obtained by modulating the voltage applied to the pixels. As a result of an Army-NASA tactical video display program with Hycom and Supertex, live TV/video with full resolution of 480x640 pixels and 16 grey shades has been demonstrated. (Ref. 4) Monolithic circuits that include all of the logic necessary to shift data in serially and out in parallel and drivers for the high voltage are now available. Panels can be built with a black layer between the rear insulator and the rear electrode where the black layer typically reflects less than 1% of the incident light. This allows the panel to be used in direct sunlight with a pixel illumination of less than 25 footlamberts, and still provides adequate contrast to be legible.

FABRICATION

TFEL display panels are fabricated by vapor deposition of conductive, insulating and phosphor layers on a glass substrate. Various deposition techniques, including thermal evaporation, electron beam evaporation, sputtering and atomic

layer epitaxy have all been used successfully. Production deposition systems in use are either batch type or inline. In a batch type, multiple substrates are loaded into the chamber at a time. An inline system processes substrates fed in at one end into finished panels coming out the other end. Precise control of the thickness and uniformity of the various thin film layers is of utmost importance. This is achieved by monitoring all of the important process parameters and using automated control to eliminate variability.

The TFEL display operating with about 200 Volts across about 10,000 Angstroms of film thickness, must withstand the highest operational electrical field of any device in use today. For this reason, the integrity of the insulating layers in the thin film structure is a most critical factor in the design. Modern thin film processes are capable of producing insulators with the required breakdown resistance over large areas, however none of these layers are perfect and minute weak spots do exist. If the entire film structure is properly designed, these weak spots will burn out when the device is first excited leaving microscopic "holes" in some pixels which are not easily visible and do no further damage to display operation. Therefore perfect films are not a requirement of perfect displays as long as the size and incidence of the imperfections can be controlled.

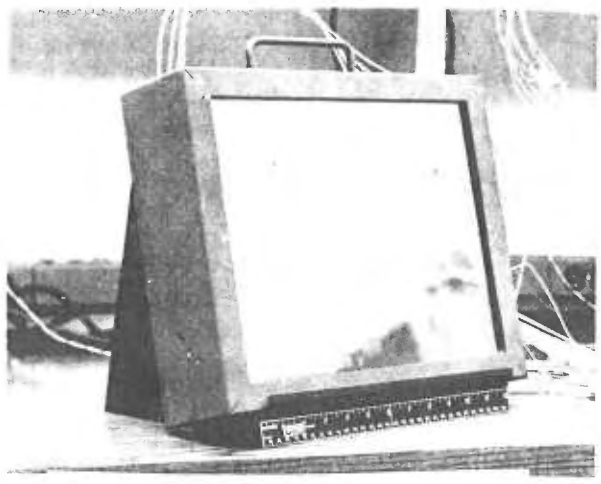
DRIVING ELECTRONICS

Monolithic circuits that include high voltage outputs as well as the logic necessary to shift data in serially and out in parallel are now available to drive the rows and columns of TFEL panels. Drivers for alphanumeric and graphic applications, operating in an "on - off" mode are available with 32 and 64 outputs. Video drivers capable of providing 16 levels of gray shades are available with 16 outputs.

Power requirements of TFEL displays are relatively modest but are still capable of being improved considerably. A 16 square inch graphics panel using conventional drive circuitry consumes in the range of 11 to 14 Watts. A similar 15 square inch panel using a partial energy recovery drive scheme has been demonstrated to operate with a power consumption of 4 to 6 Watts. Only about 5 percent of the energy applied to the panel is consumed in generating light, the remainder being stored on the panel capacitance. This capacitive energy is available to be recovered through one of several efficient drive schemes that have been demonstrated. Calculations indicate that the above panels, driven with more complete energy recovery, would operate in the range of 1 to 2 Watts. This makes the use of these devices in battery operated equipment a practical consideration.

PANEL SIZE AND RESOLUTION

Panel sizes under development range from 3 x 5 inches at 64 lines per inch up to 1 meter x 1 meter with 50.8 lines per inch with various other sizes in between. For many applications, a panel size of 10 x 12.6 in. with



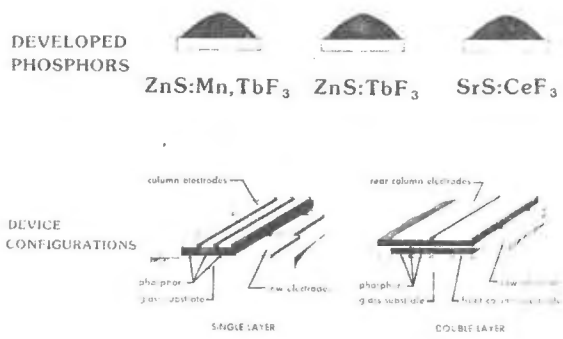
10" x 12.6" TFEL Display Panel

50.8 lines per inch is ideal. This 512 x 640 line panel is capable of displaying a 51 line by 80 character alphanumeric display as well as graphics and video.

limitations on panel size are determined by the number of rows that can be driven in a multiplexed manner and by the electrode time delays defined by the effective resistance times capacitance (RC) of the rows. The ability to multiplex many rows without degrading contrast depends on the existence of a strong threshold characteristic separating the on from the off states in the display medium. The TFEL devices have a very steep threshold and multiplexing of more than 500 lines has been demonstrated. Recent advances in the fabrication of transparent conductive layers provides an improvement in the RC time delay by a factor of three. The upper limit of TFEL panel size is still under investigation.

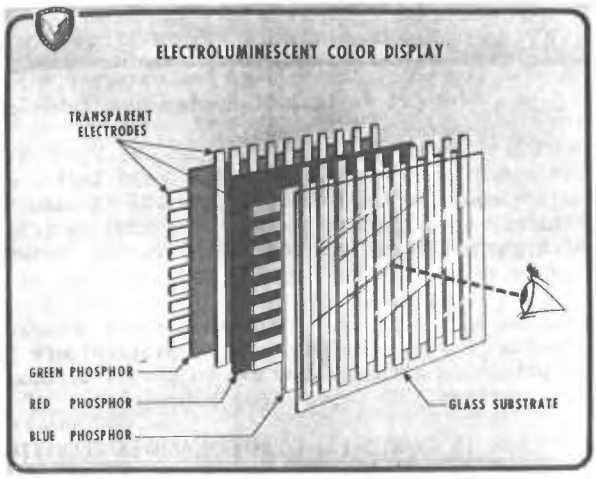
COLOR

Until recently, the majority of work in TFEL devices has centered around a phosphor film made of zinc sulfide doped with manganese which produces an amber colored display. This material is stable, easy to work with and produces good luminance and efficiency. Zinc sulfide doped with



FULL COLOR ELECTROLUMINESCENT DISPLAYS

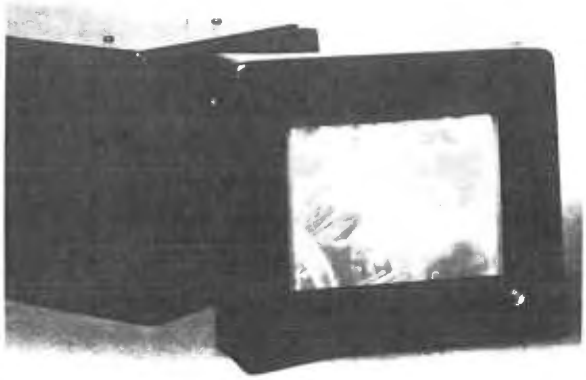
terbium fluoride produces a green display with about half of the luminance and efficiency as those of the manganese doped phosphor. As a result of an Army, Navy, NASA supported contract with Tektronix, red and blue phosphors have been developed. (Ref. 1) Although the luminance and efficiency of these new phosphors are low in comparison to the amber and green, they are at a level that could produce workable displays and improvements are expected from further research in this area. A new effort in this area is intended to examine the best approaches to combining these phosphor materials into an operational multicolor display.



EL APPLICATIONS IN AVIONICS

EL displays have been put to use in several simulator or test vehicle programs. These programs are aimed at helping move the EL display from the laboratory into the cockpit. The displays are being tested in avionics systems in real and simulated cockpit environments.

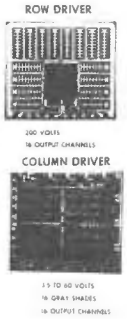
Two of these programs make use of the Video/Graphics Exerciser (VGE) developed by Hycom Inc. under a contract funded by the US Army, Navy, Air Force and NASA. The VGE has a 240 x 320 element, 6 inch diagonal EL display with a touch-panel overlay.



VIDEO-GRAPHIC TFEL DEMONSTRATOR

FIRST LABORATORY TESTS OF A HIGH-RESOLUTION LARGE-SCREEN TFEL DISPLAY USING MONOLITHIC DRIVERS

- 100 PIXELS/INCH (512 X 640 PIXELS)
- 8-INCH DIAGONAL SIZE
- RS-170 INTERFACE FOR VIDEO/GRAPHICS
- HIGH-VOLTAGE LSI DISPLAY DRIVERS

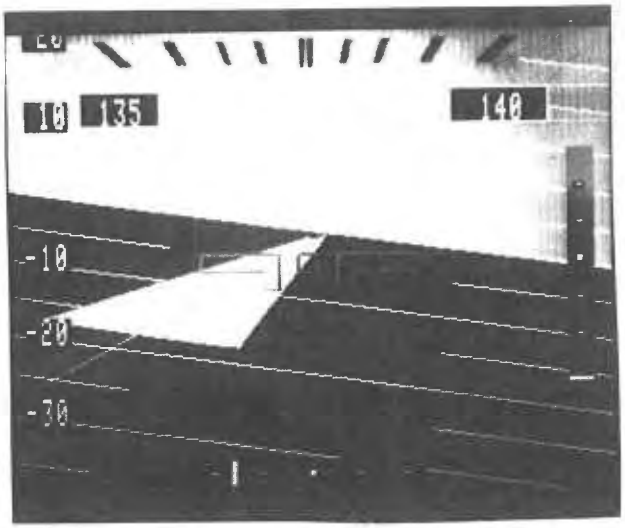


RASTER GRAPHICS IN EL

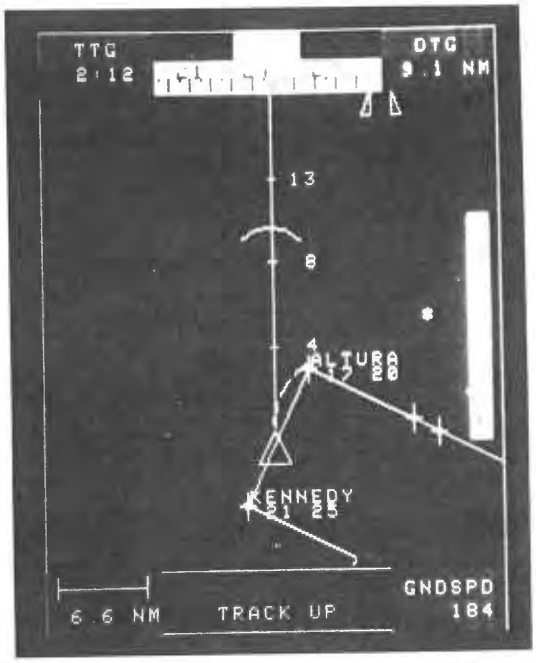
At the Cockpit Systems Branch of NASA, Langley Research Center, a VGE has been integrated with an Adage-3000 high-performance, raster-graphics, programmable display generator (PDG) and a VAX 11/780 host computer for the generation and display of integrated, pictorial primary flight displays (ref. 2).

Existing software for the programmable display generator produced am electronic attitude director indicator (EADI) and an horizontal situation indicator (EHSI). For use with high-resolution color CRT displays, these display formats were provided as 512 x512 picture elements of resolution, RGB, 30 and 40 Hz interlaced and 60 Hz non-interlaced signals. Since the VGE required NTSC monochrome, composite video as input for digitizing the video and displaying it on the 240 x 320 line EL panel, the PDG output signal formats had to be adjusted. In order to display the EADI and EHSI on the VGE, the following had to be accomplished: 1. scaling of the formats to 240 x 320 picture elements (525 scan-line, 60 Hz, non-interlaced video), 2. modification of the PGD computer program to support the appropriate channeling of pixels to required regions of the color look-up tables (LUVO's), 3. adjustment of the PDG video amplitude parameters to match those of the EL display, 4. conversion of the RGB color maps to gray-scale color maps, and 5. use of the green output of each LUVO as a monochrome NTSC signal.

The above EADI and EHSI displays showed no missing lines and almost no discernable effect of missing pixels within lines and grey scale performance was good. In addition, dynamically, there was no smearing, streaking, or flicker effects. These results are quite encouraging. The Cockpit Systems Branch plans to move forward into pilot-in-the-loop simulator evaluations once the final version of the VGE is delivered by Hycom, Inc. At this time, pixel-clock synchronization between the VGE and the PDG will be attempted to reduce pixel-level scintillation effects experienced in parts of the EADI and EHSI displays with only video horizontal and vertical drive level synchronization.



Electronic Attitude/Director Indicator Displayed on 240 x 320 Pixel EL Panel



Electronic Horizontal Situation Indicator Displayed on 240 x 320 Pixel EL Panel.

ADVANCED NAVIGATION SYSTEM

Another application of EL displays in avionics systems is the use of the VGE as part of a navigation controldisplay unit (NCDU) in an advanced navigational system which is being evaluated in a transport simulator at the Cockpit Systems Branch of NASA, Langley Research Center. The other pilot-interface portions of the NCDU is a LED multifunction keyboard (MFK) and a color-coded EHSI (map) display. This advanced navigational system which provides a simple method for the pilot to enter new waypoints and change flight plans, is described in another paper being given in the "Crew Systems: Systems

Advanced Navigation System

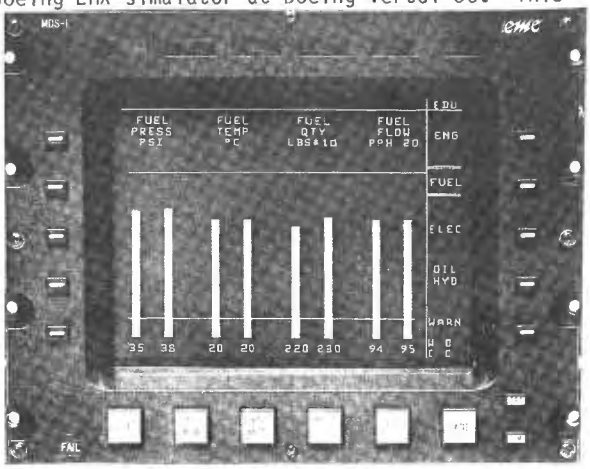
Development and Integration" session of this conference. The EL display of the VGE is used primarily as a graphic preselect/control device in which the pilot can use the touch overlay feature to preview navigational routes such as a map of a Standard Terminal Arrival Route (STAR) and to enter waypoints for defining a new route prior to entering it into the color map display (figure 3). The pilot can also use the VGE to scroll or zoom the color map.

QUICKLOOK II AN/ALQ-133 COUNTERMEASURES RECEIVING SET

A product improvement program for the Quicklook II is incorporating a 240 x 320 line TFEL display into the Cockpit Display and Control System. This unit will be used to perform preflight, postflight and diagnostic operations for the AN/ALQ-133 in addition to the inflight mode of operation in which the system will display information stored in various mission files and data files.

