DISTRIBUTED PROCESSING FOR COMPLEX SIMULATORS

By David Parkinson
Singer Link-Miles Ltd.
Lancing, Sussex, England

Summary

The requirements for training systems for increasingly sophisticated aircraft continue to extend the technology of computing systems applied to real-time simulators. This paper describes a distributed computing system, partitioned on a functional basis with the autonomous major functions connected as a network. This system, called Functionally Distributed Simulation, is the basis for a new simulator generation, the Micro Simulation Technology (MST) simulator. It permits designing and manufacturing a simulator on a modular basis, which achieves considerable advantages in life-cycle costs, especially those associated with maintenance and upgrades. Microprocessors are used throughout, to achieve even greater flexibility and reliability. Extensive use is made of parallel processing to meet the high iteration rates needed for simulators; the various ways of interconnecting the microprocessors to meet the performance requirements are described. Microprocessors are also used for dedicated tasks such as real-time status monitoring and built-in test features, especially within the integral distributed input/output system. The performance of this system is described, together with results from simulators that use this modular approach.

Introduction

Background

Training systems for sophisticated aircraft continue to extend the technology requirements of real-time computing systems. To achieve the high fidelity demanded, today's simulators use some of the most advanced super minicomputers available. While the cost of a given computing capacity tends to fall because of technological developments, increases in complexity, programming, and maintenance factors continue to drive up the cost of ownership. This paper describes a new concept that addresses this problem by applying distributed microprocessor concepts. Such a system forms the computational basis for a new simulator generation, the Micro Simulation Technology (MST) simulators.

Traditional Simulator Complexes

A traditional simulator configuration consists of a number of subsystems driven by a centralized host computer, which is connected to the subsystems by means of various interfaces (Figure 1).

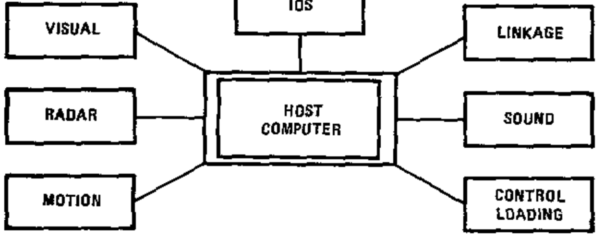

Figure 1 TRADITIONAL SIMULATOR CONFIGURATION

Where extremely high performance is required, such as for radar or visual simulation, the subsystem contains its own special-purpose computing architecture and electronic circuitry. This circuitry sometimes employs hundreds of cards to achieve the relevant display fidelity. But even these subsystems are very much dependent upon the timeliness and quality of the data inputs they must receive from the host computer complex, which contains the behavioral model of the aircraft.

Over the years, aircraft have increased steadily in complexity, standards of expected simulator fidelity have risen, and elaborate training system capabilities have been superimposed upon the basic simulator task of reproducing vehicle and weapon system behavior. As a result, processors have been paralleled within the central standard minicomputer complex to cope with the increasing computation load. Three, four, or five CPU's within such a complex, for a single simulator station, are not uncommon. Figure 2 shows a standard means for achieving such a tightly coupled central computer complex. The two key features are use of shared memory to communicate between processors and use of a single CPU for centralized control of all linkage to the simulator equipment and for control of all interfaces to peripheral devices.

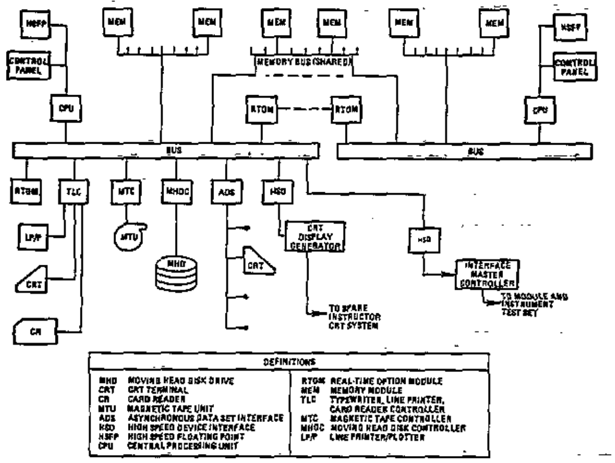
HSFP
CONTROL PANEL
MEM
MEMORY BUS (SHARED)
CPU
RTOM
BUS
RTOM
TLC
MTC
MHDC
ADS
HSD
LP/P
MTU
CRT
CRT DISPLAY GENERATOR
MHD
CRT
CR
TO SPARE INSTRUCTOR CRT SYSTEM
INTERFACE MASTER CONTROLLER
TO MODULE AND INSTRUMENT TEST SET
DEFINITIONS
MHD MOVING HEAD DISK DRIVE
CRT CRT TERMINAL
CR CARD READER
MTU MAGNETIC TAPE UNIT
ADS ASYNCHRONOUS DATA SET INTERFACE
HSD HIGH SPEED DEVICE INTERFACE
HSFP HIGH SPEED FLOATING POINT
CPU CENTRAL PROCESSING UNIT
RTOM REAL-TIME OPTION MODULE
MEM MEMORY MODULE
TLC TYPEWRITER, LINE PRINTER, CARD READER CONTROLLER
MTC MAGNETIC TAPE CONTROLLER
MHDC MOVING HEAD DISK CONTROLLER
LP/P LINE PRINTER/PLOTTER

Figure 2 MULTIPLE CPU'S IN TRADITIONAL CONFIGURATION

Efforts to keep up with the mounting computation load by continuing to add additional tightly coupled CPU's introduce problems that are far from trivial, and these difficulties increase as some nonlinear function of the number of CPU's being coupled. Even if this could be ignored, changes in avionics configuration in present-day aircraft have imposed further strains upon the traditional architecture. Aircraft systems employing highly sophisticated "black boxes," each of which is capable of processing large amounts of information, have become distributed processing systems themselves, using buses such as MIL-STD-1553 or ARINC 429 to communicate between processors.

