MULTIPLE - MICROCOMPUTER ARCHITECTURES: AN INTEGRATED APPROACH IS NEEDED FOR SIMULATION

Charles N. Pope
David M. Kotick
Mark W. Layton

Naval Training Systems Center
Orlando, FL 32813

ABSTRACT

The advent of 32-bit busses which support full 32-bit microprocessors has signaled a coming of age of microcomputer technology for simulation applications. The VME-bus, a fully non-proprietary bus standard developed through a cooperative effort of major semiconductor manufacturers, has board-level product offerings from over 200 vendors. As this number continues to grow, so does the competitive base of the various components (in low cost, general purpose form) needed to build high fidelity simulators. Although the growing availability and variety of these off-the-shelf components do not minimize the need for sound system integration, they can facilitate this crucial process, and more importantly, afford an opportunity to rethink how it can be done better. For example, in the past, shared memory has been widely used to communicate between multiple cpu's in a training system. Should this concept be adopted in microprocessor-based systems, or will emerging message passing and data broadcasting methods serve our needs more effectively? Once the best communications mechanism has been selected, what intelligence will be needed to partition the simulation, determine the data that must be transferred, and synchronize the distributed software modules of the system such that the fidelity of the simulation math model is preserved? Research has been undertaken both within the government and by military contractors to answer these questions. The Systems and Computer Technology Division of NTSC has contributed through the building of an advanced development model based on the VME-bus and commercially available components.

INTRODUCTION

Advances in VLSI technology continue to increase the power and density of general purpose microcomputers. Naturally these advances are expanding their role in training systems from non-real-time peripheral equipment to real-time subsystems. The ability to integrate large portions of new training systems from inexpensive, commercially available components has the potential for revolutionary impact. Because the capital investment can be drastically reduced, increasing numbers of progressive smaller (as well as larger) companies with microcomputer savvy will begin to vie for simulation contracts. This coupled with the inherent modularity of these systems can result in innovative solutions to simulation requirements. On the other hand, there is the potential for a proliferation of non-maintainable microcomputer training systems that could easily eclipse that being experienced with minicomputers. Which of these presently contervailing forces will ultimately prevail depends upon the effectiveness of hardware and software integration methods which are evolving for microcomputer system development.

HARDWARE INTEGRATION

In 1981, a significant breakthrough occurred that is helping to bring unity to the diversity of microcomputer system design. This was the the introduction of the VME-bus (standing for Versa Module European), the first standard 32-bit bus supporting the new generation of full 32-bit microprocessors. There are several characteristics that make this bus very attractive for simulation applications.

- o It is completely non-proprietary, i.e., there are no associated company trademarks, patents, or copyrights to encumber product developers or users.
- o Bus operation is asynchronous, meaning that boards of vastly differing performance can reside on the same bus and each is still allowed to function at its own optimal speed without impeding the others.
- o The bus architecture establishes a continuous (memory-mapped) 4 Gigabyte address space.
- o The VME-bus has been embraced as an international electronics industry standard. As of spring 1986, approximately 1500 VME board-level products were available from 200 manufacturers, and were being used by over 2000 OEM's.
- o The flexibility of the VME-bus architecture combined with the variety of off-the-shelf components available for it facilitate the development of general purpose turnkey systems. It also provides the basic building blocks needed for the construction of more specialized machines. Irrespective of whether the system is turnkey or one of a kind, it is a distinct advantage to the user when a delivered system comprises commercially available parts as opposed to hardware and software components that have been custom designed by a single manufacturer for their products (and theirs alone), not otherwise being purchasable separately.
- o The rapidly growing competitive base of VME compatible products assures a lower initial cost of hardware components and greatly increases the probability that a product with the required functionality and of equal or superior performance will be available for

upgrading or replacement if this becomes necessary later in the system's life cycle.

o The VME-bus standard neither favors nor excludes the use of any particular 32-bit microprocessor family.

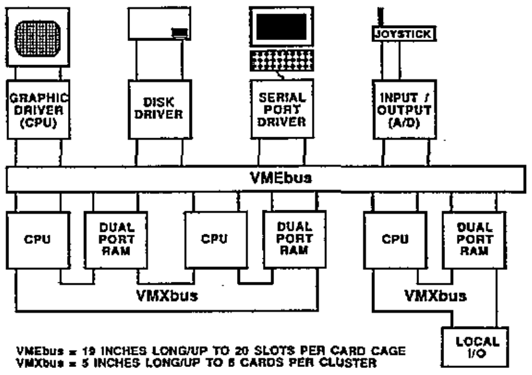

FIGURE 1
(Representative VME-bus System)

Figure 1 shows a basic VME-bus system. A wide selection of off-the-shelf single-board computers, memories, and interface cards from numerous manufacturers can be mixed and matched to meet system requirements.