Multipurpose Display System

A multipurpose display system built by Canadian Marconi Company using a 240 x 320 pixel EL display is currently being evaluated in a Boeing LHX simulator at Boeing Vertol Co. This



Canadian Marconi Multipurpose Display System.

display system, the CMA-823, has graphics capability, interfaces with 1553B or ARINC 429 data bus, and has ten software-controlled keys and five mode keys on the display bezel.

CONCLUSION

Active pursuit of TFEL technology from both a device development and a systems application point of view promises near term availability of flat panel displays offering significant advantages in the cockpit environment.

References

1. Barrow, W., Coovert, R., King, C. "Multicolor Electroluminescent Thin Films", ERADCOM Research and Development Technical Report DELET-TR-82-0384, Nov. 1982.

2. Montoya, R. Jorge: et al.: "The Application of a Color, Raster Scan, Programmable Display Generator in the Generation of Multiple Cockpit Display Formats", AIAA/IEEE 5th Digital Avionics Systems Conference, December 5, 1984, Baltimore, MD.

3. Jones, D., Parrish, R., Person, L., and Old, J.: "An Advanced Media Interface for Control of Modern Transport Navigational Systems", AIAA/IEEE 6th Digital Avionics Conference, Dec. 5, 1984, Baltimore, MD.

4. Gielow, T., Lanzinger, D., "Tactical Video Display", ERADCOM Research and Development Technical Report DELET-TR-79-0251-3, Jan 1984.

84-2659

α-Si ADDRESSED LIQUID CRYSTAL AIRCRAFT DISPLAY

D.E. Castleberry, J.E. Bigelow, H.G. Parks, W.W. Piper and G.E. Possin

General Electric Company
Corporate Research and Development
Schenectady, New York

Abstract

Liquid crystal displays are becoming accepted for aircraft use because of their obvious advantages of reliability, long-life, low volume, light weight and low power. These applications have been restricted to simple numeric and low information content displays. This restriction has recently been removed with the development of amorphous Si thin film transistors that can be deposited on glass substrates. A liquid crystal display with a thin film transistor at each pixel can have performance equivalent or superior to a CRT.

This paper will describe the characteristics of α-Si transistors, their fabrication, and the construction of liquid crystal displays employing them. Their application to a variety of display systems, including reflective, transmissive, projection, black and white, color, and video, will be illustrated.

Introduction

During the last three years there has been remarkable progress toward developing active matrix liquid crystal displays. These are displays that employ a thin film transistor (TFT) or other nonlinear circuit element to control each pixel in a matrix display. During this time, at six international display symposia(1), over 20 organizations have presented more than 40 papers describing their work. Highlights include a 4.25 inch diagonal full color video display from Suwa Seikosha(2). This 480 x 480 line polysilicon TFT addressed display has performance comparable to a small color CRT. A larger 7.23 inch diagonal color information display addressed by amorphous silicon (α-Si) from Hosiden Electronics has been demonstrated(3). Recent commercial active matrix display product announcements include a cadmium selenide TFT display module from Panelvision and a color pocket TV from Suwa Seikosha.

With several semiconductor technologies (amorphous silicon, polysilicon, cadmium selenide, and others) having demonstrated suitable TFT performance for display applications, a preferred technology can be chosen on the basis of yield, reproducibility, fewest number of mask steps, substrate material availability and cost, and the applicability to other devices. Amorphous silicon TFTs can be formed in a simple process over large areas on glass substrates and we believe that this will be the preferred technology(4).

For many aircraft applications, amorphous silicon addressed LC displays have the obvious advantages over CRTs that include lifetime, volume, weight and ruggedness.

Basic Device Operation

The basic circuit of a TFT controlled liquid crystal (LC) display is shown schematically in Figure 1. Each pixel has a TFT acting as a switch connected between the vertical data lines and the liquid crystal pixel electrode. The gates of each row of TFTs are connected to a gate line. The second electrode of the LC pixel capacitor is the common electrode on the cell coverglass.

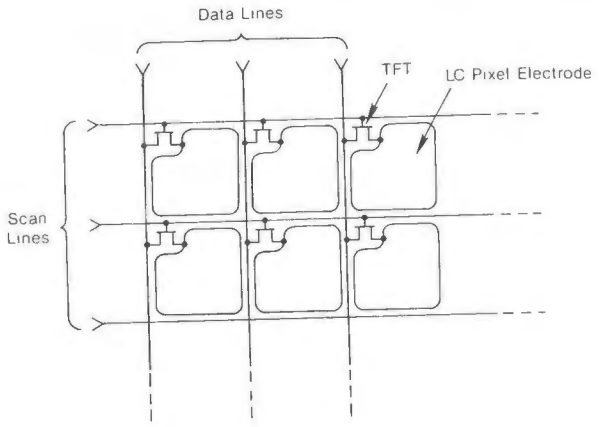


Figure 1
Schematic of TFT Driven Liquid Crystal Display

In operation the display is addressed a line at a time; that is, data voltages are applied to the vertical data lines, then a voltage sufficient to turn on the FETs is applied to a gate line. The LC capacitors in that line then charge to the voltages of the data lines. The gate line voltage is then reduced to a level sufficient to turn off the TFTs, thus storing the data voltages on a line of LC capacitors. Each line in the display is addressed sequentially in this manner; then, polarity of the data voltages is inverted and the refreshed sequence is repeated to produce an AC voltage on the liquid crystal pixel.

The minimum performance requirements for the TFT and the liquid crystal material can easily be derived as follows:

in order to avoid flicker the LC drive frequency must be >50 Hz. There are two refresh periods (T) per cycle, thus $T \leq 10$ milliseconds. If there are N lines in the display the maximum line address time t is

$$t=T/N.$$

The FET requirements are then: 1) the response time of the TFT must be short compared to t. 2) the on current I_{on} must be sufficient to fully charge C_{LC} during the line address time. That is,

$$I_{on} > C_{LC} \cdot V_{on}/t,$$

where V_{on} is the maximum data voltage. 3) the off or leakage current I_{off} must be small enough that it doesn't cause the voltage on C_{LC} to significantly change between refresh cycles.

$$I_{off} << C_{LC} \cdot V_{on}/T$$

The response times of α-Si TFTs are limited by the charging time of the gate capacitance through the channel and this is given by $L^2/\mu v$ where L is the channel length and μ is the effective electron mobility (typically $\mu = 0.3 cm^2/Vsec$). The measured switching time for a 5 micron channel length device is ~0.25μs in agreement with the above equation. For a 500 line display, n = 500, the line address time t = 20 microseconds. Therefore these devices easily meet the first requirement above.

To satisfy the second and third requirement, the I_{on} and I_{off} current ratio

$$\frac{I_{on}}{I_{off}} \geq 5000.$$

A typical current vs. gate voltage characteristic taken at room temperature is shown in Figure 2; it shows I_{on}/I_{off} ratios greater than 10^7 can be achieved. The temperature dependence of I_{on}, like the mobility μ, is slight; increasing by a factor of about 2 from 30° to 90°C. Of more concern is the temperature dependence of I_{off}. Data in Figure 3 shows about a factor of 10 increase in I_{off} from 30 to 90°C. But even at 90°C the I_{on}/I_{off} ratio is more than adequate for LC display applications.

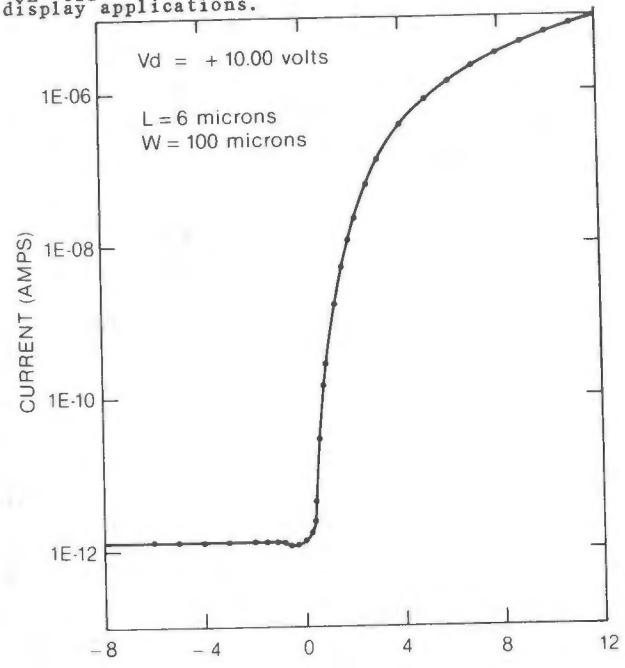


Figure 2
Typical Drain Current vs. Gate Voltage for an α-Si TFT

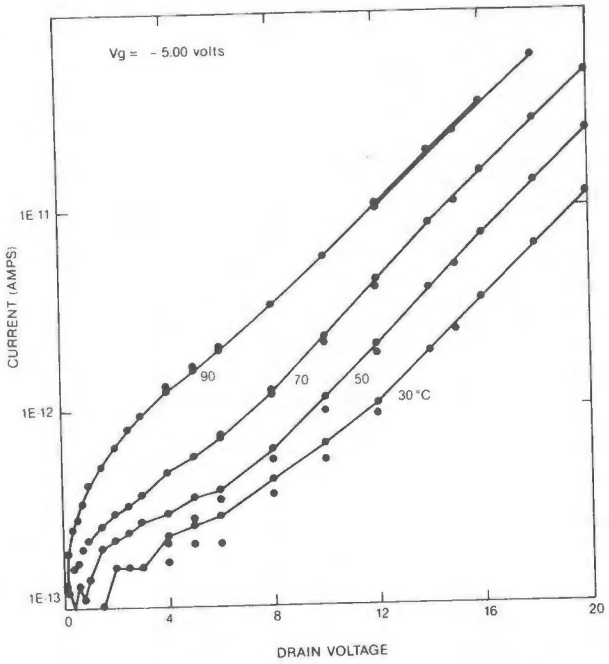


Figure 3
Off Current vs. Drain Voltage for an α-Si TFT for Temperatures Between 30°C and 90°C

The required magnitudes of I_{on} and I_{off} depend on C_{LC} or equivalently the display resolution. For a 64 lines per inch display $C_{LC} \approx 1$ picofarad; therefore the requirements are $I_{on} > 0.5$ microamps and $I_{off} < 100$ picoamps. The device shown in Figure 2 with a 6 x 100 micron channel dimensions easily satisfies these requirements. The FET I_{on} and I_{off} scale with channel dimensions are proportional to W/L where W is the channel width. Higher resolution displays use a correspondingly smaller width device.

A basic requirement for the liquid crystal material is that its resistivity must be high enough that it does not significantly discharge C_{LC} in a time T. That is, the LC time constant

$$\tau_{LC} = R_{LC}C_{LC} = \rho_{LC} \cdot \varepsilon_{LC} > T.$$

This can be met with most commercial field effect LC materials at room temperature. At elevated temperatures the time constant τ_{LC} decreases dramatically at about the same rate as the viscosity. To achieve adequate high temperature time constants proper choice of LC materials in special purification techniques are used. Figure 4 shows data on two purified commercial LC mixtures, a biphenyl and a phenyl cyclohexane (PCH) mixture. The higher resistivity and smaller temperature dependence of the PCH mixture permit its use to about 90°C.

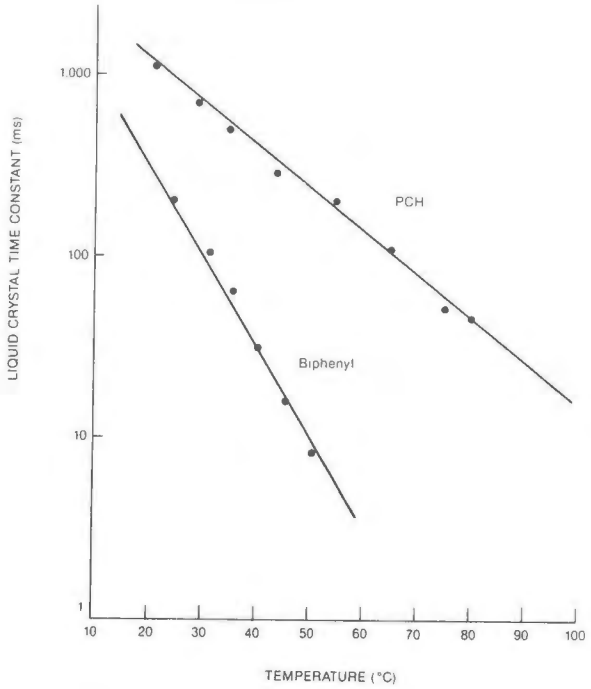


Figure 4
Liquid Crystal Material Time Constant vs. Temperature for a Purified Biphenyl and Phenylcyclohexane (PCH) Mixture

Device Fabrication

Figure 5 shows a cross-section of an α-Si FET. The silicon nitride and silicon layers are deposited in a single pump down by plasma enhanced chemical vapor deposition (PECVD) using SiH_4 with the addition of NH_3 for the nitride and PH_3 for the n^+ layer at a substrate temperature of 300°C. Using a plasma instead of high temperature to decompose the gases permits high quality films to be deposited at much lower substrate temperatures than conventional CVD. At these low temperatures, the deposited films are amorphous and incorporate between 8 and 20% hydrogen. The hydrogen passivates the dangling bonds in the α-Si film resulting in a high quality semiconductor material. The low temperature process allows the use of glass substrates.

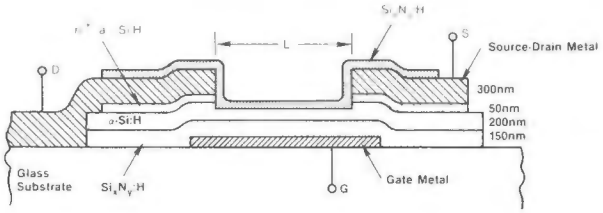


Figure 5
Cross Section View of an α-Si Thin Film Field Effect Transistor

A cross-section of a LC cell is shown in Figure 6. On the top of each FET is a thick light blocking layer, since light absorbed in the channel of the FET would cause photoconductive leakage. The gate electrode blocks light from the other side. This light blocking layer acts as distributed LC cell spacers; it is formed with a thickness equal to the desired LC layer thickness. The coverglass is bonded to the substrate with a parameter seal and the cell is filled with a volume of liquid crystal material to just fill the cell and the coverglass is in contact with the spacers. This technique produces extremely uniform thickness LC cells with D < 6 microns. These thin cells are desired for fast LC response time and wide viewing angle.

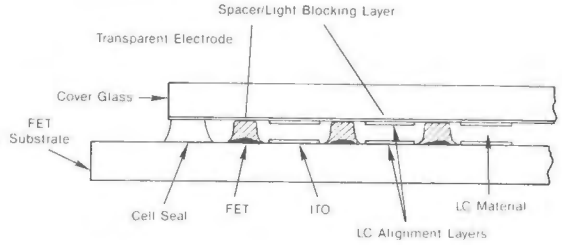


Figure 6
Cross Section of a TFT Driven Liquid Crystal Cell

Display Applications

The basic α-Si controlled LC displayed device can be used in a wide variety of applications ranging from simple reflective information displays to full color video and as either direct view or projection displays. Several of these applications which are useful for aircraft applications will be described below.

The basic design for a direct view panel display is shown in Figure 7. For a black and white binary (no grey scale) information display, the display cell uses a transmissive twisted neumatic LC effect. This requires polarizers on both sides of the cell. A suitable high brightness light source is a miniature fluorescent lamp. Figure 8 shows a photograph of a 2 x 2 inch 100 lines per inch display with this type of construction. The performance possible with this system is a contrast ratio greater than 50:1 and a display brightness of about 25% of the surface brightness of the lamp (greater than 1000 ft.L).