Since the host computer, with its linkage hardware, is not responsive enough, these aircraft units cannot be interfaced through conventional linkage like traditional panels or controls. Simulators have had to incorporate special interfaces, which have their own intelligence, to manage the communication protocol and to provide some local processing capability.

Disadvantages of Centralized Computation

As is usually the case, the search for alternatives to the traditional simulator approach has been motivated by the need to improve performance while, at the same time, attempting to reduce the cost of ownership. With regard to performance, both the computer vendors and the simulator manufacturers have steadily increased performance of minicomputers by new developments, novel architectures, special hardware features, and various software techniques. However, this performance improvement, since it remained within a centralized host, has not always been available to meet the specific demands for higher performance within individual subsystems. For example, in the case of control loading systems, it has been found more effective to achieve the high digital computation rates required to improve the dynamic responses of that system by providing specialized dedicated microprocessors embedded within the subsystem, rather than to attempt to create an adequate time segment within a host computer to accomplish the same result.

With regard to ownership costs, it should be emphasized that we are not concerned with the direct hardware cost associated with computation. Rather, we are concerned with the associated costs attributable to problems of gaining access to the centralized computing complex from multiple users and problems associated with unwanted and error-producing interactions between software components residing within a centralized host. These are both reflected in increased costs associated with software control, test, and debug and with the difficulty of maintaining integrity of already proven and error-free software during the rearrangements necessary to accommodate equipment change, update, reconfiguring, etc.

The need for more effective performance when judged by these criteria led to the study and analysis that resulted in the system described in this paper.

Conceptual Development of Distributed Computation System

In 1980, when we began a study of distributed processing techniques for flight simulation, the digital flight simulator computation system had expanded through the application of new technology and design techniques for almost two decades. This expansion had always been implemented within the constraints imposed by a central computer and input/output system. This process had led to a situation in which the behavior, structure, and basic requirements of simulator computation were no longer well understood, even by simulator designers. The first part of the study, therefore, concentrated on determining the real nature of the computation process, with particular emphasis given to the all-important problems of partitionability and of interfacing to the rest of the simulator. We collected from a range of simulators, which included an F-16, UH-60, C-130 and a B-747, data that covered:

1) Definition of major functions (modules that represent primary simulator functions, e.g., engines)

2) Hardware assembly and software interfaces

3) Module computational requirements

4) Major function computational requirements

5) Major function interfaces

6) Hardware panel and major function interfaces

7) Record/playback/reset interfaces with major functions

The interfaces between major functions (point 5) are key parameters in determining the partitionability of a computing system. The study found that a major source of misunderstanding arose from the traditional unconstrained use of shared memory.

The host computer had always contained large amounts of shared memory, and the contents of shared memory have long been thought of as "common data," "global data," or even "cross talk." The difference between "shared data" and data transmitted between various major functions was not an apparent or even a relevant design parameter. Shared memory did contain true interface data, such as engine thrust, to be passed between functions, but it also contained common data, such as the value of 180°, -1, etc. It contained true global data variables, but it also contained many more unglobal extras. It had originally been adopted as a convenience (partly to avoid having to make partitioning decisions!), and its retention and popular use had obscured the functional modularity of the actual simulator software.

The analysis showed that in the case of the C-130, for example, true functional partitioning of the software resulted in 1,341 inter-major-function parameters (not counting training system software). When we considered module iteration rates and the needs of a typical training system or instructor operating station, we arrived at a data rate of 9,531 words (32 bits) per second as the intermodule communication requirement for a functionally partitioned C-130 simulator.

Data obtained from the other simulators studied (see Table 1) followed the same pattern and agreed reasonably well with those from the C-130. Since our samples had been selected to cover a broad range of simulation, including a military helicopter, a high-performance fighter, and two types representing military and commercial versions of multiengine aircraft, it was safe to conclude that the natural structure and interplay of major simulator systems was compatible with the concept of distributed computation, though this had been masked by the undisciplined use of shared memory.

Table 1 COMMUNICATION BETWEEN MAJOR FUNCTIONS

AIRCRAFT TYPE	BROADCAST DATA (32-BIT WORDS)	TRANSFER RATE (WORDS/SECOND)
F-16	697	12,345
UH-60	627	16,809
B-747	2,129	14,778
C-130	1,341	9,531

This conclusion was further reinforced by data obtained in other parts of the study, in which a surprisingly close correlation was shown between the simulator software (both at the module and the system level) and the hardware, such as panels, controls, and instruments, associated with the given systems. It appeared feasible to think in terms of a number of dedicated computation centers, each directly associated with a major function or major simulator system.

"Distributed processing" does not in itself define either a system architecture or the type of processor to be used. However, for reasons associated with functional modularity, the then-emerging microprocessor technology seemed to offer the greatest promise for a true advance in simulator architecture.