Contention for the 40 Mbyte/sec VME-bus can be reduced by "clustering" boards (in groups of up to six boards) in VMX-bus configurations. The VMX-bus is an auxiliary parallel bus, and is fully defined within the VME-bus architecture through its own specification.

SOFTWARE INTEGRATION

Software Partitioning

Regardless of the interprocessor communication strategy chosen (see next section), the process of mapping the software modules of the system onto its hardware resources will be largely the same. This process, known as partitioning, can be broken up into three steps which are performed iteratively until a satisfactory configuration is achieved. As a mneumonic aid we will refer to these steps as define, align, and consign.

Define. The various functions comprising the simulation are first coded into software tasks and procedures (modules). This process has been eloquently described by numerous seminal software engineering papers and texts and would be superfluous to review in any detail here. Let it suffice to say that these principles are of equal if not greater importance in multiple microprocessor systems because of their typically distributed nature. An important rule of thumb for multiple microprocessor software design is to avoid binding large parallel functions within a single module. This will permit the full inherent parallelism of the simulation to be exploited.

"In the old days," we are told by wizened simulation veterans, "when analog computers roamed and ruled the earth," the inherent parallelism and feedback in systems flowed naturally through networks of potentiometers, integrators, and summing amplifiers. But then came the digital computer juggernaut, and in the words of a Binghamton philosopher, "they serialized the parallel universe" through inexorable conquest. Yet, even now a sea of powerful microcomputers assemble outside the gates of a rigid, centralized minicomputer empire. Will the real-time computing world be thrown into chaotic dark ages, or will a renaissance dawn with the rediscovery of the parallel essence of simulated systems. Certainly innovation is needed in the areas of math modeling and software definition. Promising new approaches such as state variables, object oriented design, and the Ada programming language have been proposed. The distributed nature of microcomputer systems will mandate the use of such advanced simulation design and software engineering implements and, perhaps, will be responsible for bringing them to an earlier fruition.

Align. With the simulation coded in task and procedure form, the data and time interfaces between these structures must be determined. In the analytical (math) model of the simulation functions are typically depicted graphically with block diagrams (see figure 2). The parameters received as inputs and sent as outputs from and to other blocks are listed on the outside of each block using appropriate mathematical symbols.

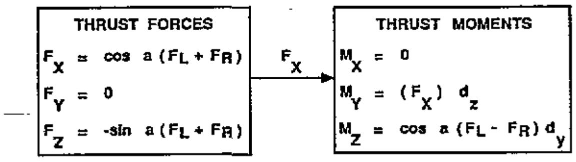
THRUST FORCES
$F_X = \cos a (F_L + F_R)$
$F_Y = 0$
$F_Z = -\sin a (F_L + F_R)$
F_X
THRUST MOMENTS
$M_X = 0$
$M_Y = (F_X)\ d_z$
$M_Z = \cos a (F_L - F_R)\ d_y$

FIGURE 2
(Partial Math Model Block Diagram)

When the model is translated into software some of these blocks are combined, some divided, while others remain the same. Therefore, there is now a need to find an analogous representation of the relationships between the functional software "blocks" (or modules) of the simulation (see figure 3). This is accomplished, first of all, by

1) identifying explicitly the high level language symbolic variable names that are the inputs and/or outputs of each module, and
2) associating with each input a fixed time constant (if the input provides feedback information to the module, e.g., in performing an integration) or a bounded time constant (if the parameter provides simple precedence information which is always valid as long as it has been updated within a reasonable time period).

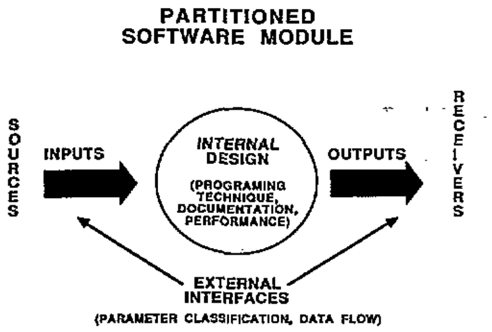
PARTITIONED SOFTWARE MODULE
SOURCES
INPUTS
INTERNAL DESIGN
(PROGRAMING TECHNIQUE, DOCUMENTATION, PERFORMANCE)
OUTPUTS
RECEIVERS
EXTERNAL INTERFACES
(PARAMETER CLASSIFICATION, DATA FLOW)