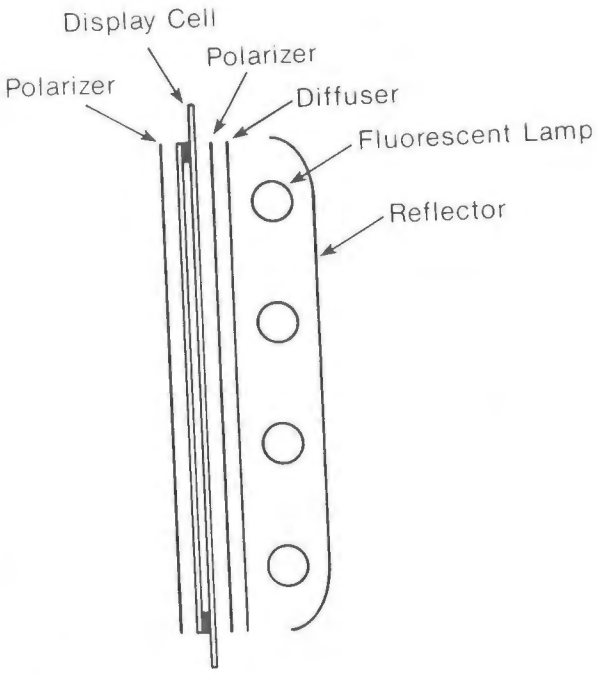


Figure 7
Construction of a Transmissive LC Display Using a TFT Driven Twisted Nematic Display Cell

Figure 8
Photo of a 2 x 2 inch , 100 lines per inch a-Si TFT Driven Transmissive LC Display

There are many possible variations to this basic design. For video, grey scale could be achieved by replacing the binary data drivers with analog drivers to modulate the voltages stored on the pixels. Color can be achieved by placing an array of color filters inside the display cell as shown in Figure 9. Suitable thin color filter arrays have been demonstrated using dyes absorbed into a gelatin film in selected areas defined by standard photolithography techniques(5).

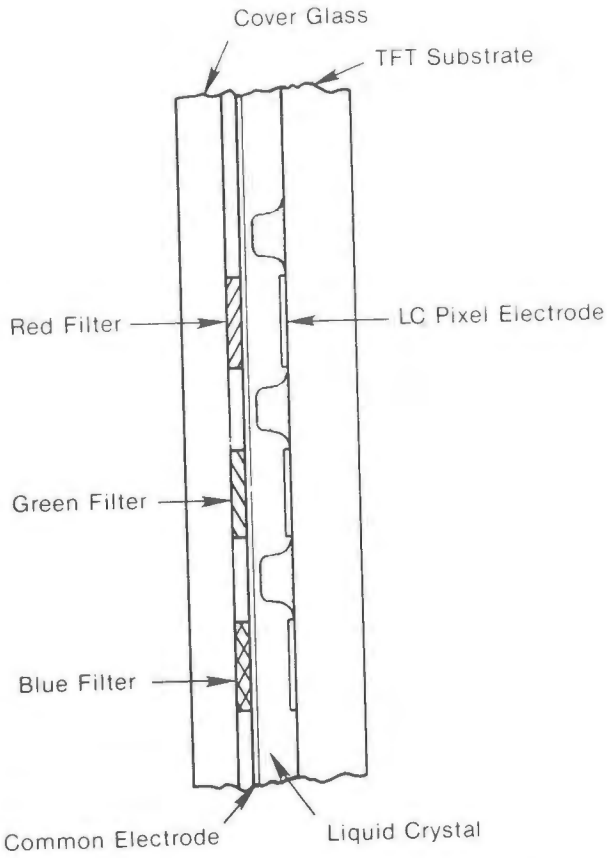


Figure 9
Liquid Crystal Cell Incorporating Internal Color Filters

Reflective displays can also be constructed and are desired where low power consumption is required. The preferred position for the reflector is inside the cell rather than behind the rear glass wall since parallax would be a severe problem on high resolution displays. With the reflector inside the cell the twisted nematic effect could not be used, instead, a dichroic LC effect which requires either one front or no polarizer is preferred. For the simple structure shown in Figure 1, only about 70% of a pixel cell is active area; the remainder is occupied by drive lines and spaces and the TFT. Therefore, the reflective contrast ratio or brightness will be less than for direct driven reflective LC displays.

An example of a projection display using the basic transmissive cell is a high brightness head up display (HUD), Figure 10. Since light can be generated by an efficient bright source such as a small high pressure mercury arc lamp, a display brightness on the order of 10^5 ft.L appears possible.

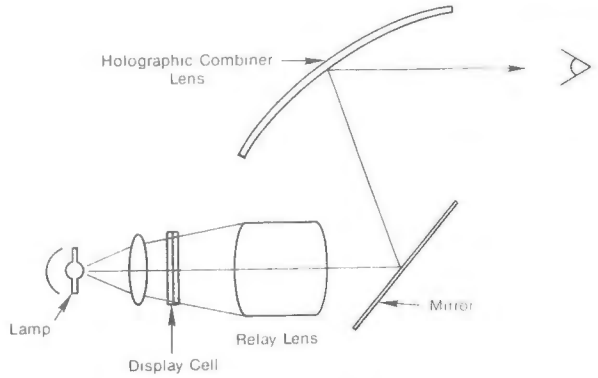


Figure 10
Head Up Display Using a Transmissive α-Si TFT Liquid Crystal Display Cell

Acknowledgements

We are indebted to H.S. Cole for the liquid crystal materials work, S. Aftergut for the color filter development, and the entire technical support staff.

References

(1) 1982, 1983, and 1984 Society for Information Display International Symposium: 1982, 1983 and 1984 International Display Research Conference.

(2) S. Morozumi, et al., 1984 SID International Symposium Digest, p. 316, 1984.

(3) Y. Ugai, et al., 1984 SID International Symposium Digest, p. 308, 1984.

(4) G.E. Possin, D.E. Castleberry, W. W.W. Piper, H.G. Parks, 1984 International Display Research Conference, 1984.

(5) T. Uchida, Proceedings of the 3rd International Display Research Conference, p. 202, 1983.

APPLICATIONS OF VOICE INTERACTIVE SYSTEMS -- MILITARY FLIGHT TEST AND THE FUTURE 84-2660

Carolyn A. Moore*
Manager of Crew Station Simulation
VERAC, Incorporated
San Diego, California

R. Douglas Moore*
Advanced Avionics Systems Engineer
VERAC, Incorporated
San Diego, California

Dr. John C. Ruth*+
Vice President of Marketing
and New Business Development
McDonnell Douglas Electronics Company
St. Charles, Missouri

Abstract

Due to the ever increasing complexity of fighter aircraft, new methods of command and control must be established to enable the pilot to fulfill his current and future mission requirements. Voice recognition and synthesis technologies provide a powerful new tool to the pilot to optimize aircraft command and control. The integration of an interactive voice system in an aircraft cockpit is a new discipline that imposes many unknowns and difficulties both in the hardware technology areas and in the pilot-cockpit interface implementation areas. Advances in recognizer/synthesizer hardware development, and in recognition algorithm development are progressing rapidly due to the combination of new hardware technology and the availability of data acquired from voice recognition flight tests. The definition of the pilot-synthesizer and pilot-recognizer interfaces requires further development and study in order to optimize cockpit operation in a high workload environment. This paper details possible methodologies that can be used in designing and optimizing the pilot-cockpit interface for an interactive voice system.

Introduction

As the complexity of fighter aircraft avionics increases, new problems in command and control arise that can no longer be successfully addressed by traditional control and display methodologies. Automation and Artificial Intelligence techniques can solve some of these problems by reducing the amount of raw data that the pilot must mentally process, but who or what will control these Automation and AI programs or subsystems?

The concept of the "pilot as a manager" evolves as avionic systems become more and more sophisticated. The pilot will not only need to monitor and direct the various functions, but will need to access specific data or ask a subsystem for relational data (e.g. "What is the highest priority target I can reach with my remaining fuel?"). If designed to optimize the pilot-cockpit interface, voice recognition and synthesis will be an efficient form of communication for advanced aircraft.

In this paper, three distinct areas of voice interactive systems will be addressed. First, the benefits resulting from the use of interactive voice recognition and speech synthesis as a viable tool in current and advanced aircraft will be analyzed. Second, current voice recognition systems for military aircraft will be discussed with an emphasis on the AFTI/F-16 Phase I flight test effort. Finally, a detailed look at how interactive voice systems can be implemented in current and future aircraft systems will be undertaken. The lessons learned from voice command flight tests on the AFTI/F-16 Phase I aircraft will be examined and possible solutions to operational problems will be discussed.

Why Interactive Voice in the Cockpit?

Today's single-seat tactical missions require a high performance, coordinated, all weather strike aircraft which can perform in a hostile, high-threat environment. The current hands-busy/eyes-busy mission severely taxes both pilot and aircraft. Furthermore, as the mission complexity increases, the ability of the pilot and aircraft to successfully perform the mission decreases. In the future, mission requirements in high threat environments will result in even greater complexity and workload, potentially compromising the effectiveness of manned aircraft. By augmenting or automating certain cockpit functions, it will become possible to reduce the button and switch workload of the pilot in order to better use his managerial skills.

Military aircraft pilots, weapons systems operators, and helicopter pilots are all faced with the critical problem of excessive workload. The evolutionary design of aircraft avionics subsystems and assorted weapons has resulted in a tremendous burden being placed on crew members to serve as real time integrators of aircraft systems in order to meet mission requirements and manage aircraft systems. A pilot is confronted with a myriad of displays, controls, and information in the head-down mode which competes for his attention with information received out the window and via radio. The F-16C, for example, has over 200 basic

*Member IEEE
+Member AIAA

information arrangements or "pages," each with numerous possible permutations and combinations being shown on four displays. The number of pages will increase dramatically with the addition of new systems and capabilities. In a highly integrated ATF cockpit of the 1990's, a conventional display approach could generate literally hundreds of "pages" creating an impossible pilot-aircraft operating interface. During critical mission phases such as air-to-ground weapon delivery, the pilot must maintain hands-on/head-up control of the aircraft. Since the use of his hands and eyes is maximized and often overloaded during these critical mission phases, it is difficult and disadvantageous for the pilot to remove his hands from the stick and throttle and to look around in the cockpit to control, for example, display modes, parameter selection and radio communications control.

It is therefore imperative to optimize the pilot-cockpit interface by augmenting the selection and control of displays and switchology in the avionics suites, and by enhancing perception of mission/aircraft/threat relationships which might be exploited or changed to achieve a higher mission success or survival probability. This will reduce pilot workload and increase situation awareness.

Speech and hearing are highly efficient means of communication. They are two of the most under utilized methods of control in the cockpit. If properly implemented and thoroughly integrated into the avionics, voice recognition and speech synthesis can take advantage of the pilot's speech and hearing capabilities and help optimize the pilot-cockpit interface.

General Dynamics' Voice Recognition Flight Test

McDonnell Douglas Corporation, Crouzet S.A. Division Aerospatial, Marconi Avionics Limited and other aircraft manufacturers have initiated voice recognition and synthesis avionic integration programs; however, the General Dynamics Fort Worth Division has led the way in airborne implementations to date.

Since 1981, General Dynamics' Fort Worth Division has been vigorously evaluating the use of the pilot's voice as an alternative method of achieving interaction between the pilot and the weapons systems. The ultimate goal is to off-load tasks from the pilot's hands to his voice, thereby enabling him to exercise hands-on control of the airplane a higher percentage of the time. This is particularly crucial during critical periods of the mission.

General Dynamics has structured a three-phased program to achieve this goal; Phase 0 was a laboratory simulation evaluation completed in 1982, which verified that voice control of certain avionic functions can reduce pilot workload. Phase I was a limited airborne environmental flight test completed in August 1983, which verified voice recognition operation in the airborne environment and accumulated data for future recognition algorithm development. These programs were followed by Phase II, which is a functional utilization evaluation flight test program. Both Phase I and II required a significant ground simulation effort prior to flight test. The Phase II simulation study is scheduled to be completed in 1984 and Phase II flight test is scheduled for 1985. A brief analysis of the Phase I flight test will now be discussed.

Phase I

The goals of the Voice Command Phase I flight test program were: (1) to determine the effects of the airborne environment on the pilot's voice and on the recognition algorithm performance, (2) to develop a reliable Voice Command System (VCS), (3) to demonstrate feasibility in airborne applications, and (4) to establish a basis for further functional studies.

Voice Recognizer Development. A suitable voice recognition system that could withstand the severe airborne environment had to be built and tested. In 1981 a survey of various recognizers was conducted and after basic testing it was realized that commercial recognizers couldn't cope with the vibration, ambient noise, G forces, oxygen mask and breathing noises that occur in the airborne environment. These environmental factors affect the speech pattern of the pilot and degrade the quality of the acoustic speech signal itself leading to a lower recognition reliability. Since the system templates were trained in a benign environment, these degradation factors of the environment had to be resolved by the voice command system.

Two vendors were willing to build military qualified prototype recognizers. Each recognizer underwent strenuous testing in the AMRL laboratories to ensure a minimum recognition accuracy of 50% using flight test audio tapes consisting of pilots repeating a predefined vocabulary under varying noise and G-force conditions. In addition to this algorithm testing, both recognizers had to meet flight test hardware specifications.

Voice Function and Vocabulary Section. Since the main purpose of Phase I was to evaluate voice recognition in an airborne environment, the pilot-cockpit interface with the voice recognizer was not configured for a production aircraft. Rather, a limited subset of avionic functions was considered in order to ensure safety-of-flight (e.g., there is no weapon release voice command), simple pilot operation, and an easy software implementation.

Table 1 lists the 34 vocabulary words according to these corresponding functions: mission phase selection, specific Multifunction Display (MFD) selection, MFD page selection, MFD page option control, data entry option control, and MFD button/numeric data entry selection. Figure 1 shows the AFTI/F-16 cockpit, where the MFD switches and the Mission Phase switches are identified. By mechanizing these functions, the pilot will be able to operate more in a head-up, hands-on manner. The 34 word vocabulary was thoroughly tested in the General Dynamic Research & Engineering Simulator and in the AMRL laboratories with both vendor's recognizers to ensure all rhyming and ambiguous words had been eliminated.

Most of the Phase I functions were very simplistic--each voice command corresponded to one switch depression except for the 'LEFT' and 'RIGHT'

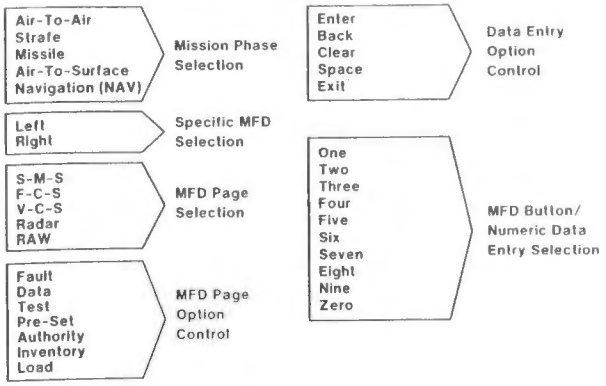


Table 1 AFTI/F-16 Phase I Voice Command Vocabulary

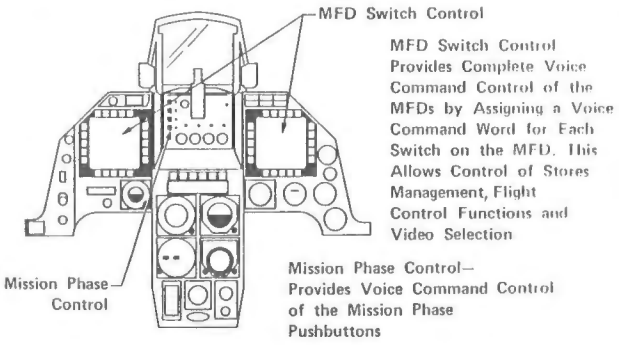


Fig. 1 AFTI/F-16 Phase I Cockpit Display

command used with a page selection command. For example, when the pilot requested 'RADAR' for the MFD Radar page, which MFD was he talking to (see Figure 1)? The 'LEFT' and 'RIGHT' commands specified the left or the right MFD and were used to move the Display-of-Interest (DOI) indicator (a small highlighted square that appeared in the lower right corner of the MFD). Therefore, after 'LEFT' or 'RIGHT' was spoken, all following voice commands addressed that particular MFD.