It is important to understand the issues at stake in this decision. The simulator study had used the term "major function" to represent a collection of software modules that would remain closely coupled in a functional partitioning of the simulator. Good design methods could ensure a functional hierarchy of software during design, and the benefits of this technique were well documented and understood. In the past, these methods had not been applied within the host computer structure; hence, whatever benefits may have been derived from functional modularity during software development had been lost from the start of integrated testing, and forever after. The results of our study showed clearly that this was not inherent in the simulator requirement and recommended techniques of software management which would maintain structured software functionality regardless of the processor characteristics. The advent of the "supermini," with its internal capability for multiprocessing, was especially compatible with this concept. The supermini, when employed with the structured concepts which had been proven viable in the simulator data study, could lead to a substantial improvement in untangling the chaotic state of simulator software organization.

If this half-way solution had been accepted, we could never have obtained the full benefits of total functional partitioning. Full functional partitioning implies more than development of software within protected environments, crossed only via carefully defined interfaces (which can be provided by the proper utilization of standard centralized computer architectures). It also implies the ability to specify completely functional packages, to design, fabricate, and assemble along these same functional boundaries, and finally to test functional systems as complete modules independent of the status of other modules that comprise the entire simulator. This implies embedding the computational capability itself in the system module design in a way that is not feasible with minicomputers -- and even less with superminis -- because of their packaging and unit cost. Furthermore, it implies an intimacy between computation and simulator hardware which is not provided within conventional computer linkage concepts. "Distributed processing" was only one of the essential elements needed to achieve true functional modularity, and for reasons associated with the size of the computing capacity increments, the packaging possibilities, and the treatment of I/O channels with respect to the computation process, only microprocessor technology seemed to offer the possibility of achieving such modularity.

Important questions had to be answered before the decision to take this approach could be made with confidence. These questions centered upon the the ability of microcomputers to handle the necessary computation at rates and accuracies which not only would match the capabilities of current conventional technology but also would allow for future enhancement of those standards. We took to be representative of then-current simulator technology the performance of minicomputers whose capability was about 0.7 MIPS (Millions of Instructions Per Second) and supermini's capable of about 2.5 MIPS, each executing FORTRAN ANSI 77, with a 32-bit word length.

In the discussion that follows, it is important to note that the design decisions described, and the performance achieved, were made using the technology available in the years during which the system was developed (1982-1984). Nothing is more axiomatic in modern technology than the year-to-year improvement we now expect in the state of the art. Therefore, the system to be described should be considered as a baseline design that is already in the process of being upgraded. This upgrading, described below, does not in any way change the structure of the system.

The microprocessor family selected for evaluation and later incorporated into the system design was the Intel iAPX86. In the early 1980's, this was the only family which offered a hardware floating-point processor as well as full software support for FORTRAN ANSI 77. Both capabilities were essential to achieve the necessary performance in a design that would be compatible with modern simulator software technology as well as with our own facilities for software development and support.

Since individual microprocessors clearly could not match individual minicomputers for computational throughput, it was necessary to assume that several micro's would have to be clustered within at least the larger major functions. The analysis of module rates which had been made in the simulator study had shown the performance required from each functional area. The predicted performance for a single microprocessor under typical simulator benchmark conditions was taken to be 200 KIPS. This indicated that the most demanding requirement -- the flight major function in an aircraft with unusually complex function-matching requirements -- would require six such microprocessors. It was well within the capacity of the multimaster bus system being considered for the design to accommodate this number of microprocessors. Further research successfully proved the feasibility of partitioning the flight area modules among six processors to achieve the required performance.

The requirement, however, was that this computation capability had to be achieved without compromising the 32-bit performance that had become industry standard for simulators. It is especially interesting to consider the impact which the architectural approaches taken by microprocessors have had on this question.

In the minicomputer environment, the use of 16-bit or 32-bit words had been definitive in establishing computational power. The transition from the former to the latter was accompanied by substantial upheaval and increased cost at a time when computational power was still a significant cost factor. By their choice of architecture, and by taking advantage of very high levels of integration within VLSI technology, the microprocessor manufacturers have practically bypassed the factors that made the word-length question such a determinant of performance in the minicomputer environment. Of course, at the present time, all major manufacturers either have started production or have announced intended production of "true 32-bit" microprocessors along the growth path of their present designs, and by 1986 these components will be available in production quantity. The point is, however, that even in the 1982-1984 time frame, the technology already existed to achieve 32-bit performance, using microprocessors whose data words were a nominal 16 bits -- the traditional indication of a performance limitation. The three factors which contributed to this were:

- Microprocessor families include numeric coprocessors as working partners with the processor chip itself. For example, in the Intel family used for the system being discussed, the 80287 Numeric Processor Extension is used with the 80286 microprocessor and is dedicated to executing particular functions or instructions associated with numeric operations. It is more than a hardware add-on mathematical unit, in that it shares the local bus with the host processor and can access memory. The coprocessor interface allows specialized hardware to appear as an integral part of the host architecture. All instructions that require numerical operations are executed by the coprocessor, which can handle 32- and 64-bit floating-point data types. This capability is transparent to the programmer, who simply specifies the data type when writing source code in a high-order language such as FORTRAN or Pascal. The code that executes on the coprocessor is generated automatically by the object code compiler.

  The use of a coprocessor is a powerful concept. The coprocessor increases the performance of processors in particular applications and is not just an add-on arithmetic unit to overcome a resolution limitation with 16-bit processors. The 32-bit microprocessor families, including such examples as the Intel 80386 and the Motorola 68020, will continue to use coprocessors.

- The use of a 32-bit-wide bus in a minicomputer environment enhanced performance by reducing the time to fetch data and by providing for single fetches of instructions greater than 16 bits in width. The same benefits will also accrue when using 32-bit microprocessors but will be of considerably less importance. For example, the iAPX 286 has a bus interface that can achieve 8 Mbytes per second. Throughput is enhanced by using a prefetch instruction queue, which results in a limited pipelined operation. The bandwidth of the CPU bus, as in the case of the minicomputer, is limited by the technology that drives the bus and by the activity, or contention, on that bus. A 32-bit data word, as opposed to a 16-bit word, is of secondary importance in this context.