FIGURE 3
(Interfacing of a Simulation Software Module)

Next, the parameters which will be passed between each pair of modules in the system are identified, and a binding time relationship is established. For example, assume Module A passes two parameters to Module B. If one of these parameters must be exactly one real-time frame period old when received by B from A, and the other is valid as long as it is not more than ten frame times old, then a fixed lag of exactly one frame time is to be established as the required time relationship between Module A and Module B. It is the only one that will satisfy the time relationship constraints of both input parameters. This procedure is repeated for each pair of modules in the simulation that have a data interface. The end result is this - the flow of data and the real-time constraints between the modules of the entire system are precisely specified. Without this analysis the integrity of the simulation cannot be guaranteed when the modules comprising it are scheduled for execution by one or more processors.

Consign. One more step is needed before the software modules can be allocated intelligently to processing elements. It is now known which parameters are shared between modules and what the corresponding intermodule time relationships are. Now we must find reliable values for the time it will take to execute each individual module on the target processor(s). Rather than develop sophisticated estimating programs, it may be easier to bring up the simulation directly on a single microcomputer and determine best, worst and average case timings for each module as it interacts in its ultimate environment. Software callable timers on single-board microcomputers simplify this task greatly. With this data in hand, it is now possible to decide how the modules of the simulation should be divvied up so that real-time processing is achieved.

Three classes of analytical techniques have been proposed to aid in this determination.

1) Linear Programming : a linear programming model of the simulation is developed, i.e., an objective function is formulated which will either be minimized or maximized subject to the satisfying of a set of constraint equations. For example, it may be desired to minimize the total time required to execute a complete cycle of the simulation software, while keeping interprocessor communications and the number of microcomputers needed below acceptable levels.
2) Graphical : this technique involves creating a directed-arc network of the simulation software. Using a commercially available software package, the partitioner either enters the individual modules of the system as nodes in the diagram and the flow of data as arrows interconnecting the nodes (Critical Path Method - CPM model) or vice versa (Program Evaluation and Review Technique - PERT model) depending on the model implemented by the software package. The partitioner then enters the relative execution times for each of the modules. The software is instructed to calculate the total time to execute one pass of the simulation, as well as the critical path. If real-time constraints cannot be satisfied, the partitioner proceeds by reassigning modules on the critical path to other resources, i.e., microcomputers, after which a new critical path and total execution time is calculated. The partitioner continues this iterative process until a partitioning scheme results by which real-time can be achieved.
3) Heuristic : heuristic models attempt to strike a "happy medium" between unwieldy linear programming models (where solution time grows exponentially with the number of independent variables, i.e., the number of software modules) and intuitive graphical techniques. Analysis is more rigorous than graphical methods, however, less rigorous than linear programming formulations because an optimal solution cannot usually be attained.

In the cyclical type simulations found in training systems, there is usually nothing to be gained by achieving some absolute minimum execution time of the software. Instead, it is quite satisfactory to operate comfortably under some maximum real-time frame limit while preserving the logical integrity (proper sequence of execution) of the simulation programs. The exception is the case when the only way real-time can be achieved is to operate at that absolute minimum execution time or very close to it. This may be reason enough to consider higher performance (albeit higher price) hardware alternatives. In general, however, it will be satisfactory to use any method (graphical, heuristic, or linear programming) for partitioning the software as long as it can be made efficient from a design standpoint (efficiency in this case is synonomous with the degree of automation).

With allocation choices made, it now remains only to schedule the distributed software modules on their elect microcomputers. Scheduling is simply the process of calling or activating the software modules assigned to a computer in the proper sequences and at the right frequencies such that the required time relationship between each pair of communicating modules, whether located on the same or separate computers, remains satisfied. This is a process that is also performed routinely for simulations targetted for a single minicomputer.

Implementation of a communications strategy (next section) will ensure that when each module is activated the data it needs as input from each of its source modules is available and valid.

INTERPROCESSOR COMMUNICATIONS

Except for the smallest of applications, the software computational load, as determined by the partitioning process, will mandate the use of multiple microcomputer units. The resulting communications between these units/nodes will take one of two forms - communications between parts of a single task (e.g., flight dynamics) distributed over multiple cpu's, or communication between separate tasks also resident on multiple cpu's (e.g., flight dynamics and cockpit analog inputs).