Vocabulary ease of use can be achieved by creating a vocabulary and a corresponding grammar/syntax structure that is easy to remember (especially in emotionally stressful situations), concise enough to perform most of the necessary functions, and consistent. For example, in Phase I, voice commands are analogous to the label of the switch they are activating, and only one syntax node (consisting of the entire vocabulary) and a two-word grammar structure was needed since the vocabulary size was small enough. Phase II and successive work at VERAC, Inc. will address a more comprehensive word/grammar/syntax structure that will control a more sophisticated avionics cockpit.

Voice Recognizer Control. During Phase I, current recognition technology had progressed to the point where speaker dependent, isolated word recognition was possible in the airborne environment. The problem in Phase I was how to limit what the voice recognizer "hears". For example, most background noise (engine, wind microphone, breath, oxygen mask, and EMI) and pilot conversation must be eliminated or reduced since the voice recognizer must respond only to the current command and not to any miscellaneous noise or conversation. This problem was solved in three ways: First, a special amplifier and microphone cable hookup were built in order to reduce extraneous microphone and intercom noise: second, the recognition algorithms were tailored to filter out noise--especially breath noise: and third, a hands-on-control (HOC) Throttle switch (UHF) was used to enable the voice recognizer to listen to audio inputs (i.e., whenever the pilot wanted to voice a command, he would depress the UHF switch while he was speaking). In order to provide head-up visual feedback of the voice recognizer being enabled, a small square was displayed on the lower right corner of the Head-Up-Display (HUD) whenever the UHF switch was depressed.

Voice Template Manipulation. With speaker dependent, isolated word recognition, all 34 vocabulary words must be trained and stored for each pilot. A Data Transfer Unit (DTU) was used to load/save the pilot's word templates from/to a Data Transfer Cartridge (DTC). Each pilot carries his DTC with him and loads his templates into the voice recognizer prior to takeoff.

Voice template (voice prints stored in the recognizers) manipulation was accomplished through several MFD Voice Control System (VCS) pages: TRAIN, UPDATE, VERIFY, ACTIVE, and STANDBY. The TRAIN, UPDATE, and VERIFY mode pages were used to train, update and verify word templates respectively. In order to train the vocabulary, a pilot was required to select the TRAIN mode page and repeat each vocabulary word three or more times until the VCS had obtained three good voice templates. In order to verify the trained words, the pilot selected the VERIFY mode page, repeated each vocabulary word once and observed if the correct word was recognized.

The UPDATE mode is a special subset of the TRAIN mode. While in a flight, if a pilot noticed a particular voice command was not being recognized correctly, the pilot could simply go to the UPDATE mode and update the word templates once. This saved time since less word repetitions were needed to update rather than train a word. It also resulted in a higher word recognition rate since new templates were added in with old templates in the queue to be searched.

The ACTIVE and STANDBY VCS pages were selected whenever the pilot wanted voice commands to be acted upon by the avionic computers or to be deactivated.

Failure/Status Information. The display of failure/status information was separated into three areas: (1) Failure information that was critical to the pilot was displayed on a Pilot Fault List (PFL) page as various fault entries in the PFL list. An example of a PFL entry is a total VCS 1553 multiplex bus failure, where the VCS is no longer communicating with the rest of the system. (2) Failure information that was necessary for

maintenance action but not critical enough to interrupt the pilot was displayed on a Maintenance Fault List (MFL). An example of this is a partial VCS bus failure where the VCS can still communicate. (3) VCS template manipulation status/failure information was displayed in several windows on the STANDBY, ACTIVE, TRAIN, UPDATE, and VERIFY pages. Some examples are LOAD, SAVE, current recognized word, NOT RECOGNIZED, NEXT, END, UNTRAINED, etc. The status indicators 'NEXT' and 'END' were used respectively to signify when a word was trained and the pilot could increment to the next vocabulary word, and when the pilot had trained all of the vocabulary.

Flight Test Results. AFTI/F-16 Phase I flight test results are shown in Figure 2 and 3 for varying noise levels (96 dB - 110 dB) and varying load factors (1 g - 5 g). Noise levels were varied primarily by changing the aircraft altitude (the lower the altitude, the greater the ambient air noise), the defog lever setting, the center console "eyeball" nozzle, and the ECS (Environmental Control System) settings. Load Factors were varied by conducting 1-5 G aerial maneuvers while the pilot was repeating the vocabulary. As expected, voice distortion was at its greatest when the pilot was experiencing high G forces. High in-flight recognition accuracies for the worst case conditions of 5 Gs and 110 dB reflect the tailoring of the recognition algorithms for the severe airborne environment.

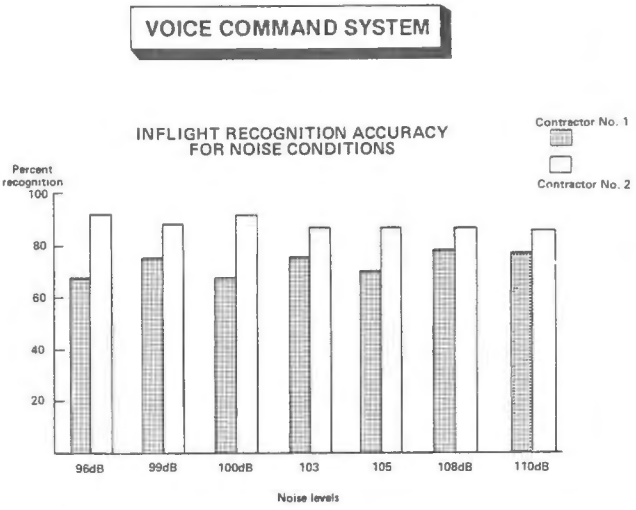


Fig. 2 AFTI/F-16 Phase I Flight Test Results for Varying Noise Levels

The data from the Phase I flight test program established the feasibility for the use of a voice interactive system in the harsh environment of a high performance tactical fighter. The payoff for voice will have to be more than just an alternate means of emulating the manual switchology. One voice command will have to be able to replace a sequence of manual switch operations or to provide control functions that were heretofore unavailable in a manually controlled cockpit. The Phase II portion of the General Dynamics' program and programs underway at VERAC, Inc. will address the operational utility of voice in the aircraft.

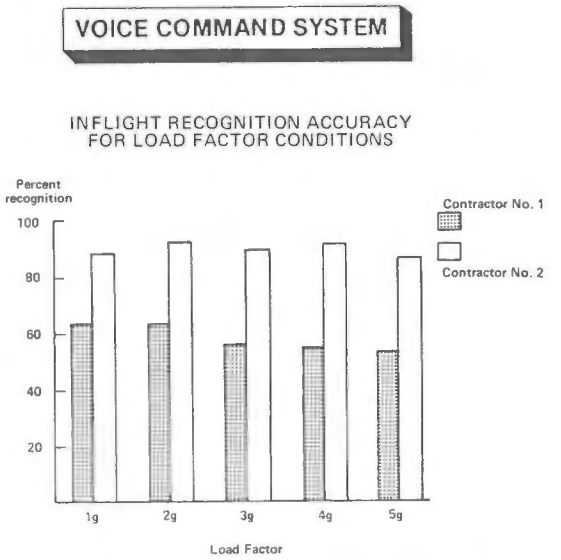


Fig. 3 AFTI/F-16 Phase I Flight Test Results for Varying Load Factors

Advanced Interactive Voice Implementations AFTI/F-16 Phase II and Beyond

The main goal of the General Dynamics AFTI/F-16 Phase II program and related work at VERAC, Inc. is to optimize the pilot-cockpit interface, assuming that voice recognizer technology can progress to the point where it is even more reliable in the airborne environment. Voice recognition and speech synthesis are not viable avionics tools unless they actually reduce pilot workload, reduce critical function performance times, improve an already existing PVI mechanization and allow the pilot to maintain head-up, hands-on control of the cockpit. The implementation must, of course, allow the pilot to operate an interactive voice system with as little effort as possible.

A simple and consistent system design philosophy is crucial to the success of any design project. The ongoing design must be reviewed and evaluated constantly to ensure that the final product incorporates the design philosophy and goals. Some of the design guidelines that have been developed are: (1) keep the number of system inputs and outputs to a minimum to allow for a simple and efficient operation, (2) do not distribute basic system functions over various subsystems (keep high cohesion within each subsystem), (3) optimize the system performance and the system response time by weighting the system mechanization (task-tailoring) so time critical and high payoff commands take the shortest amount of time to implement, (4) allow the system and the pilot-cockpit interface to have a consistent mechanization, (5) apply artificial intelligence techniques to the System Design and pilot-cockpit interface, and (6) design the pilot-cockpit interface to be as natural as possible (emulate human interaction).

Voice Recognition Implementation

For a voice command system, the design of the pilot-cockpit interface falls into three categories: (1) voice-controlled candidate function selection, (2) vocabulary definition, and (3) displays and switchology definition. The selection of candidate functions for voice recognition is very aircraft-dependent and will not be discussed in this paper; however, possible methodologies for determining the proper vocabulary, displays and switchology will now be discussed.

Vocabulary Definition. Five basic areas must be investigated when defining the vocabulary. These are determining: (1) vocabulary size, (2) word ambiguity, (3) the grammar structure, (4) artificial intelligence implementations (default/heuristic structures), and (5) syntax structures.

Vocabulary size is bounded by several factors: (1) too large a vocabulary is difficult for the pilot to remember and use properly, (2) too large a vocabulary is difficult for a voice recognizer to search through in an adequate response time, and (3) too small a vocabulary may not cover all of the time-critical, high-workload functions that need to be implemented. Phase II uses a 100 word vocabulary (maximum capability of the recognizer) that enables the pilot to control some of the critical functions, and still remember and easily use the voice commands.

Word ambiguity can be divided into two categories: (1) word ambiguity for the voice recognizer such as for rhyming or non-robust words (peculiar to each recognition algorithm), and (2) word ambiguity for the pilot such as voice commands that are not easy and logical for the pilot to use. Ambiguous words can be eliminated from the vocabulary by exhaustively testing how the recognizer and the pilot respond to vocabulary words under various operating conditions.

Since voice command is an extension of the pilot's normal verbal response, it is essential that a natural, human-like grammar structure be used. A natural grammar structure is easy to use because: the pilot does not have to learn an unnatural language structure, the pilot does not have to remember a set of unrelated commands, and the language structure can prompt the pilot on what to say next.

In evaluating the types of candidate functions that were needed in a particular advanced cockpit, it was discovered that all voice functions could be performed by either selecting or cancelling an individual object from a set. Cancelling an object may seem redundant since what is really happening is that the former object is being reselected, but there are times (especially in stressful conditions) where a pilot may want to go quickly back to the status quo without having to reselect all of the functions he needs.

In order to select/cancel an individual object from a set, the verb (select or cancel), the set to be acted upon, and any individual selection within that set must all be identified. For example, the sentence 'SELECT RADAR MENU' has 'SELECT' as the verb, 'RADAR' as the set or display type in this case, and 'MENU' as the specific radar display. If these three grammar states are linked together, along with a fourth state for data entry, the resulting state-driven grammar structure is shown in Figure 4.

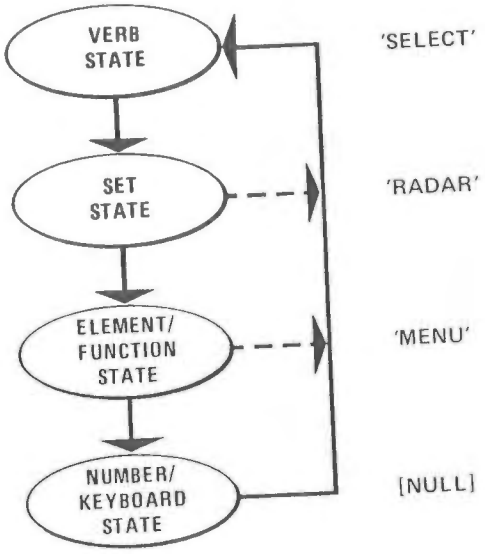


Fig. 4 An Example State-Driven Grammar Structure

In this state-driven grammar structure, after execution of the 4th state, the system is automatically reset to the 1st state again, freeing the pilot from having to reset the system via a special switch or keyword. Voice commands are executed as each word is spoken to the best of the voice interpreter's ability resulting in quick execution times since the interpreter is not waiting for the system to be reset. In Figure 4, brackets indicate a word that is not spoken, but is supplied by the voice interpreter for correct grammar interpretation.

What happens when a sentence consists of only a verb and an element of a set (e.g. 'SELECT GROUNDMAP'), or a sentence consists of only a set (e.g. 'RADAR')? The voice interpreter has to be able to 'fill in' all of the missing states--such as where there is a 'null' state (e.g. function has no data entry capability) or an implied state (e.g. in the sentence 'SELECT GROUNDMAP', the radar display set is implied). This is where default structures come in.

Default structures are a very simple artificial intelligence application where the voice interpreter is aware of the current state of the avionics and of what the pilot has already said, and then can make a decision about what should be done for skipped grammar states. In order to do this, default values are assigned to particular grammatical states in particular voice command sentences. Default values, at least for this generation of VCS systems, should always be non-critical, non-destructive, unambiguous commands that when used will never cause an adverse system response. An example is shown in Figure 5. All vocabulary words shown in parentheses are denoted as default values which can be spoken or not. If the pilot said 'PRIORITY', the system would respond the same as if he had said 'SELECT RADAR PRIORITY ONE'--in this case the radar assigns priority numbers to all displayed radar targets and locks on

to the highest priority target. In order to make default structures work properly, the parsing of the grammar must be done very thoroughly. All possible vocabulary combinations and unique keywords must be carefully defined. As voice command systems expand with the ever increasing complexity of avionics, default structures can expand into more elaborate and powerful structures.

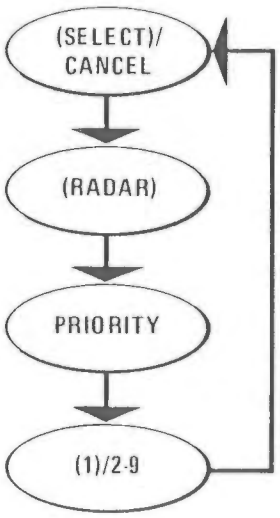


Fig. 5 An Example Sentence Using Default Values

While default values are concerned with the pilot-cockpit interface (spoken grammar), syntax structures deal with the voice recognizer to voice interpreter interface. A syntax structure is a structure showing the interrelationships between all syntax nodes. A syntax node is the partitioning of allowable words by grammatical element and avionic state (e.g. the list of acceptable displays in a given avionic state). Current systems have very rigid syntax structures. For example, if the word 'RADAR' was spoken while the interpreter was looking for a verb, the recognizer would not recognize 'RADAR' since 'RADAR' is not in the verb syntax node. However, since we have a grammar/default structure that allows the pilot to skip over grammatical states, our syntax structures must be very flexible. Therefore, for intelligent systems, if a word is not found in one syntax node, then other syntax nodes can be searched according to a predefined order for each grammar state.