  Performance limitations that result from requiring double fetches of data words simply determine the number of microprocessors to be employed to serve the needs of a given major function. When the system migrates to the use of full 32-bit microprocessors, it will be possible to reduce this number in some cases. In other major functions, there will be no change. In neither case will there be a difference in quality of computation, since the current practice of maintaining 32-bit or 64-bit precision will continue to be employed. In particular, the availability of the greater performance per microcomputer which will result from the transition to the 32-bit generation will not be used to reduce the number of major functions in the simulator. As we have seen, the number of major functions results from optimizing the structural organization of the simulator, not from a need to economize on the number of microprocessors employed.

- The third benefit conferred by the use of 32-bit wide buses is an extension of the addressing range of the computer. Both the SEL 32/77 and the iAPX 286/10 have a 24-bit bus, which shows that 32-bit performance is not related directly to the addressing capability. The addressing range of the iAPX 286, with its integrated memory management, is 16 Mbytes of physical, and 1 Gbyte of virtual, memory.

  It is not the size of programs which require this massive addressing capability, but the technique of structured programming and the desire to provide system protection for programs and data by means of segmentation which can be effectively applied in line without performance penalty. The transition to 32-bit microprocessors will make it possible to provide these structural advantages more lavishly than at present and therefore will represent an improvement at the software system design, rather than at the computation, level.

Provision for Simulator Management

The requirements for simulator management, taken in the broadest sense, were considered, and it was clear that many of the facilities hitherto provided in the central host computer could be not only equaled but improved upon with distributed microprocessors. Functions such as built-in system tests, real-time performance monitoring, debug, plus other requirements for product support were identified as candidates for separate, dedicated computer resources. Through distributed processing, each function could be satisfied without compromise with the real-time operations. In fact, integration of these "management" functions and the means by which they could communicate with external resources was as important to the successful development of the distributed computation concept as the derivation of the real-time processing system itself. The microprocessor environment provided a natural avenue by which the necessary communication needs could be satisfied and optimized for each function that had to be provided. The vehicle by which this was accomplished was the development of a bus structure to provide both inter- and intra-major-function communication suitable to the total needs of the simulator.

Although many of the management functions which had to be exercised during the actual operation of the simulator could best be accomplished at the distributed, i.e., functional, level, there remained a set of global functions which could not reasonably be allocated to any particular major

function module. The solution here was to incorporate into the system architecture a single, dedicated *management* **computer,** *similar in structure* **to the computers used in the major function modules, but dedicated in its software and its interfacing entirely to the global management requirements of the simulator.**

Design Considerations for A Distributed Microprocessor System

Data gathered on existing simulators, in terms of module timings and interface requirements, could now be related to microprocessor performance and potential architecture to arrive at a general specification.

Local and global bus structures were defined which supported the functional partitioning that the simulator study had shown was achievable for even the most complex simulators. Since a distributed microprocessor system implies the use of many processors, the specification of this bus structure is critical; it is described in detail below. The required global bus performance can *vary enormously, depending upon how global memory* is defined. The functional partitioning which was preserved in the system specification gave the desired benefit of minimizing this traffic, thus permitting use of a message system interface with quite low bandwidth requirements.

The remaining attributes of the distributed *system architecture can be summarized as follows:*

1) A total system could range from four to 40 processors, depending upon the complexity of the simulator. These would be grouped into from four to 16 major functions, depending upon complexity.

2) All processors would receive synchronizing timing signals.

3) Each processor would have a time-frame structure similar to that imposed by current computer executive programs.

4) All processors would use "data messages" to control execution.

5) Each processor would have access to private memory via its own private bus.

6) Each major function would have immediate access to its own dedicated linkage system.

7) Hardware inputs and outputs would be mapped into local memory.

8) Each major function would contain:

   a) executive
   b) debug
   c) initialization
   d) record/playback
   e) malfunction control
   f) freeze control
   g) error message logic
   h) linkage handler

9) Peripheral access would be provided through only one processor or processor cluster, which would be tasked with global manage*ment functions for the entire simulator.*

Functionally Distributed Simulation

*The research study and the available micro*processor technology clearly indicated the feasibility of developing a distributed computing system. To convey the basic design philosophy, this radical new approach was called the Functionally Distributed Simulation (FDS) system. This section describes the FDS structure and performance characteristics. (See Figure 3.)

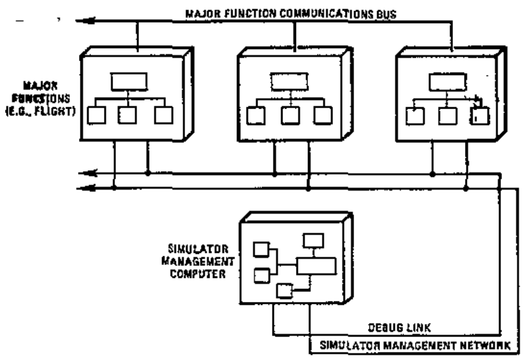
MAJOR FUNCTION COMMUNICATIONS BUS
MAJOR FUNCTIONS (E.G., FLIGHT)
SIMULATOR MANAGEMENT COMPUTER
DEBUG LINK
SIMULATOR MANAGEMENT NETWORK

Figure 3 FUNCTIONALLY DISTRIBUTED SIMULATION

System Overview

Real-time computing requirements are partitioned on a functional basis and referred to as "major functions." The major functions are autonomous, i.e., each contains processing power, memory, and linkage for one (or more) functions, such as flight, engines, etc. To achieve the desired degree of modularity, the major functions also have clearly defined hardware and software interfaces to other functional areas. Although the more traditional methods of achieving modularity through resource partitioning can optimize a given *resource, such as CPU time, functional partitioning* provides more important advantages, such as flexibility and upgradability.