For a single task requiring no more than three or four single-board microcomputers a traditional shared memory configuration is still achievable and even viable for the same reasons that have made it a popular approach in minicomputer systems, i.e., the convenience with which minor software changes can be made, and the intuitive gratification of having the most significant software parameters of a system in a central location.

A variation of the shared memory approach is that of data broadcasting. A system employing this method would replicate the parameters of a traditional shared memory so that an identical version exists in each microcomputer cooperating to accomplish a single task. Whenever any of the participating microcomputers modifies the value of a shared parameter, the new value is transferred (sometimes referred to as reflected) to others through a special hardware interconnection. Although this approach is being pursued by simulation companies in specific cases (e.g., by Singer and Triad Microsystems) it has not as yet resulted in any commercially obtainable off-the-shelf products.

The alternative to shared memory is to use some form of message passing. The fundamental difference between the two is that message passing schemes avoid the potentially crippling bus contention created by multiple cpu's trying to access the same memory area simultaneously. This is accomplished by sending shared data from one memory to another, either in blocks or parameter by parameter piecemeal fashion, only to those locations where it is needed. Its not unlike the girls in your high school english class used to do. Writing messages to each other using the front black-board (shared memory) would have been unthinkable! For those preferring a less frivolous explanation, visualize a shared memory communication as always involving only one physical memory and a message passing communication as always involving at least two separate physical memories - the local memories of the sending and receiving microcomputers, and possibly of those acting solely as buffers (e.g., to route data from a memory on one card cage to a memory residing in another card cage).

A MULTI-MICROCOMPUTER CASE STUDY

In 1983, the Systems and Computer Technology Division of NTSC initiated an in-house project (Multiple Microcomputer System Architecture - MMCSA) to explore the capabilities of off-the-shelf microcomputer technology for training systems. The approach taken was to develop a representative trainer solely from microprocessor-based components. System development and integration followed the same basic outline discussed earlier in this paper.

Hardware Integration

While the application software was being analyzed and prepared for successful retargetting on multi-microcomputers (i.e. partitioned), hardware from numerous vendors was being evaluated, procured, configured and integrated into a programmable system (see figure 4). The required microcomputer boards, graphics cards and controller, analog/digital converters, mass storage interface board, and the actual two card set VME-bus backplane were all purchased from separate manufacturers. A simulation cockpit was also obtained and interfaced to the system. Graphics displays of instruments were then brought up on three CRT's mounted at the front of the cockpit in place of the instrument panel.

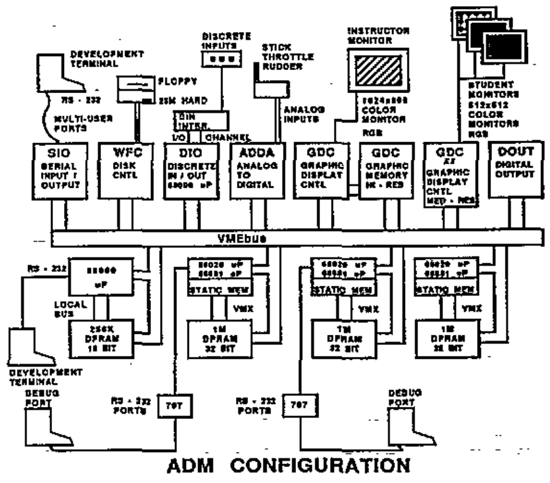

FIGURE 4
(Multi-microprocessor OFT)

Software Integration

define. The MMCSA project team first ported an existing FORTRAN operational flight trainer (OFT) simulation with acceptable modularity (~6000 lines of source code excluding comments bound in 75 subroutines) from a SEL 32/75 to the Systems and Computer Technology Division's VAX 11/780. Non-compatible language features were remediated and the simulation subroutines (modules) were all successfully compiled.

align. No acceptable commercially available data flow tools were found so, rather than perform this function manually, it was decided to write the necessary programs for determining (and organizing) module inputs and outputs, sources of inputs, destinations of outputs, and the existing time relationships between modules (see figure 5).