This allows the pilot not only to skip over words in a sentence, but also to change his mind in mid-sentence without any adverse effects. Changing his mind in mid-sentence will result in a slightly slower recognizer response since the voice recognizer has to search more syntax nodes, but with a good syntax node partitioning and a good parser, the time difference will be imperceptible.

Display and Switchology Definition. The remaining task in determining the pilot-cockpit interface is the definition of all voice command related displays and avionic switchology.

In Phase I, the UHF Throttle switch was used to enable the voice recognizer to listen to audio inputs. This mechanization was used only as a flight test workaround since every time a voice command was spoken, it was broadcast over the UHF channel. Another problem was observed in Phase I flight test: having the pilot say 'LEFT' or 'RIGHT' in order to select a new display-of-interest. Quite frequently pilots would forget to select the appropriate MFD and would be making display changes on the wrong MFD--this resulted in the pilots having to look down and reset all of their MFDs to the desired configuration.

In Phase II, both problems were simply solved by expanding the current display-of-interest (DOI) selection switch to include voice. AFTI Phase II has two four-position DOI selection switches on the Side Stick Controller, giving the pilot the capability to select either MFD, the HUD, and other subsystems simply by depressing one of the eight combined switch positions.

Popping a DOI switch (holding the switch down for less than 0.5 seconds) reassigns the system DOI to the new selected subsystem (there is a DOI symbol that appears on the current system DOI display). By holding any of the DOI switches down for longer than 0.5 second (exact time to be determined in Phase II Research & Engineering Simulator studies), the voice recognizer is enabled and a temporary voice DOI is selected (so the pilot can manipulate a different display from what the avionics system is looking at) for as long as a DOI switch position is depressed. This mechanization optimizes the manual/voice interface and eliminates a display-of-interest syntax node.

In Phase I, quite frequently the pilots would become confused about where they were in a set of commands and whether or not the voice recognizer understood them; a heads-up display of recognizer status was needed to feed back to the pilot what the recognizer/interpreter understood. A window in the lower right-hand corner of the HUD was used to display the current recognized word if recognized by the VCS, a not recognized label if the spoken word was not recognized by the VCS, or the current recognized word plus a null action indicator if the spoken word was a valid vocabulary word but the sentence that was spoken was meaningless to the voice interpreter.

Pilots like to configure their displays (especially MFD displays) to a standard configuration throughout a mission phase. In Phase I every time the pilot wanted to change any function on a display page, he had to reselect his previous display configuration. This is not only annoying and a waste of time, but it can also be dangerous during a time critical, high workload situation. In order to solve this problem in Phase II, temporary display pages were created in addition to the existing permanent primary and alternate display pages.

Temporary display pages allow the pilot to display important pages quickly without destroying the existing primary and alternate display configuration.

If a function has an automatic display selection default value, then whenever the function was requested, the corresponding display page would appear as a temporary page. Figure 6 shows a typical temporary page where the corresponding

primary, alternate or temporary label is highlighted depending upon what type of display it is.

There are three ways to remove a temporary display: pop the corresponding temporary DOI switch, say 'CANCEL' (temporary page name)', or request a new permanent or temporary display. Temporary page capability allows the pilot greater flexibility in determining his cockpit configuration. Temporary display pages may not be suitable for all applications. Audio feedback or limited visual feedback (a specific window on a page instead of the entire page) can also be used to feed back information to the pilot.

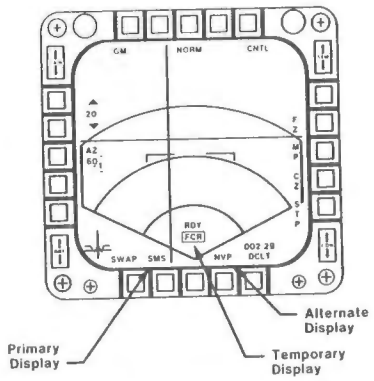


Fig. 6 An Example TEMPORARY DISPLAY PAGE

Speech Synthesis Implementations

Speech synthesis integrated with voice recognition into the cockpit environment will result in a more complete and optimized pilot-cockpit interface than would result using voice recognition alone.

Similar to a voice recognition system, there are three categories that must be considered when designing the pilot-cockpit interface: (1) synthesizer candidate function selection, (2) synthesizer vocabulary definition, and (3) the definition of unique speech synthesis displays and switchology. Specific candidate functions will not be discussed in this paper, however, possible methodologies for determining the proper vocabulary, displays, and switchology will be discussed.

Vocabulary Definition. Vocabulary definition can be divided into (1) identification of vocabulary components and their corresponding grammar structures; (2) prioritization of vocabulary components and identification of synthesis conventions (artificial intelligence implementations); (3) syntax structures definition; and (4) word ambiguity evaluations.

The identification of vocabulary components is directly derived from the candidate function selection and analysis. A basic list of components that would apply to most synthesis applications is: words, phrases, lists (soliloquy), and tones. Next, the characteristic type of each vocabulary component must be identified. For example, is the component a warning, a caution, a status, a response, or a "repeat input" request (used in conjunction with voice recognition)? For each of these typed components, a grammar structure must be identified in order to facilitate the interpretation of the synthesis commands into spoken words, and to prepare the pilot for the different types of responses he will hear (this decreases the amount of interpretation performed by the pilot).

Basic artificial intelligence implementations for synthesis are similar to those for recognition. Essentially, the vocabulary components must be prioritized and synthesis conventions must be established. Multiple voices within the cockpit due to radio communications and communications between aircraft crew members in multiple person cockpits pose a serious problem for synthesized speech transmission. Without establishing component priorities and synthesis conventions, synthesis would be almost useless and possibly would endanger the lives of the pilots by blocking other important communications. An example list of vocabulary component priorities, in descending order, is: warning, caution, status, response, "repeat input".

Artificial intelligence techniques can be used to implement synthesis conventions. The following speech suppression conventions are examples of output command conventions that could be used in the airborne environment: (1) establish speech suppression mode so the synthesizer won't speak unless spoken to, and (2) establish speech suppression capability at any priority level so the synthesizer won't speak unless incoming commands are of a higher priority than the current priority level. Some example input command conventions might be (1) put all incoming commands in a synthesis queue, and (2) the next-to-speak item in the queue is determined by the item priority and the time-of-arrival.

Different audible synthesized sounds can be used to relay information to the pilot; for example, some sound output conventions are: (1) use tones to announce new items in the queue (especially if the incoming commands are of lower priority than the current priority level and won't be spoken until the pilot changes the priority level or empties/lists the queue); (2) use different tones to announce different types of incoming items into the queue; and (3) use different types of synthesizer "voices" to portray different types of messages or priority levels -- there have been many studies in this area alone e.g., the effectiveness of male versus female voices.

Interrupt conventions must also be established for synthesized output. Some interrupt conventions might be: (1) voice recognizer control switch (push-to-talk) interrupts synthesized output to allow the pilot to speak commands; (2) do not interrupt externally generated voices (e.g., pilot, radios, intercom - multiple crew members) unless necessary; (3) a word cannot be interrupted; (4) a word can be overlayed by tones, etc.; (5) a phrase can be interrupted, and the entire phrase is repeated after the interruption; (6) a list or soliloquy of words can be interrupted, and the list or soliloquy is continued after the interruption from where the interrupted occurred, and (7) a list or soliloquy of phrases can be interrupted, and after the interruption, the phrase that was interrupted will be repeated entirely before the list or soliloquy continues. Obviously, determining the synthesis conventions that will be used and how they interact with each other is a very complex process that has direct impacts on the usability of the system.

Syntax structures for speech synthesis are similar to syntax structures for voice recognition except that the synthesis syntax structures deal with the speech synthesizer to speech interpreter interface. One example of an advanced syntax structure for synthesis might be to use multiple language syntax nodes for each grammar structure. Therefore, similarly to each pilot storing his voice recognition templates on a Data Transfer Cartridge, the proper language syntax structure can be selected by the pilot for synthesis output (e.g., one pilot may choose English, another French).

This would allow multi-national use of the synthesis system. The language dependent syntax nodes could be stored on a Data Transfer Cartridge for maximum flexibility or preprogrammed into the voice synthesizer.

Word ambiguity is an important factor in developing an optimal synthesis system. Any rhyming words or non-robust words for the synthesizer will be hard for the pilot to understand in a high work load situation; these words should be eliminated from the vocabulary. Care should also be taken to ensure that synthesized words are not 'spoken' out of context. All synthesized words should reflect the terminology the pilot is familiar with to increase understandability and ensure optimal operation.

Displays and Switchology. There are three basic areas that must be investigated in defining the displays and switchology: speech synthesizer control, synthesizer vocabulary word manipulation, and feedback of failure/status information.

Speech synthesizer control consists of selecting or displaying the priority levels, synthesis interrupt conditions, and the queueing of incoming commands.

Synthesizer vocabulary word manipulation addresses the problem of allowing a pilot to modify the synthesized words--e.g., a data transfer cartridge (DTC) may be used to store customized synthesizer words in several languages for international use.

The feedback of failure and status information from the speech synthesizer interpreter to the pilot is crucial. All critical information must be displayed on head-up displays or instruments, or spoken by the speech synthesizer (if it is still operational) in high priority messages. Non-critical information can be displayed on secondary head-up display pages, or on head-down displays, or through lower priority synthesized messages or not at all depending upon the type of data. As stated before, the quality and completeness of failure and status information presented to the pilot is often the critical factor by which a speech synthesis system is judged operational or not.

Future Cockpit Implementations

To fully optimize voice recognition and speech synthesis implementations, the entire avionics suite must be designed from the ground up with recognition and synthesis thoroughly integrated into it. A future cockpit suite, for example, might consist primarily of a Helmet Mounted Display (HMD) and an Interactive Voice System. Pilot input to the avionic computer systems would be through spoken commands to the voice recognizer and by manipulating Hands-on-Control switches. The HMD would provide the pilot the capability to enter data by "pointing" at objects with a crosshair on the display. The avionics would output concise information to the pilot via the speech synthesizer and the Helmet-Mounted Display. The traditional hands-busy/eyes-busy workload problems are reduced because the pilot does not need to look down or manipulate any non-hands-on-control switches to control the majority of cockpit functions. Combining advanced voice recognition and speech synthesis technologies with other emerging technologies will effectively reduce pilot workload and optimize cockpit control.

Summary

The use of voice recognition and synthesis integrated with aircraft cockpit displays is in its infancy, however, many lessons have been learned to date. Several simulator studies, e.g., AFTI Phase 0 study, have demonstrated that voice recognition can reduce pilot workload and increase situation awareness. The flight test of a voice recognition system in AFTI/F-16 Phase I proved that the use of voice recognition is an alternative to manual switchology in the airborne environment. Ongoing voice recognition algorithm and hardware development is progressing rapidly as the circuitry undergoes another phase of miniaturization and as new algorithms are developed based upon the environmental effects data obtained from flight test. Phase II and related programs at VERAC, Inc. and other companies are evaluating the pilot-cockpit interface and attempting to optimize it for voice recognition and speech synthesis. Methodologies for selecting candidate functions, defining vocabulary, and designing related displays and switchology are being established. In future aircraft, voice recognition and speech synthesis will be an integral part of the avionics suite.

Acknowledgements

The authors were employees of General Dynamics up to January 1984 and want to thank General Dynamics for the permission to use public data in this paper.

References

(1) J.C. Ruth, A.M. Godwin, and E.B. Werkowitz, "Voice Interactive System Development Program", presented at the Agard Avionics Panel Symposium, Blackpool, England, April 1981

(2) A.M. Godwin and J.C. Ruth, "Voice Command - The Next Threshold in Cockpit Operation", presented at the 4th Digital Avionics Systems Conference, St. Louis, Missouri, November 1981

(3) C.A. Moore, "Voice Command - The Next Threshold in Cockpit Operation", presented at NAECON '83 Conference, May 1983

(4) C.A. Moore and Dr. J.C. Ruth, "Use of Voice Integrated with Aircraft Cockpit Displays", presented at SID '84 Conference, June 1984

SESSION 11 SYSTEMS AND SOFTWARE - VERIFICATION AND TEST TECHNIQUES

Chairmen:

Edward A. Delanty
The Boeing Co.

John G. Weber
VERAC Inc.

This session addresses innovative approaches to the significant problem of verification and test of digital avionic systems and software. Equipment, software tools, techniques, and experience are included.

VERIFICATION TECHNIQUES FOR IMPROVING SOFTWARE QUALITY THROUGH AUTOMATED REQUIREMENTS DATA BASES

Glenn Lueders
Principal Engineer
Sperry Corporation, Flight Systems
Phoenix, Arizona 85027
(602) 869-1902

Abstract

Data base programs, such as the Sperry MAPPER system, have been used as a verification tool to control and provide traceability in software development as required by RTCA/DO-178, Software Considerations in Airborne Systems and Equipment Certification. Data bases can be established that associate the verification parameters for each software module and requirement. These parameters are operated upon by the search, sort, match, and update capabilities of the MAPPER system to extract reports to control the verification activity and ensure requirement coverage, module coverage, and traceability. Sperry Flight Systems has used the MAPPER system in the verification testing of the Boeing 757/767 flight management computer (FMC) applications computer program. MAPPER data bases were used for both system testing and detailed verification testing. This paper discusses the techniques and the advantages realized through the use of these techniques.

The Verification Testing Problem Posed by Large Systems

The software quality goals of traceability and total requirements coverage in verification testing are difficult to attain, but they are more difficult with a large system. Traceability and coverage problems inherent in large systems were resolved on the 757/767 FMC system and detailed verification testing by the use of automated requirements data bases. A discussion of these problems follows.

System requirements documents are typically organized by function, with subsections relating to submodes and operation in multiple configurations. As in actual aircraft operation, system testing is usually conducted by operational simulation scenarios that simultaneously exercise multiple functions, each in a single configuration and submode. With thousands of requirements and multiple submodes and configurations, it may be very difficult to confirm that a test has been conducted on each requirement in the relevant mode or configuration. Introducing RTCA/DO-178 functional criticality on each requirement, mode, or configuration adds another dimension to the coverage and traceability problem.

The detailed verification testing is conducted using the software specifications and system specifications. A large modular system may consist of over 1,000 special-purpose software modules. Requirements in the system specification may be supported by complex relationships among many modules, and a change to a module may affect numerous functions in the system. Verification techniques may also vary according to RTCA/DO-178 criticality classification because each module has a criticality classification associated with the criticality of the requirements it supports.

Unless a sophisticated tracking system is implemented, controlling the verification technique according to criticality while accumulating a record of the verification testing of module and functional level changes can become a monumental problem. The Sperry FMC used two sets of data bases to address these problems: one for systems testing and the other for the detailed verification testing.

757/767 FMC System Verification Data Base

A direct approach to the traceability of system testing was chosen. All requirements designated for test were entered in a data base. Each requirement was then tagged with verification parameters.

A maximum of 15 verification parameters could be associated with each requirement. The parameters were divided into three classes: software, documentation, and test. These parameters provide the traceability.

The software parameters include the software component that performs the stated requirement, and the RTCA/DO-178 criticality of the requirement. The software parameters provide generalizations in many cases since the specification is not a software specification. The criticality assignment can be made from the specification when the specification designates the criticality of the requirements.