The major functions provide real-time processing. A simulation management computer provides product support, primarily through software tools such as editors and compilers, plus other software for total simulator maintenance and diagnostics. Functional separation of this computing facility, a general-purpose microcomputer and industry standard operating system, from the demanding real-time requirements is an important characteristic of FDS.

The major functions and the simulation management computer are connected by three main communication paths -- the major function communication bus, the simulator management network, and the debug link. The rationale for using these buses is to provide paths matched to data requirements and to utilize message passing between units instead of traditional direct memory access; this ensures simple, clearly defined interfaces. Without these features, the system could not truly be considered distributed processing, and major functions would not exist.

Major Function Communication Bus

The major function communication bus provides a high-speed (8 MB/s) path between major functions for time-critical data transfer. The data are transferred using a broadcast system with a simple *hardware/software polled protocol. During a real-*

time frame, typically 33 ms long, each major function calculates a predetermined amount of data required by one or more other major functions. Before beginning processing of the next frame, each major function takes its turn to broadcast these data. (See Figure 4.)

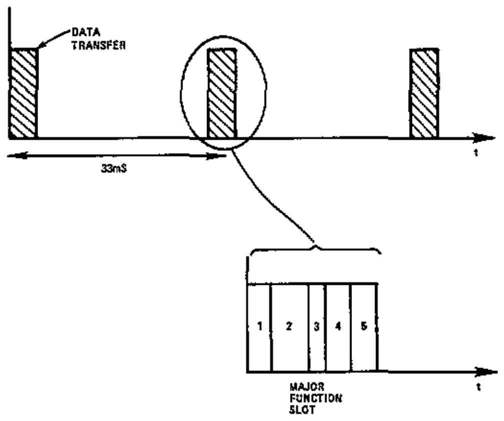

Figure 4 MAJOR FUNCTION COMMUNICATION BUS

Since each major function has a relatively small amount of data to broadcast (as discussed previously), the total time to complete this transfer is 1-2 ms.

The bus specification ensures data integrity by adopting a synchronous transfer and double buffering of data within the major functions. In this distributed system, with each computing system consisting of multiple CPU's, the emphasis was to achieve stable, valid, and coherent data throughout each real-time frame. An asynchronous operation would have required a more complicated software protocol and the design of multimaster bus arbitration schemes, for which the state of art in software tools and the prerequisite use of FORTRAN ANSI 77 did not provide the necessary facilities. Other bus systems, including the use of industry standards such as ETHERNET, did not provide enough bandwidth. Those that did (typically the vast number of microprocessor system buses such as VME, or Multibus I) did not satisfy the requirements of a global bus for a distributed system striving to achieve modularity.

<u>Simulator Management Network</u>

Not all data that pass through a simulator computer are time-critical with respect to real-time performance. Since the nature and purpose of these data present a very different requirement, we use a local area network (ETHERNET) as the communication bus. Primarily, this gives each major function access to a mass storage device associated with the simulator management computer, enabling transfers of data for program loading, diagnostic reporting, and access to database information. Linking all major functions in this manner to the simulator management computer is a very light load for ETHERNET, so there is good performance -- typically 20-30 KB/s. The other benefits of ETHERNET, apart from its being a de facto industry standard, are the low cost per node for this level of performance, made possible by using VLSI components, and also its ability to expand the resources served by the simulator management network (with other computers, file servers, printer servers, etc.). Incorporating a local area network in this role provides a good opportunity for vertical integration.

<u>Debug Link</u>

For a large system, we use a separate communication path for debugging major functions. As discussed later, each major function has a dedicated CPU for this role. It needs access to information within the simulator management computer. An additional network for this purpose provides a high degree of fault tolerance.

In addition to transferring predefined debug pages on request, the debug link gives access to the symbol dictionary on disk, to give a real-time symbolic debug capability, even when a major function is being worked on independently. ETHERNET again gives a good performance through its low latency for low loads. A symbol can be typed at the debug terminal and transmitted through the debug link to the simulator management computer, which accesses the Symbol Dictionary on disk to obtain the absolute address. This is transmitted back to the major function so the debug CPU can display the contents of that symbolic variable. Typically, this total process takes less than one second.

The debug link also permits accessing major functions or performing debug tasks from the simulator management computer. The networks ensure that the advantages of distributed processing are maintained but make global access to information from one centralized point available when applicable.

<u>Major Functions</u>

The system architecture of a major function is shown in Figure 5, which shows two main buses dedicated to internal processing.