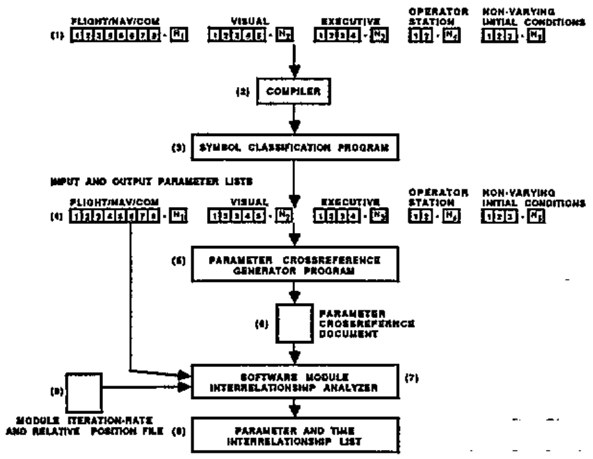
(1)
FLIGHT/NAV/COM
VISUAL
EXECUTIVE
OPERATOR STATION
NON-VARYING INITIAL CONDITIONS
(2) COMPILER
(3) SYMBOL CLASSIFICATION PROGRAM
INPUT AND OUTPUT PARAMETER LISTS
(4)
FLIGHT/NAV/COM
VISUAL
EXECUTIVE
OPERATOR STATION
NON-VARYING INITIAL CONDITIONS
(5) PARAMETER CROSSREFERENCE GENERATOR PROGRAM
(6) PARAMETER CROSSREFERENCE DOCUMENT
(7) SOFTWARE MODULE INTERRELATIONSHIP ANALYZER
(9) MODULE ITERATION-RATE AND RELATIVE POSITION FILE
(8) PARAMETER AND TIME INTERRELATIONSHIP LIST

FIGURE 5
(MMCSA Software Partitioning Process)

consign. A commercially available software package was found to assist in the process of assigning the simulation software modules to multiple microcomputers. This generic software package implemented a graphical critical path method analysis on a personal computer workstation. Given an approximate execution time, precedence constraints (which other modules must be executed before a given module), and a processor assignment for each module, the program calulated the time it would take to execute one pass of the simulation. Modules were moved from processor to processor until an acceptible distribution was arrived at, i.e., one in which modules were grouped in an intuitively pleasing manner and yet were still executable in real-time.

It was decided to adhere as closely as possible to the scheduling (the calling order and frequencies in which the software modules are executed) used by the system it was originally designed for. This reduced the number of intermodule timing discrepancies that had to be resolved for each separate partition (unique distribution of software modules to microcomputers). As discussed before, changing the exact time interval between the execution of any two given modules can change the characteristics of the simulation itself adversely if a fixed or bounded time constraint binding those modules is broken.

Interprocessor Communications

A communications strategy had to be chosen and implemented so that the data needed by each given module from modules on other microcomputers would be transferred and valid when referenced. Both message passing and shared memory schemes were conceived.

In the case of message passing, the locations of parameters needing to be transferred from one microcomputer to another had to be sequenced precisely in memory so that efficient transfer of data could occur. This was accomplished in standard FORTRAN alone through the ordering of parameters in the FORTRAN COMMON statement. For the shared memory approach, the exact sequence of parameter data in memory is arbitrary.

In both shared memory and message passing communication systems, a mechanism is needed to ensure that the execution of modules on one microcomputer does not race ahead of interdependent processing taking place on cooperating microcomputers. This means that one processor may need to suspend its execution until required data arrives from another microcomputer during the course of a real-time frame. Otherwise, the integrity of the simulation will be compromised.

A software tool was written to determine the exact parameters shared between microcomputers, organize these parameters for efficient transfer (in the case of message passing), and insert wait mechanisms where needed to control the execution of modules across microcomputer boundaries. Information generated by the programs shown in figure 5 was used directly as input to this final program.

Emulation

Before downloading the first partitioned simulation to the multimicrocomputer environment, an emulation facility was established on the VAX 11/780 using resident concurrent programming utilities. This was needed to verify that data would be transferred between microcomputers as expected and synchronization would be complete when the simulation was ultimately brought up on multiple microcomputers. The emulation was accomplished through the creation of a separate VAX process corresponding to each microcomputer being used by the partition. Modules were called and executed by each VAX process as they would on the microcomputers they were eventually to run on, except instead of communicating across a bus, data was transferred through VAX "mailboxes" and the processes were synchronized through software "event flags" rather than interrupts. In this way, a functioning simulation was assured before downloading to the target hardware. This left only problems unique to the VME-bus environment to be resolved with the debug facilities available at that level.

The End Thereof

At the time of this writing, the OFT simulation had been run on one, two, and three microcomputers using message passing communications. Current plans are to expand the hardware and develop additional partitions so that total system performance and interprocessor communication times can be compared for message passing and shared memory approaches in simulation subsystems having between two and six processors.