The documentation parameters include the following:

- Specification reference for requirement
- Test document reference
- Version of specification
- Change status of the requirement
- Engine airframe applicability for the test

The test parameters are entered as part of the verification test preparation. Scenarios are constructed in the system test by extracting combinations of these parameters:

- Flight phase
- System configuration
- Guidance mode
- Leg transition
- Navigation mode
- Flight-plan status
- Holding-pattern status
- Engine-out status

Implementation of the System Data Base

The data base, including the requirements and specification references, is used in developing the system test. Each requirement is accessed individually as to the applicability of the test parameters; the parameters are then entered. Applicable combinations of flight-plan lengths, cruise altitudes, cost indexes, and gross weights are selected to determine how many scenarios need to be conducted. Lists of the requirements to be tested in each flight phase are extracted from the data base and selected for test in the resulting scenarios. The test script's paragraph numbers are inserted in the data base. The scenario test script is prepared by extracting all requirements and the selected paragraph for the scenario.

Accomplishment

Using a data base accomplishes two main objectives: 1) it guarantees total requirements coverage, and 2) it assures a traceable reference between each specified requirement and where the requirement is tested. Other advantages were also realized by using this development process.

Quality of Testing

Isolating the test-case design process removes the distractions of procedure script preparation and other unrelated requirements.

Training

The tasks of designing tests, writing test scripts, and debugging tests can be delegated to verification team members without any loss in test coverage.

Bank of Reusable Tests

Economics motivates the writing of tests that can be rerun to detect regressions in system functions. It is useful to have the capability to extract tests that can detect regressions in system functions that are related to any of the test parameters.

757/767 Detailed Verification Data Bases

The problems associated with coverage and traceability in the detailed verification testing were addressed by using two data bases. The data bases were linked by the functional capability provided by groupings of software modules called Threads. The module groupings were entered in a Thread Matrix data base, while the functional capability which the threads provided was entered into the Function Matrix data base. Each data base had software and functional/documentation verification parameters associated with the entries.

The Thread Matrix software parameters included the name of each module and its functional component membership. The version of each module that had received FAA certification in a product baseline was included, along with a record of all subsequent edits to the module and versions that were subjected to verification testing.

The Thread Matrix functional verification parameters included the thread name, the RTCA/DO-178 criticality, an indication of the functional test node which the module supported in the thread, and an indicator that declared the module as either a target module whose functions would be actively tested in the thread or as an included module that would be passively tested in the thread.

The documentation parameters in the Thread Matrix included the module specification reference, a reference to the problem report which caused a change to the module, and a reference to official correspondence that initiated a module change.

The Function Matrix data base software parameters included the functional test node and its functional component membership. Functional parameters for the Function Matrix included the thread name that linked it to the supporting modules in the Thread Matrix, and the RTCA/DO-178 criticality.

Documentation parameters included the reference to the system software specification, an optional reference to the external specification, a reference to the correspondence that may have initiated a change to the thread functions, and a reference to the problem reports which resulted in changes to the thread functions. Reports could be extracted by any verification parameters in the Thread and Function Matrix data bases.

Implementation of the Thread and Function Data Bases

The data-base system was implemented to support the enhancement of an existing certified product baseline. Hence the data-base parameters for each module were initialized from data existing in configuration management files, the system software collection map, and the RTCA/DO-178 partitioning document. An analysis was done to subdivide the system into threads. As the software development progressed, all changes to each module were accumulated in the Thread Matrix data base for each module. The formal verification testing was initiated by assigning threads to members of the verification team. Thread test reports were extracted to control the verification test process. The thread test reports were spontaneous audits which summarized the change status of all target software that supported the system functions. Changes to a target module that supports multiple functions prompted verification activity for each thread whose functions were supported. The verification activity was tracked through the matrices by entering the tested requirements in the matrices for each thread, along with the version of each module tested.

Accomplishment

The use of the Thread Matrix and Function Matrix data bases made it possible to provide a total accounting of the changes and subsequent verification testing performed on all modules in the 757/767 system. Verification testing also accumulated a summary of all functional testing that was conducted in the development of a new, certified product baseline. The Thread Matrix and Function Matrix solicit a specification reference for every change to a module or function. This format directly supports the traceability requirements specified in RTCA/DO-178.

Finally, the verification approaches specified for the RTCA/DO-178 criticalities can be uniformly monitored and enforced by specifying the criticality with each Thread report. Additional benefits realized through the use of the detailed verification data bases were easy assessment of verification staff work loads, generation of a master schedule, and tracking progress to the master schedule.

Conclusions

The use of data bases in verification testing can improve the software quality by providing spontaneous audits of module and requirements coverage. In addition, the system requirements data base can be an effective tool to develop and track system testing.

Outlook

The use of automated requirements data bases can potentially greatly reduce software development costs while significantly improving the quality of the software and the associated documentation. Linking word-processor documentation systems, automated problem-reporting systems, and configuration-management systems with automated requirements data bases can substantially reduce the overhead in a verification testing program. The quality of the software and documentation can be significantly improved when automated methods prompt software engineers to update documentation and to anticipate problems in related software.

About the Author

GLENN LUEDERS is a Principal Engineer with Sperry Corporation, Flight Systems, in Phoenix, Arizona, where he is employed in the Commercial Flight Controls Department. He joined Sperry Univac in 1976, and was involved in testing air traffic control systems. He has been involved with flight management systems since 1980, testing and certifying aircraft control systems.

Glenn earned a B.S. degree in math and physics (1969) and an M.A. degree in physics (1971) at Mankato State University, Mankato, Minnesota, with graduate work toward a Ph.D. at Virginia Polytechnic Institute, Blacksburg, Virginia, 1971.

He has also been certified as a high school teacher by Mankato State University.

AUTOMATED BENCHMARK GENERATION BASED UPON A SPECIFICATION LANGUAGE

N. Rajan* and S.E. Feteih†
Department of Aeronautics and Astronautics
Stanford University, Stanford, CA 94305

J. Saito‡
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

The problem of validating and verifying digital flight control system (DFCS) software is addressed in this paper. A new specification language [DIVERS] is proposed, and is the keystone in our approach. This language consists of keywords where each keyword represents an element in the block diagram of a DFCS. DIVERS has a dictionary which contains all the keywords a DFCS designer might need. Translator programs convert the system specifications into an executable, high-level language program. The features of translators are discussed and are elucidated by examples. This language is used to describe a typical flight software module.

Introduction

An obvious heuristic approach to validating software systems is to compare the output of the implemented system to its specified output. The input to the system is varied over as much of the specified input domain as is practicable, and the output is checked against the specifications. This is often the only recourse when verifying Digital Flight Control System (DFCS) software which contains a varied mix of logic, arithmetic, and input-output operations. A completely independent simulation of the DFCS may be needed when checking software against the specifications. The problem of specifying the output of the DFCS is approached by developing a special purpose high-level language, which makes it easy for the control system designer, with eventual verification in mind, to describe the input-output behavior of the DFCS at the design stage. The language is designed to allow for machine translation of the specifications into an executable program. The specification language is meant to be simple, yet general enough, to describe most modern digital autopilots. For this reason it is not intended as a language for actual written DFCS code. The latter is implementation specific and would complicate the language.

Examples of specification languages are abundantly available in the literature (1, 2). Reference 1 describes a specification language called ESPRESO which is tailored toward a formal documentation setup of the specifications for process control applications. The language described in Ref. 2, called SLAN-4, was designed as a formal language for writing specifications, and was meant to be used during the development of software systems for design, communication, and documentation purposes.

The motivation for the development of a specification language for DFCS is essentially as a tool for verification. In earlier work (3) a technique was developed for testing DFCS software on a module-by-module basis. This required the comparison of the module outputs with those of its model, or benchmark, for the same input values. Obviously, this technique presupposes that a model or benchmark of the flight software module can be written in a simple and reliable way. Most DFCS consist of relatively few building blocks used repeatedly and interconnected in various ways. This feature can be used to design a special language that describes DFCS in terms that are familiar to control system designers. This language should encourage systems designers to take note of the performance specifications for the DFCS which can later be used to check the implemented system. The features of the proposed language DIVERS (digital flight control software verification specification language) are discussed in the following section.

Design Objectives

The specification language was designed with the following goals in mind.

1) The control system designer should find the language simple and natural to use. Thus its constructs should embody terms with which he is familiar. Control engineers tend to describe systems in block-diagram form using transfer functions or algebraic equations for each component. Systems usually have several parallel paths. The specification language description should correspond one-to-one with the block diagram description and should be recognizable to any control engineer who does not have special programming knowledge.

2) The language should encourage the system designer to document his physical insight into system performance. The system designer usually has a strong intuitive insight into the behavior of the control system. For example, the angle of attack for most transport aircraft will not exceed 6° in the positive direction and approximately half that in the negative. The designer can note this fact in the system specification. At the system verification stage, this fact can be machine-extracted and used to set up an executable assertion (4). This is a nontrivial example; there are actual DFCS implementations where the angle of attack has been scaled to 360°. The syntax rules imposed on the designer must be kept to an absolute minimum which results in a heavier burden being placed on the translator. The information which cannot be framed according to the syntax of DIVERS should still be included as comments. Thus, the objective of any translator of the language will be to extract as much information as is possible from any specification set.

3) The language should be easy to modify and to expand. As flight control system components become

*Research Associate. Member AIAA.
†Research Assistant. Student member AIAA.
‡Mathematician.

more sophisticated, the language in which they are described also needs to be updated. The upgrading of translators should be easily accomplished, without rendering earlier versions obsolete.

The raison d'etre for this specification language is the verification of DFCS software. As a result, the design of the language must be based upon a survey of the features of currently available DFCS systems, and those which will be available in the future. As described in this paper, the language is focused on DFCS which are a digital implementation of essentially analog autopilots. Its features and rules are informally described.

Language Structure

Upon examining a typical block diagram of a control system, Fig. 1, it is seen that in order to describe it in words, a name has to be given to each block and then the inputs, outputs, and parameter values have to be mentioned. The number of different names needed are limited because DFCS systems are composed of a relatively small number of functional block types, which appear at different places with different parameter values. The names of the blocks are the keywords of the language, and a list of all the keywords is maintained in a dictionary. The inputs and outputs are the variables in the language. The parameters are constant values, which represent the gain of a filter or its time constant, or they are Boolean conditions which determine switching. There is no ambiguity in the above description because the keyword determines what the parameter should represent. Specifically, if the keyword is a switch, then the parameter will be a string specifying a Boolean condition.

The syntax for a complete specification of one component of a block diagram is as follows:

keyword[input token;parameter token:output token]

where the input token (a token by definition is a series of ASCII characters) consists of the names of the input variables that are separated by commas, that is,

input token := input variable name 1,...,
input variable name n .

The same convention is followed for the parameter and output tokens. The input and output variables are "declared" at the beginning of the specification text. A declaration is a token of the following form:

variable name:physical unit, lower limit,
upper limit, fullscale value, type, size;

Here the upper and lower limits are the bounds on the variable that are expected in practice; the full-scale value is the maximum value to which the variable is scaled. The type indicates whether the variable is real, integer, or Boolean; size specifies the number of bytes needed to store the variable. The full-scale value is an absolute value and it is usually greater than the maximum value. Variable type is a token of the form

token1 token2

where token1 may be "testvar" or null, and where token2 is "real," "integer," or "Boolean." Example variable types are

testvar real

testvar Boolean

integer

The "testvar" token indicates that the variable referred to is a variable whose values will need to be read or modified during verification of the DFCS. It is an indication to the programmer who is responsible for coding the DFCS that the specified variable should explicitly appear in the DFCS code under the same name. This is a restriction imposed so that benchmark generation may be automated. The declaration of each variable includes its physical unit, bounds, and full-scale value to permit consistency checks of the specifications, and to permit checks of the scaling for accuracy and precision.

Two other constructs are permitted in the language. These are comments which take the following form:

/*this is a comment*/

and assertions which are statements about the normal, expected behavior of variables appear in the form:

/$assertion statement$/

For example, suppose the variable ALPHA is expected to remain within the bounds -0.5 to 0.5 at the output of a given block; the specification of the block can be followed by the assertion

/$-0.5 < ALPHA < 0.5 $/

Clearly, the language proposed here is very simple. This implies that much of the burden of generating benchmarks from the above specifications falls upon the language translator program. We will now describe the functional features of translator programs.

Features of Translator Programs

A translator will have a database which has two elements, a dictionary of the keywords that are currently available, and a macro library for each keyword. The keywords in the dictionary are arranged alphabetically. The macro library contains one macro per keyword which is also arranged alphabetically. Each macro is structured as follows:

keyword[token1;token2:token3]

macro definition

where the three tokens represent the input, parameter, and output strings, respectively. Each string consists of dummy variable names that are separated by commas. An alternate form that is allowed for specifying input and parameter strings is:

(variablename,n)

Here the n is an integer which equals the maximum number of names that are allowed in the input/parameter string. The idea behind this provision

is to allow the number of input variables and parameters to vary from 1 to n. Since we know that an input string consists of variable names that are separated by commas and that are terminated by a semicolon, the number of variables in the string is equal to the number of commas plus one. This feature proves useful, for example, where the output is the sum of several inputs whose number may vary.

The macro definition consists of code that is written in the object/output language of the translator. This code, written in a high-level language, C, in our case, describes the action of the keyword in terms of the dummy input, parameter, and output variable names. If the dummy variables in the macro definition are replaced by actual variables, the resulting code should simulate the keyword. Consider, for example, a summer with the number of inputs varying from one to ten, that is, a block with the following characteristic:

$$o = \sum_{i=1}^{n} c_i \, e_i$$

where the c_i's are constants, and the e_i's are the inputs. The macro definition for this block is as follows:

```
summer[(alpha,10);(c,10):o]

    o = 0;

    $[o = o + c * alpha;]
```

The dollar sign is a unitary operator whose action is to call for a repetition of the token within the square brackets. The repetition count is obtained by scanning the argument list in the keyword for the number of input variables, alpha, and for the number of parameters, c, and by taking the lower number as the count. Thus a call such as

```
summer[alpha,beta,gamma;10,1.5:sum]
```

would result in a repetition count of 2. The expansion of the keyword will be

```
sum = 0;

sum = sum + 10 * alpha;

sum = sum + 1.5 * beta;
```

From this example, it is also seen that the keyword macro is expanded by replacing the dummy variables by the arguments in the call. Also the variable "gamma" is not used in the macro expansion. The translator will report an error.

Since the output of the translator is a complete, executable C program, the translator must be able to expand the keyword macros into C statements in proper order , declare variables, and include input and output statements for some of these variables at appropriate points in the program. Only the variables that are declared "testvar" will appear in these input-output statements. A "testvar" variable that is an input to a keyword will appear in an input statement that precedes the macro expansion for that keyword in the translator's output program. A variable that is the output of a keyword will appear in an output statement that follows the keyword's macro expansion. A variable that is both an input of one keyword and the output of another, will appear in both of the above places. Since each keyword has both input and output variables, keywords can be written down in any order. The translator arranges the keywords into a proper sequence of keywords so that all inputs to each keyword are available at the point in the program where the keyword is translated. To describe the algorithm for ordering keywords, we introduce the following notation for keywords:

$$K_i[e_{i,j}, j = 1,\ldots,\ell_i;\ p_{i,k}, k = 1,\ldots m : o_i]$$

where

K_i is the ith keyword, $i = 1,\ldots,N$

N is the total number of keywords

$e_{i,j}$ is the jth input signal to K_i

$p_{i,k}$ is the kth parameter of K_i

o_i is the output signal of K_i

ℓ_i is the total number of inputs

The algorithm for ordering the keywords into groups is as follows:

Keyword ordering algorithm

Step 1: Group 1, contains those keywords K_i, whose input signals $e_{i,j}$, $j = 1,\ldots\ell_i$ are all testvar.