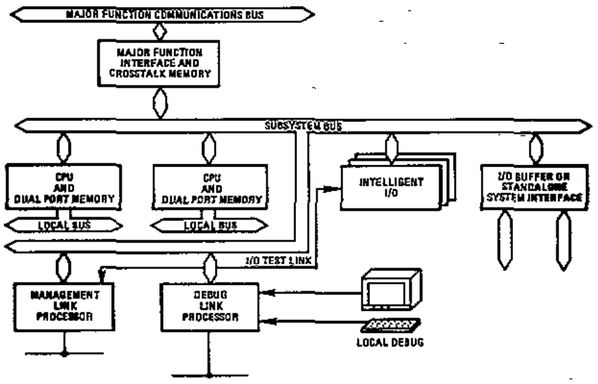
MAJOR FUNCTION COMMUNICATIONS BUS
MAJOR FUNCTION INTERFACE AND CROSSTALK MEMORY
SUBSYSTEM BUS
CPU AND DUAL PORT MEMORY
CPU AND DUAL PORT MEMORY
LOCAL BUS
LOCAL BUS
INTELLIGENT I/O
I/O BUFFER OR STANDALONE SYSTEM INTERFACE
I/O TEST LINK
MANAGEMENT LINK PROCESSOR
DEBUG LINK PROCESSOR
LOCAL DEBUG

Figure 5 MAJOR FUNCTION SYSTEM ARCHITECTURE

Using a parallel system bus for communication and module-wide control functions, plus a local bus associated with each processor to maximize performance when multiple processors are used, is typical of the latest advanced microprocessor bus systems. All the circuit cards required for a major function, including linkage cards, are connected through the parallel system bus.

<u>The Major Function Interface Card</u> -- The major function interface isolates each major function in a manner independent of the number of major functions or the number of processors within them. This results in full functional partitioning. The

major function interface contains on one circuit card all the hardware and memory for the major function communication bus. Data broadcast is controlled by hardware and is executed independent of any other activity within the major function. The 32 KB of memory on each major function interface card is mapped, so that the area for each major function in the system is unique. This is programmed with the system software so that there is flexibility in terms of the number of major functions and of the amount of data each broadcasts.

Processor Circuit Card -- The microprocessor card executes the real-time software modules. The multiprocessing capability within the major function gives high performance to meet the requirements of the functional areas. Each card contains multimaster arbitration logic and the interfaces to the two buses that communicate with other processors within the major function. Each card also contains memory and other components to provide local resources for low-level debug and built-in test. This gives a significant advantage when designing or testing microprocessor systems, because a single circuit card requires only a very few external resources for complete test or debug.

The performance of the processor circuit card using the Intel 80286 microprocessor and 80287 coprocessor, as used in the FDS-based simulators to date, has been measured, using company benchmarks that represent flight simulator software written in FORTRAN ANSI 77. The results are shown in Table 2.

Table 2 COMPARATIVE PROCESSOR PERFORMANCE IN KIPS

BENCHMARK	STD. MINI	SUPERMINI	80286/287
1	383	2448	97
2	708	6694	328

Benchmark 1 consists primarily of floating-point operations; benchmark 2 has a mixture of floating-point and integer operations. The minicomputer used a single CPU with a high-speed floating-point unit. The supermini used a single CPU with a high-speed floating point unit and 12 Kbytes of cache memory. Both benchmarks were small programs that ran completely in cache memory. A normal simulator load would not fit into cache, resulting in a reduction of the performance shown for the supermini.

Three additional benchmark results are provided which illustrate the point previously made regarding the ability of even "16-bit" microcomputers to achieve or surpass 32-bit minicomputer performance as a result of their architecture and reliance upon dedicated coprocessors. The results are shown in Tables 3, 4, and 5.

Table 3 FLOATING-POINT MATH

COMPUTER	SINGLE-PRECISION 32-BIT RESULT ERROR	DOUBLE-PRECISION 64-BIT RESULT ERROR
FDS	15×10^{-3}	1×10^{-12}
COMP. A	473×10^{-3}	4537×10^{-12}
COMP. B	31×10^{-3}	11×10^{-12}

Table 4 SPIRAL NAVIGATION BENCHMARK

COMPUTER	DIRECTION OF SPIRAL	MAX ERROR, LATITUDE (DEG x 10^{-6})	MAX ERROR, LONGITUDE (DEG x 10^{-6})
FDS	Clockwise	6	320
	Ctr Clockwise	42	397
COMP A	Clockwise	244	1984
	Ctr Clockwise	168	626
COMP B	Clockwise	31	153
	Ctr Clockwise	122	290

Table 5 INTEGRATION BENCHMARK

RATE OF CLIMB (FT/MIN)	FDS % ERROR	COMP A % ERROR	COMP B % ERROR
1	100	100	100
2	76	100	100
3	17	100	100
4	12	100	100
5	30	100	100
6	17	100	100
7	1	100	100
8	12	12	12
9	17	22	22
10	6	29	30

The first benchmark primarily evaluates arithmetic performance, and especially the precision and accuracy achieved in floating-point operations. In its outer loop, the benchmark sums the values 1 to 100 and subtracts the result from the nominal value. Therefore, the result should be zero. To exercise the floating-point process vigorously, the square root of each successive value is taken 10 consecutive times and then resquared 10 times before it is added into the sum of values. The data presented in Table 3 show the results for both single-precision (32-bit) and double-precision (64-bit) floating-point values.

Table 4 shows results of benchmark tests specifically relevant to navigation calculations in simulation. It consists of range, azimuth, and reset calculations taken from a starting position which is successively incremented by 100 range steps of 10 nautical miles and 100 bearing steps of 3.6°. The test is then repeated by decrementing through corresponding steps until closure to the original position. All calculations are performed in double precision (64 bits). The maximum deviation recorded upon calculating reset to the starting point from any test point along the traversed spirals is indicated in Table 4.