Initial expectations were that only one 32-bit single board microcomputer would be necessary to execute the OFT simulation at the required 33 millisecond per frame rate. The services of a full three microcomputers had been enlisted before this figure was finally met. The SEL 32/75, on which it originally ran, is rated at 0.7 MIPS (millions of instructions/sec), and required only 17 milliseconds (average) of the 33 millisecond frame to finish. On the other hand, three "2 MIP" machines (Motorola 68020's with 68881 floating point coprocessors) were needed to accomplish this same task in approximately 33 milliseconds - an order of magnitude devaluation of their rated performance relative to the SEL computer. It is being investigated whether this discrepancy is to be attributed to differences in benchmarking of the raw machines, or other factors, especially that of compiler efficiency.

A bright spot was the speed with which interprocessor communications were performed on the VME-bus. The number of words needing to be transferred for two, three, and four processor partitions were determined to be 550, 650, and 850, respectively. Using the basic message passing scheme, these parameters can be transferred across the 40 Mbyte/sec VME-bus in 1.3, 1.5, and 2.0 milliseconds, including software overhead. This is perceived as a highly tolerable figure for a typical 33 ms. OFT frame time.

CONCLUSIONS

Evolving general purpose 32-bit microcomputer packaging technology and standards are truly impressive. The competitive base of products based on these standards continues to grow rapidly. This does not mean, however, that this technology should be turned to naively by the simulation community. If the results of the ongoing study presented in this paper are any indication, it may be some time before simulation subsystems of large magnitudes (i.e., greater than 10,000 lines of code) can be successfully designed and maintained using multi-microcomputers. Presently, the lack of optimized FORTRAN and Ada compilers, and associated multiple processor software development tools represent the greatest void. As these products become available the power of microcomputer technology to meet simulation needs will increase dramatically. Several heads-up simulation houses, having recognized the potential of this technology, are beginning to fill the existing void with their own proprietary products. This will, of course, improve their own capability, but will unfortunately leave the promise of simulation subsystems integrated from completely non-proprietary components unfulfilled.

There is a need for continued commitment both to expressing the needs of the simulation community to the microcomputer industry, and to evaluating the river of products they continue to introduce. Through this process better and more maintainable microcomputer-based systems can be built by specifying that commercially available products be used first before turning to proprietary components.

BIBLIOGRAPHY

(1) W.W. Chu, L.J. Holloway, M.T. Lau, K. Efe, "Task Allocation in Distributed Data Processing", Computer, November 1980, pp. 57-69.

(2) R.P. Lee, R.R. Muntz, "On the Task Assignment Problem for Computer Networks", Proc. 10th Hawaii International Conference on System Sciences, Jan. 77, pp. 5-9.

(3) H.S. Stone, S.H. Bokhari, "Control of Distributed Processes", Computer, July 1978, pp. 97-106.

(4) Kemal Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems", Computer, June 1982, pp. 50-56.

(5) Michael B. Ash, "Functionally-Distributed, Microprocessor-Based Simulation: Once a Concept — Now a Fact", Proc. 7th Interservice/Industry Training Systems Conference, pp. 179-188.

(6) David Parkinson, "Distributed Processing for Complex Simulators", Proc. 7th Interservice/Industry Training Systems Conference, pp. 169-178.

ABOUT THE AUTHORS

Charles N. Pope is an Electronics Engineer with the Systems and Computer Technology Division of the Naval Training Systems Center and is responsible for evaluation of current and emerging computer systems. He received the B.S.E degree with a major in Electrical Engineering from the University of Central Florida, Orlando, Florida. He has researched issues relating to microcomputer simulator design at NTSC since January 1983 through the development, with the co-authors, of a multi-microcomputer trainer test bed.

David M. Kotick is an Electronics Design Engineer with the Systems and Computer Technology Division of the Naval Training Systems Center. His principal responsibilities include the integration, evaluation and design of microprocessor-based board-level products. He has received both a B.S.E. and M.S.E. in Electrical Engineering from the University of Central Florida. He has worked in the area of real-time multi-microprocessing at NTSC since 1983. Mr. Kotick is a member of the IEEE and IEEE Computer Society.

Mark W. Layton is a Computer Scientist with the Systems and Computer Technology Division of the Naval Training Systems Center. He has applied operating systems and graphics expertise to the area of microcomputer simulation design. He received his B.S in Computer Science from the University of Central Florida in 1983 and is currently pursuing his M.S. in Computer Science.