Step 2: Suppose that groups $1,\ldots,r$ have already been determined. A keyword K_i, which has not yet been ordered belongs to the (r+1)th group if each of its input signals $e_{i,j}$, $j = 1,\ldots,\ell_i$ satisfies one of the following three conditions (mutually exclusive):

a) $e_{i,j}$ is testvar and, if $e_{i,j} = o_a$, the corresponding keyword K_a has already been ordered.

b) $e_{i,j} = o_a$; o_a is the output of keyword K_a that belongs to one of the groups already ordered, that is, groups 1 through r.

c) $e_{i,j} = o_h$; o_h is the output of any history keyword K_h.

Step 3: Repeat step 2 until all the keywords $K_i, i = 1,\ldots,M$ that satisfies conditions a, b, c of 2 have been ordered.

Step 4: If $M < N$ this indicates that some keywords remain unordered, print an error message.

In condition c, the history keyword is just a one-unit delay. This facility is necessary to handle situations where the outputs of filters are fed back as the inputs of other keywords, as for example, in Fig. 2a. The outputs of the low- and high-pass filters are fed back to the summers, through keywords that are labeled "history" in Fig. 2b. The output a3.p and the input a3 of this keyword are related by the difference equation:

$$a3.p(k) = a3(k - 1), \quad k > 1,$$
$$a3(k) = 0, \quad k = 0.$$

Once the keywords are properly ordered, they are expanded in the second pass in terms of the actual input and output variable names. The necessary declarations and initialization statements for the variables are included. Data files that contain input values for the test variables are created based upon the types of these variables and the test pattern desired by the designer. In the next section, the use of the language is illustrated through examples.

Examples

The digital autopilot described in Ref. 3 was used as the basis for the development of DIVERS. A dictionary of keywords that is adequate for describing this autopilot is given below.

Dictionary. ARCTAN
ARCSINE
ARCOS
COMPARATOR
GAIN
HPF
LPF
LIMITER
SUMMER
SWITCH
HISTORY

Two of the keywords, a low-pass filter "LPF" and a limiter "LIMITER" are described in detail below.

A block diagram and the difference equation representation of the LPF are shown in Fig. 3. The latter is based upon an Euler integration of the filter differential equation. The macro for the LPF is:

```
LPF  [ input    ;
sampling time, time constant :
output        ]
```

This keyword is implemented as a C subroutine. Thus, every time the keyword LPF is seen in the specification of a flight software module, the translator replaces it by a call to the subroutine LPF with the appropriate arguments. The subroutine is provided in a library so that it can be linked with the benchmark program. The block diagram for the limiter is shown in Fig. 4. The macro definition for the limiter is:

```
LIMITER [input; upper limit, lower limit: output]
```

The macro expansion for this keyword is:

```
if (input > upper limit)
  output = upper limit;
else if (input < lower limit)
  output = lower limit;
  else  output = input;
```

Figure 1 shows the block diagram of part of a typical flight software module, called "speed-comp." This module performs the computations that are necessary for angle of attack determination as a function of take-off and go-around conditions, as well as defining angle of attack (AOA) limits. It also performs autothrottle calculation, i.e., it computes the acceleration that is necessary to attain a given speed when the aircraft dynamics, the existing acceleration, the airspeed, and the existing constraints that are defined by a prespecified flight envelope are taken into consideration. The inputs to the module are simply:

* The flaps angular position, "THETA f ".
* The aircraft angle of attack, "ALPHA ".
* The longitudinal acceleration, "Ax ".
* The calibrated airspeed, "CAS ".

Its output consists of:

* A reference angle of attack for further computations.
* Takeoff, go-around reference flap angle.
* Forward longitudinal acceleration.

The only representative parts of this module which will be demonstrated are those which have longitudinal acceleration, and calculated airspeed as an input, and which have longitudinal acceleration as an output. The specification of the module in DIVERS is shown below:

```
Variable declarations :
cas.ms:knots , 500, -500, 512,
                        testvar real,4;
cas.ms.p: knots, 500, -500, 512,
                        testvar real,4
long.acc.ob:ft/sec**2,120,44,3000,
                        testvar real,4;
a9 :ft/sec**2, 60 , 32 ,3000 , real,4 ;
a4 : knots,498,-498,512,real,4 ;
a5 :knots,18.7673,-18.7673,real,4 ;
z5.d:ft/sec**2, 250, 0 ,3000 ,real ,4 ;
a6  :ft/sec**2,64.4,-64.4,3000,real,4 ;
o.cas.ms:knots,500,-500,512,real,4 ;
o.cas.ms.p:knots,500,-500,512,real,4 ;
a7   :knots,1000,-1000,1024,real,6 ;
a8   :knots,56.33,-56.33,512,real,4 ;
```

```
z4.d:ft/sec**2, 89, 0.0,3000 ;real,4 ;

u.dot.comp:ft/sec**2,1694,0.0,3000,
                        testvar real,4;

u.dot.avrg:ft/sec**2,1694,0,3000,
                        testvar real,4;

y5.d :ft/sec**2,250,0,3000,real,4;

y6.d :ft/sec**2,250,0,3000,real ,4 ;

y4.d:ft/sec**2,89 ,0.0 ,3000,real ,4 ;

y7.d:ft/sec**2,339,0.0 ,3000,real ,4 ;

z6.d :ft/sec**2,250 ,0.0 ,3000,real ,4 ;

z7.d :ft/sec**2,339 ,0.0 ,3000,real ,4 ;

flag.acc.dif:volts ,1, 0; 1,boolean ,1 ;

      LIMITER [ cas.ms  ;
                  500.0 ,
                 -500.0 :
                    a4       ]
  /$   a4 <= 488.3   $/
      GAIN    [ a4   ;
                0.336 :
                   a5       ]
      SUMMER [ a5 ,  z4.d ; :
                              z5.d    ]
   /$  z5.d <= 250    $/
/*long.acc.ob obtained from Ax via ADC*/
      GAIN [ long.acc.ob ;
             0.5          :
             a9           ]
   /$   a9  <= 120  $/
      SUMMER [ a9  , a3 ; :
                                a6    ]
/*This is a two way switch, it switches*/
/*on when the other computer  is working*/
      SWITCH [ a6 , z4.d ; o.box.on :
                  y4.d    ]
  /*when o.box.on true,y4.d = a6*/
      GAIN [ cas.ms  ;
             0.169     :
             cas.ms.p   ]
 /$    cas.ms  <= 512       $/
  GAIN [ o.cas.ms;
         0.169     :
         o.cas.ms.p ]
  SUMMER [ cas.ms.p , o.cas.ms.p ; :
                                a7   ]
  LIMITER [ a7       :
          159 ,
         -159  :
           a8                    ]
   /$ a8 <= 89.56  $/
  SUMMER [ a8 ,y4.d  ; :
              y5.d.p              ]
  SWITCH [ y5.d , z5.d ;  o.box.on  :
                        y5.d        ]
  LPF [ y5.d ;
        0.1 ,
        5.0  :
        y6.d  ]
  SUMMER [ y6.d  , y4.d ;  :
                          y7.d     ]
  HPF [ y7.d ;
        0.1 ,
        5.0 :
        u.dot.avrg  ]
  LPF [ z5.d;
        0.1 ,
        5.0 :
```

```
        z6.d    ]
SUMMER [ z6.d , z4.d ; :
                        z7.d    ]
HPF [ z7.d ;
      0.1 ,
      5.0  :
      u.dot.comp   ]
COMP [ u.dot.cmd , u.dot.comp ;
                    0.926   :
                flag.acc.dif    ]
```

When Fig. 1 is examined, it is seen that the specification is written by simply replacing the blocks with their keywords and their input-output signals. These signals are "declared" at the top of the specification. Assertions have also been introduced at various points that use the '/$' facility; for example, the variable 'a4' is declared to be less than 488.3.

Conclusions

In this paper, the problem of automatically generating simplified models or benchmarks of DFCS modules has been addressed. A specification language has been proposed which makes it easy for the system designer to describe flight software. This language specifies the DFCS in a form that is machine translatable. It also embodies constructs that enable the designer to express facts about system performance that can later be employed to set up executable assertions for software verification. The language can be readily updated to express new features of DFCS by the addition of keywords. The salient features that the language translators should have are described and the language is applied to a typical DFCS module.

References

1) Ludewig, Jochen, "ESPRESO: A System for Process Control Software Specification," IEEE Trans. on Software Engg., Vol. SE-9, July 1983, pp. 427-436.

2) Beichter, Friedrich W., Herzog, Otthein, Petzsch, Heiko, "SLAN 4, Software Specification and Design Language," IEEE Trans. on Software Engg., Vol. SE-10, March 1984, pp. 155-161.

3) Rajan, N., de Feo, P. V., Saito, J., "Stress Testing of Digital Flight Control System Software," IEEE/AIAA 5th Digital Avionics Systems Conference, Seattle, Wash., Nov. 1983.

4) Andrews, Dorothy M. and Benson, Jeoffrey P., "An Automated Program Testing Methodology and Its Implementation." Proceedings, 5th International Conference on Software Engineering, 1981, pp. 254-261. Also reprinted in "Tutorial: Software Testing and Validation Techniques," 2nd edition, 1981, pp. 349-356.

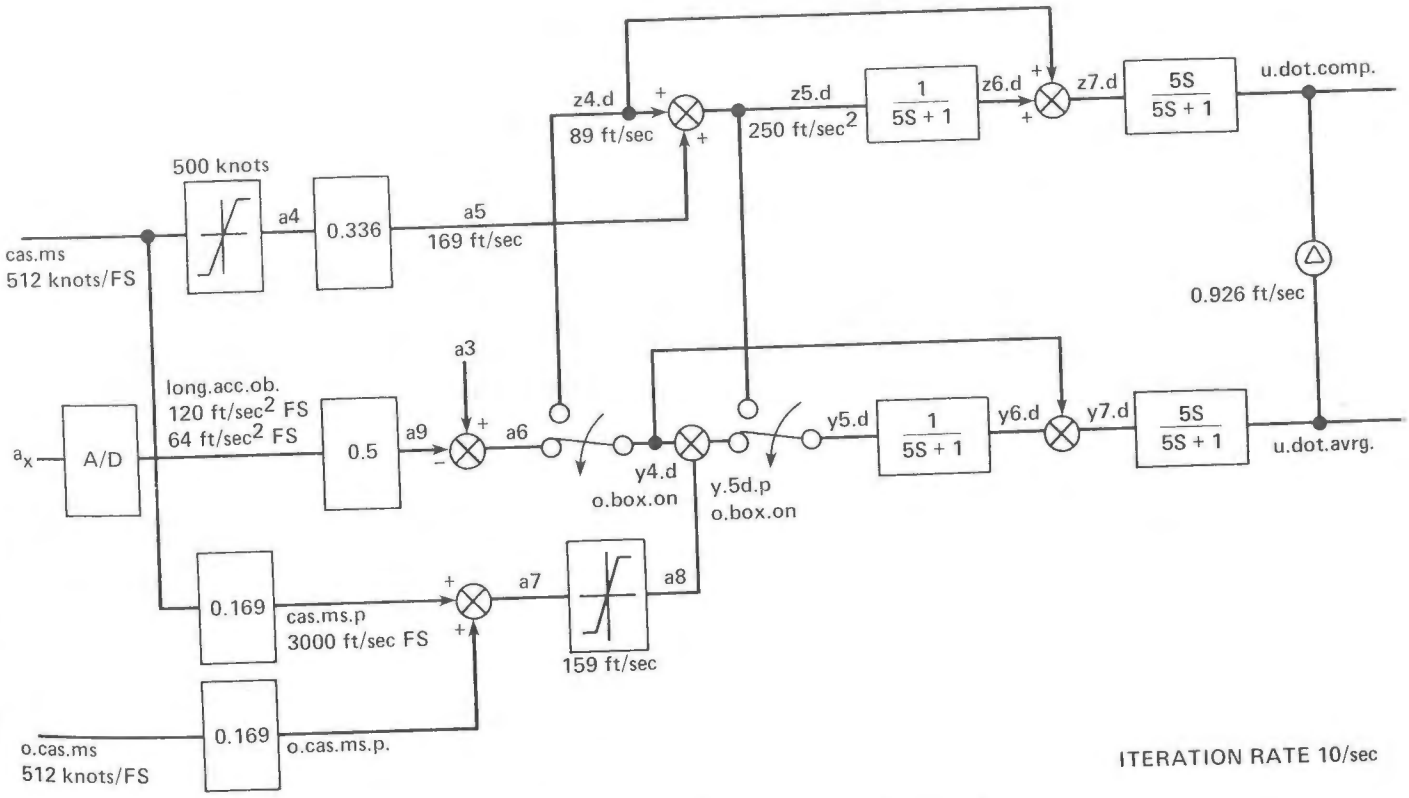
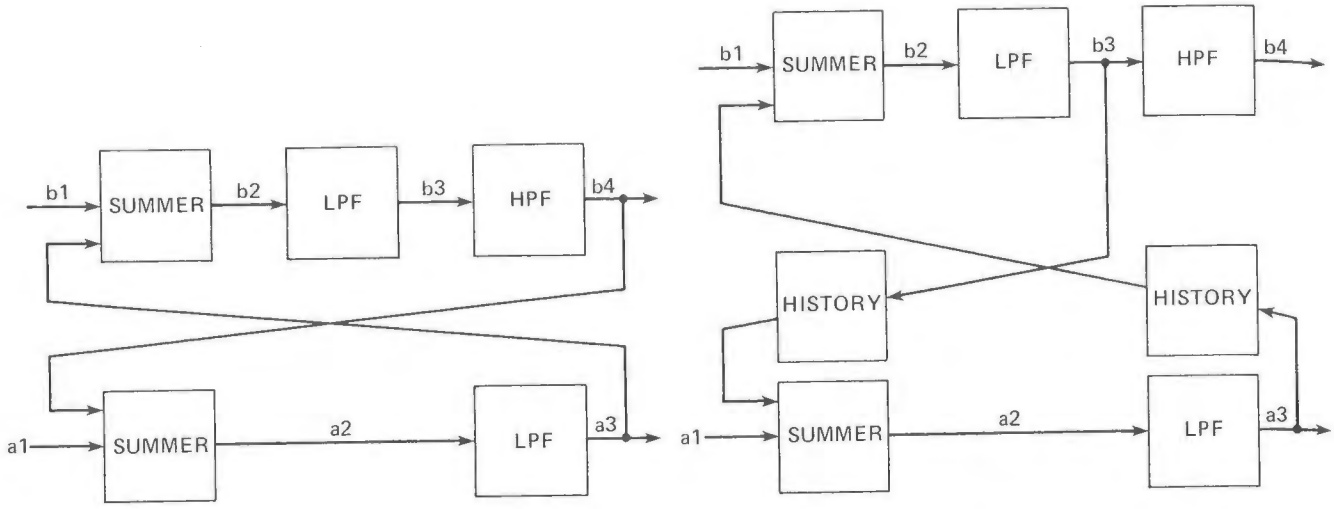


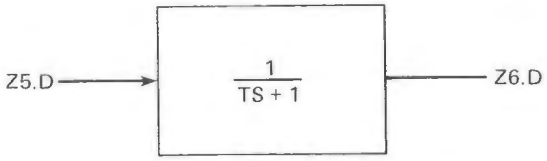
Fig. 1 Block diagram for the speed-comp module.



a) Without history keywords.

b) With history keywords introduced.