The final benchmark demonstrates the effects of rounding errors during an integration process such as that required to solve the differential equations representative of aircraft flight. In the particular case chosen, aircraft rate of climb is integrated to obtain a solution for aircraft altitude. In a simulator, these processes are important to achieve the desired "feel" of the aircraft. As the rate of climb is reduced, the altitude change produced becomes inaccurate, because of computational error, and finally ceases to change at all. It is important that the rate of climb at which this occurs be imperceptible to the pilot. The magnitude of the error for the change of altitude over one minute (expressed as a percentage) is tabulated against rate of climb in Table 5 for each of the three computer types being compared. (In all cases, double-precision calculations gave results very close to the correct answers.)

The Debug and Management Link Processors -- These two circuit cards are based upon a common design; the debug card has an additional module that drives a memory-mapped VDU and keyboard. In keeping with the overall design philosophy, these two cards provide functionally independent resources for system debug and real-time status monitoring. (These extensive facilities are covered later in the software section.) The management link processor provides the main communication path to the simulator management computer, assimilates the status of the major function, and reports back to the SMC. The components of this highly integrated circuit card are:

Microprocessor	--	80186
ETHERNET processor	--	82586
VDU controller	--	8275

Linkage

To increase functionality and to improve latency through the entire system, each major function contains its own linkage. The linkage card configurations depend upon the requirements of the major functions. Since this sometimes necessitates more circuit card slots than the standard card bin has, additional bins are used solely for linkage in these cases. They are tightly coupled to the major function processors.

System Design -- One of the most significant advantages of microprocessors is the ease with which I/O devices can be connected. Microprocessors have a relatively simple interface specification for control signals and are highly standardized. With high-performance systems using multiple microprocessors, this feature can be retained; with single minicomputers of comparable performance, the interface signals and timing present a much more demanding design for I/O. As the performance of a single processor increases, not only does the amount of centralized I/O in a flight simulator increase, but it becomes more difficult to provide an interface with low latency. In general, simulators have never had I/O directly connected to the host processor. Centralized linkage has been serviced by using a Direct Memory Access (DMA) unit and other distribution circuitry. Partitioning the linkage functionally reduces the I/O throughput and offers the potential of lower latency; the increase in processor performance and higher iteration rates can be combined with improved I/O techniques to realize higher fidelity in real-time simulation.

Linkage Built-In Self-Test -- A typical flight simulator uses 200-300 linkage circuit cards, which range from simple electrical drives to high-speed serial interfaces for as-in-the-aircraft avionics. The economics of simulator usage demand that fault diagnosis and routine maintenance be as quick and infrequent as possible.

Not only can microprocessors be configured to replace conventional minisystems, but they can be applied in areas previously restricted by space or cost. The philosophy of the FDS linkage is to have a microprocessor on each card to provide local intelligence for built-in system test. This gives significant advantages over having to use the host computer as the only source for diagnostic software:

- o There is a higher level of diagnostics, since the processor is closer (and easier to interface) to the components it is monitoring and can devote all its time to this task.
- o It provides functionality, i.e., the built-in system test remains with the circuit card when it is removed from the system environment.

Figure 6 illustrates the linkage built-in-system test philosophy of FDS.

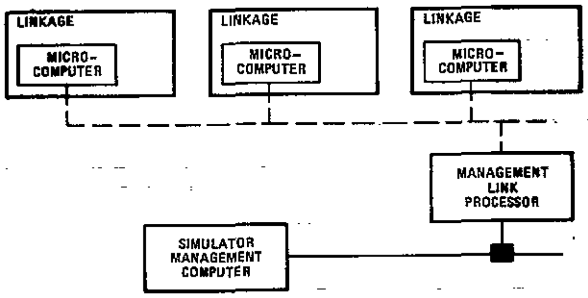

Figure 6 FDS BUILT-IN SYSTEM TEST

The general facilities provided by the on-card microprocessor are:

- o Power-up confidence
- o Real-time monitoring
- o Configuration control information

Intelligent Linkage Cards -- In conventional simulators with single host computers, certain interfaces have to have local processing power or intelligence. These are usually associated with special high-speed interfaces to aircraft avionics. The ability to give linkage circuits intelligence in the form of microprocessors can continue the design concept of functionality to the circuit-card level. FDS uses a number of such circuits for driving serial data highways, for special instruments, such as synchros, and for multiplexing simple switch/indicator channels.

The AC synchro driver is a typical example of the improved functionality that can be achieved with an on-board microprocessor. Each card can drive two synchros. It requires only a demanded angle and channel number. The local firmware calculates the necessary output voltages and damping requirements, which in the host computer traditionally would be called as sequential subroutines. Thus the main math model, in simulator software, is isolated from final instrument consideration.

As previously described, the local intelligence can also be used to provide maintenance and diagnostic facilities beyond those generally applicable. The AC synchro driver has seven tests for slewing and stepping, at various rates selected by on-board switches. All these tests are available, even when the card is isolated, and can be used independent of other activities for checking the circuit card and the driven instrument.

FDS Software

One FDS design criterion was to retain as many software development tools as possible and to be able to run existing FORTRAN-based application modules for specific aircraft. Transfer of FORTRAN programs for a SEL or Perkin Elmer minicomputer to an Intel microcomputer system proved very efficient, with very few changes required. This section describes some of the major software for an FDS system that strongly resembles that of current minicomputer systems.

Real-Time Executive -- Most simulators have a proprietary real-time executive that runs and schedules the real-time modules. FDS relies upon the same principle of a synchronized real-time frame of, say, 30 Hz (dependent upon specification), but each major function has its own executive. A synchronous scheduler synchronizes the major functions and the multiple processors within each major function. It provides for different frame lengths, and module execution rates can easily be changed, since the elapsed time between module execution is calculated dynamically.