Fig. 2 Cross connection of keywords.



LOW PASS FILTER REPRESENTATION

DIFFERENCE EQN.:

$$\text{PRESENT OUTPUT} = \frac{\text{TIME-CONSTANT} - \text{SAMPLING TIME}/2}{\text{TIME-CONSTANT} + \text{SAMPLING TIME}/2} * \text{PAST OUTPUT}$$

$$+ \frac{\text{SAMPLING TIME}/2}{\text{TIME-CONSTANT} + \text{SAMPLING TIME}/2} * (\text{PRESENT INPUT} + \text{OLD INPUT})$$

Fig. 3 Representation of a low-pass filter.

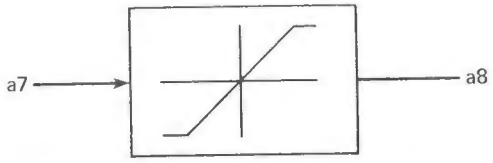


Fig. 4 Representation of a limiter.

AUTOMATED SOFTWARE TEST SYSTEM FOR THE 737-300 FLIGHT MANAGEMENT COMPUTER

Steven C. Runo*

Senior Engineer
The Boeing Commercial Airplane Company
Seattle, Washington

Abstract

Traditional manual approaches to software validation could not produce the required testing rigor within the inflexible 737-300 Flight Management Computer System program schedule. In response, an integrated and highly automated validation test system was developed based on experience gained from previous validation efforts. The "user-based" system consists of three basic elements: 1) a specialized simulation for generating expected results, 2) an automated test bench that interrogates the operational flight program "in situ", and 3) a program to document and compare test results to expected results. The entire procedure is automated from test case design through final analysis of the test results. The system has proved to be an efficient and rigorous validation of the flight software within a tight time schedule and limited budget. Further, the user is released from tedious laboratory testing and allowed to concentrate on test analysis.

Introduction

In recent years, automated software testing methods have been suggested as a means to provide rigorous verification and validation of operational flight programs without the drain on time and manpower that manual techniques normally require. Suggested methods include: specialized simulations, theoretical "best" approaches to test design, result comparators and automated test benches. However, in many cases these automated methods reduce the effort required in a specific area only to increase the overall testing effort by expanding the number of test conditions measured or the required degree of analysis.

The time and budget constraints of the 737-300 Flight Management Computer System (FMCS) project would not permit manual testing methods or inefficient automated methods. For example, approximately one third of the validation program for the 737-300 Flight Management Computer was testing of the performance functions. This was initially expected to require at least 10,000 separate test points and hundreds of laboratory test hours for each new software release. As a result, validation of the performance functions was considered "risky" and it was determined that the required testing could only be accomplished with an integrated, automated system for preparing expected results, conducting the laboratory testing and finally, analyzing the results.

The system that was developed is largely the result of several years of experience in similar testing efforts. Appropriately therefore, this paper begins with a discussion of the previous Boeing Co. validation programs that significantly shaped the 737-300 Flight Management Computer System validation effort. Next is a detailed discussion of the specific elements of the performance function validation testing, including: the plan of test, the simulation software for generating expected results, the automated laboratory test system and the methods used to compare and archive the test results. The paper concludes with a brief description of the experience-to-date utilizing this approach.

History

Performance Data Computer System (PDCS). The Performance Data Computer was originally developed in the mid-seventies in response to rapidly increasing fuel costs. The system was designed by Boeing and the hardware / software vendor, Lear Siegler, Inc. of Grand Rapids, Michigan, to optimize the performance of 727 and 737 aircraft via a mixture of stored and computed speed schedules and throttle setting targets based on current flight conditions.

Validation testing of the PDCS was one of the first efforts of its type and thus has played a significant role in shaping the development of later validation efforts. The earliest test cases were designed to test each of the functions of the PDCS at conditions that were likely to be encountered in normal

*Member AIAA

operations. All early testing of the PDCS was accomplished via manual entries through the Cockpit Display Unit (CDU) keypad or through test bench discrete switches and variable potentiometers. Results were manually recorded from the CDU responses of the PDCS. This test approach was highly time-consuming and thus, necessarily limited the scope and detail of practical testing to about 1000 total test points.

Late in the PDCS development program an automated test bench was developed. The automated test system would enter Cockpit Display Unit and aircraft system inputs, wait an appropriate length of time, read the CDU responses and compare the test results to pre-stored expected results. Because this system was somewhat inflexible and required a great deal of pre-test effort to convert the manual test cases to the automated format, it was primarily used to test only the uniformly-formatted propulsion test cases. However, the system did provide valuable experience for development of the 737-300 test system.

Performance Navigation Computer System (PNCS). The Performance Navigation Computer System was developed as one of the earliest "flight management"-type systems. The original intent was to integrate PDCS performance information with navigation and guidance capability in a single computer for 737-200 aircraft. Limited sales interest and technical problems forced cancellation of PNCS development prior to the anticipated system certification. Again, the accumulated experience would prove to be valuable.

This system introduced the complexity of navigation and guidance computations overlayed on the basic performance information. Using a "flight plan buffer dump" developed by Lear Siegler for the PNCS program, much of the performance information could be captured at each waypoint in the predicted flight plan. The data could then be analyzed to determine if aircraft performance was being computed correctly. However, if an error was found, it was often difficult to trace it to its origin because the flight plan buffer dump could not include all of the performance variables and their intermediate values. The PNCS performance plan of test did not actually increase the total number of test points acquired, partly because of confidence in the previous Performance Data Computer aircraft and engine models and partly because the higher-order functions of the PNCS required significantly greater test and analysis time. The experience of the PNCS test program suggested an entirely new approach to validation testing of flight computer software was required.

757/767 Flight Management System (FMS). When the new generation airliner programs were launched in the late seventies, it was recognized that testing of the 757 and 767 would be the most rigorous ever attempted. This especially applied to the new "glass" cockpits and flight management systems. In response, major projects were undertaken to develop automated testing techniques for validating the performance functions of the 757/767 Flight Management System. The first major project was the creation of a series of programs to generate flight management system expected results. These series of programs became the Boeing Standard Programs (BSP) and are detailed below. Other test tool projects included the development of an automated test bench and a test report comparator program. The latter is also detailed in the discussion below.

The Performance Algorithm Test System (PATS) was designed to set test conditions and extract results from the operational flight program (OFP). The OFP source code is first prepared by adding input and output routines to translate between the unique FMS software structure and the PATS driver. This modified code is then loaded and executed in the Flight Management Computer hardware according to commands given in the PATS driver file. Test results are recorded in a file formatted identically to the expected results generated by the Boeing Standard Programs simulation software. The 757/767 work was a major influence on the development of the 737-300 Flight Management Computer performance function test program and many of the features of the system described below were originally developed for the 757/767 FMS test program.

737-300 Validation Testing

Development of the 737-300 Flight Management Computer performance function test program began with the design of an overall plan of test and the development of requirements for the individual elements of the test system. The three major elements required under the plan of test were: 1) the Boeing Standard Programs (BSP) expected results generator, 2) the Performance Validation Test System (PVTS) automated test bench and 3) the COMPEX test results comparator / report generator. The BSP simulation software would require major revisions to reflect unique 737-300 Flight Management Computer design requirements and the 737-300 airframe / engine combination. The Performance Validation Test System would have to be developed from the "ground up" to interact with the new flight computer hardware. However, the COMPEX report generator

developed as part of the 757/767 BSP project would only require user experience.

Performance Plan of Test. The performance plan of test was developed given two major considerations: 1) the experience gained in the PDCS, PNCS and 757/767 FMS test programs, and 2) the timing of required software deliveries from Lear Siegler. The plan of test calls for a "bottom-up" approach starting with module-level validation of the aerodynamic, propulsion and atmospheric data bases. The "flight phase" testing level is designed to validate the integration of data base information. The final "mission"-level testing is directed at validating the the complete performance function system. Each subsequent level of testing is dependent on the successful validation of the lower levels of testing. This scheme not only simplifies pinpointing errors, but also blends well with 737-300 program schedules that called for the delivery of increasingly capable software in distinct packages. The plan of test is depicted graphically in Figure 1.

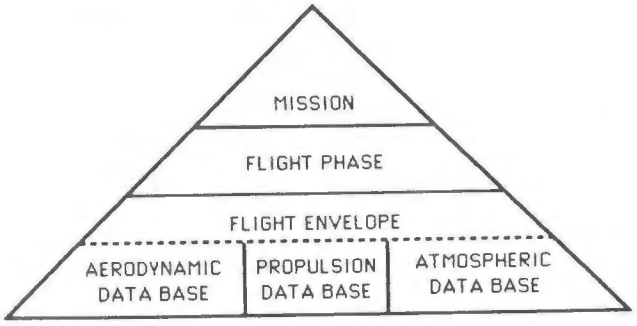


Figure 1 Performance Plan of Test

The data base level is designed to check the individual software modules by varying each of the input parameters over, and outside, the full range of possible values, with particular attention to regions where errors are likely to occur. Testing tolerances are limited to computer "round-off" errors since the polynomial nature of the data bases eliminates inexact interpolated or approximate extractions. This level also verifies the correct computation of the complete flight envelope defined by the combination of aerodynamic, propulsion and atmospheric data bases.

The flight phase level of testing examines the computation of each separate mode of flight, e.g., maximum gradient climb, long range cruise, economy descent). Intermediate results are compared at each performance integration step, rather than simply at each navigation-defined waypoint. Further, the values of intermediate parameters, such as thrust and drag, are checked against expected results derived from the Boeing Standard Programs. Tolerances at this level are a combination of the database and FMCS requirements-specified tolerances.

The mission level of performance function testing is designed to validate the complete integration of a flight plan or "mission". This "top-level" testing also includes examination of performance function interactions with the other Flight Management System components. However, the scope of these tests are limited by separately defined "flight scenario" tests that are designed to check the complete range of Flight Management System component interactions. The intermediate results examined in the flight phase tests are also available on this level, but testing tolerances are now based only on tolerances specified in FMCS requirements.

Boeing Standard Programs. The Boeing Standard Programs (BSPs) are a set of mainframe computer programs originally developed to validate the performance management functions of the 757/767 Flight Management Computer (FMC) system. The BSPs consist of nine separate, but inter-related programs that generate expected results in a format permitting automated comparsion with test results. Previous Boeing performance programs were generalized to support a wide variety of uses and not specifically tailored to the requirements of flight management software validation. The programs emulate the nine functions shown in Table 1.

PROGRAM	DESCRIPTION
A523 AEROEX	Aerodynamic data base
A524 PROPEX	Propulsion data base
A525 ATMOSX	Atmosphere and wind prediction
A526 FNLIMX	Speed envelope calculation
A527 ALTEX	Altitude limits calculation
A528 SPEEDX	Speed generation
A529 LEGEX	Leg integration calculation
A530 STEPEX	Step-cruise optimization
A531 BSPEX	Full flight path trajectory

Table 1 Boeing Standard Programs

The BSPs are designed to share common modules with each other. This is best illustrated in Figure 2. The solid-lined boxes represent a related set of routines; the solid-lined boxes inside the larger dash-lined box (e.g. speed generators, leg integrators, etc.) are termed "functional" modules and represent sets of routines used by more than one BSP. The solid-lined boxes outside the dash-lined

box are termed "driver" modules and represent routines which are used by only one BSP (e.g. SPEEDX, LEGEX, etc.). Arrows point from calling routines to the routines called.

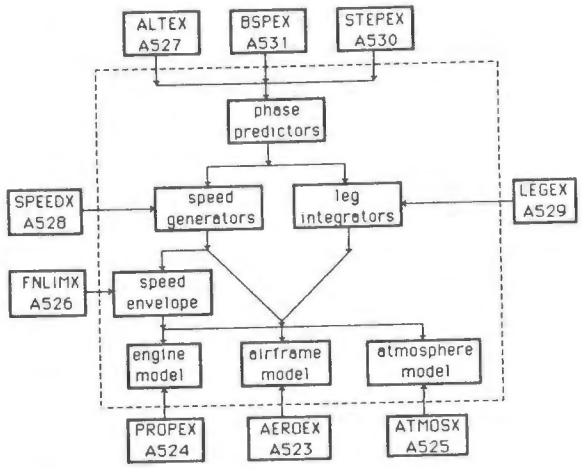


Figure 2 BSP System Architecture

Execution of the BSPs requires files containing the necessary aerodynamic and propulsion coefficients, and a performance input file that selects the particular test conditions and functions to be executed. As flight testing of the 737-300 updated the available performance data, this new information could be periodically added to the propulsion and aerodynamic data bases by editing the coefficient files. Updated expected results could then be generated without redesigning the test cases or recoding the BSP routines.

For the 737-300 FMCS performance validation effort, the BSPs required a new engine model, an enhanced aerodynamic model, polynomial speed generators and additional logic in the phase predictors, and leg integrators for the specific requirements of the 737-300 FMCS. Since each version of the BSPs is based on a single engine model, a new BSP version was created for the 737-300 FMCS program. However, the other changes were added to the existing BSPs as selectable options available to all versions of the BSPs. For example, all versions of the BSPs currently allow the user to select speed generation based on 757/767 table look-up methods, 737-300 polynomial equations or generalized computational methods.

Despite these relatively major changes, the BSPs were ready for use in a short period of time because of the extensive 757/767 development work that had already been completed and validated. In addition, a generalized version of the BSPs was also available that premitted preliminary generation of expected results and even development of the performance data base equations.

Performance Validation Test System. The Performance Validation Test System (PVTS) is an automated test bench designed to accept Boeing Standard Program format inputs, execute the corresponding software modules in the Flight Management Computer, and record the results in a format suitable for automated comparsion with BSP-derived expected results. The system is unique in several respects. First, the vendor-supplied operational flight program (OFP) is interrogated without modification while it is resident in the vendor-delivered hardware. This is significant because it ensures testing of the flight software in nearly actual operational form. Second, PVTS utilizes the same inputs to drive the execution of the OFP that were used to execute the BSP simulation software. This allows the test engineer to rapidly redesign a given test case to meet changing requirements and obtain both the test results and expected results without having to formulate a separate, and possibly different, test condition file. Finally, PVTS allows testing of a single function or a complete series of functions without user interface except at startup. This feature relieves the tester from monotonous and error-prone manual testing, while producing repeatable results much faster than manually possible.

The PVTS hardware consists of a VAX 11/780 computer connected to a Lear Siegler, Inc.-supplied Computer Control Unit (CCU) via an eight-bit parallel bus. The Lear Siegler-developed interface system allows the VAX to set and examine variables internal to the Flight Management Computer and execute the operational flight program (OFP) from breakpoint to breakpoint. Lear Siegler "symbolic debug" software looks up the memory addresses of variables allowing the variables to be accessed by name. The system is shown in Figure 3.

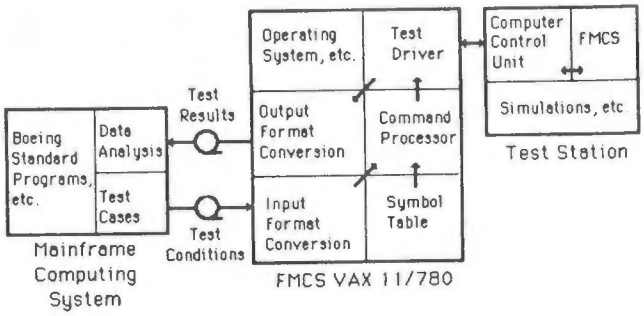


Figure 3 PVTS Overall System Diagram