Real-Time Debug -- The efficiency of debug in a large, real-time software system requires considerable attention. In FDS, the debug system has been functionally partitioned from the real-time environment by allocating a separate microprocessor, memory, and interface. This gives a high degree of fault tolerance during development, because problems in the real-time processors have little or no effect on the debug software. Information available through debug includes module and frame timings, 30 continuously monitored symbols, system faults, and executive/linkage status. Debug commands permit enable/disable of modules and linkage, modification of displayed symbol values, and change of VDU pages. A hardcopy of VDU pages can also be produced at the SMC using the debug link.

Off-Line and On-Line Tools -- Many software development tools are required for flight simulation, and these have been developed for FDS systems. On-line tools, such as debug and real-time error reporting, are provided by dedicated microprocessors. Off-line tools for maintenance, diagnostics, and software development are also provided. While these are comprehensive and are important to simulator development, they do not currently exhibit any novel concept that involves the use of microprocessors. We await the day when software engineering and advanced workstations combine to form a more efficient environment within reach of all.

Typical Configurations

Functionally Distributed Simulation has now been applied to many training devices, including Fighting Vehicle Simulators, Wide-Body Airline Simulators, Engine Maintenance Trainers, Jet Fighter Simulators, and Submarine Control Simulators. All these require extremely different performance and configurations, but each is based upon the same concept and modular building blocks.

A British Aerospace Hawk Simulator and an Airbus A300-600 Simulator represent typical configurations. They contain, respectively, five major functions, with a total of nine CPU's, and seven major functions, with a total of 27 CPU's, for their real-time application software.

Benefits of Functionally Distributed Simulation

The following sections summarize the benefits of FDS, including basic product configuration, development methods, and cost of ownership.

Functional Modularity

The benefits of partitioning a problem into smaller, more manageable tasks are well understood. Traditionally, this has been achieved to a limited extent by resource partitioning, as opposed to functional partitioning. Although there are advantages associated with high-performance centralized computing systems for applications that cannot be partitioned easily, these systems tend to be extremely complex, and they lack the ability to respond effectively to changes in requirements. Resource partitioning requires a tightly coupled system with high bus bandwidth, and it does not encourage standardization of interfaces.

Functional partitioning gives a modular design that reduces complexity and gives local autonomy of response and the flexibility to incorporate changes. VLSI technology permits achieving functionality, and this functionality permits applying new VLSI devices to each functional area. The functionality also allows the benefits of the early phases of parallel development to continue much longer, easing test and calibration and software updating.

Parallel Computation

Parallel processing is now a way of life and the only practical way of achieving the computing performance required for Artificial Intelligence, Computer-Aided Engineering, and Real-Time Simulation. It is now relatively easy to connect many hundreds of microprocessors in integrated systems, although, unfortunately, these do not always provide a suitable environment for all problems. True concurrency of operation on tasks that are complex and not easily partitioned is still awaiting a solution.

FDS uses parallel processing for an application that can be partitioned. It is not unreasonable to expect that the real-world behavior of a machine can be simulated by a number of parallel processes. The concept of parallel processing is not complicated or novel; the problem in the past has been the attempt to represent the real world as a sequential process. Parallel processing can achieve a significant increase in responsiveness and at the same time reduce overall complexity.

Reliability

FDS has improved the reliability of a Flight Simulator by its use of VLSI technology and by its modular design. The number of electronic devices and their methods of fabrication are fundamental to system reliability. For a given simulator, FDS uses approximately half the number of electronic devices as does a simulator based on a centralized minicomputer. The manufacturers of microprocessors are also beginning to use advanced CMOS technology. This not only permits even higher levels of integration through lower heat dissipation but makes microcomputers more reliable than minicomputers that use TTL or ECL technology.

A modular design is less complex; a system with 16 major functions is not necessarily more complex than one with just two major functions. The design concept also achieves a high degree of fault tolerance through graceful degradation. Although certain failures still impact total training availability, using a distributed processing system and dedicated microprocessors for diagnostics improves fault isolation and maintainability.

Cost of Ownership

One of the most important factors in training today is the cost of ownership of these increasingly complex devices. Maintenance and running costs over many years can exceed initial procurement cost, and the need to upgrade performance very often exceeds the initial design specification requirement. The FDS design has addressed the complex cost-of-ownership equation and has minimized it by using a new design concept that incorporates new, advanced technology.

Support and maintenance are very significant factors in cost of ownership, and the FDS design has now adopted industry standards wherever possible. Although this was not feasible in the early days, because standard bus architectures lacked the necessary performance, the latest buses developed by various microcomputer manufacturers satisfy basic FDS requirements. The major functions are now based upon Multibus II instead of on the original proprietary bus system. All major computing elements for these industry buses, such as CPU and memory circuit cards, are available commercially from more than one vendor. The modular design of FDS and its adoption of industry-wide standards are significant steps in reducing the cost of ownership of flight simulators.

About the Author

Mr. David Parkinson is a Technical Development Manager with Singer Link-Miles Ltd. and is responsible for all Research and Development Programs undertaken at Link-Miles. He holds an honors degree in Electronic Engineering and is a Member of the Institute of Electrical Engineering.

He joined Link-Miles in 1971 as an Electronic Development Engineer. His early experience was mainly in circuit design using a variety of analog and digital techniques. After being responsible for a number of new designs associated with simulators, he became a Group Leader in 1977, supervising the staff of the Development Department. During the last few years, he has been involved in the introduction of advanced technology and associated development. Major programs which have recently been undertaken include an Electronic Warfare Simulation System, a Computer-Generated Visual System, a Distributed Microprocessor Computing System for a Flight Simulator, and a Digital Control Loading System.