NASA Technical Memorandum 81731

NASA-TM-81731 19810013273

An Approach to Real-Time Simulation Using Parallel Processing

Richard A. Blech and Dale J. Arpasi
Lewis Research Center
Cleveland, Ohio

Prepared for the
1981 Summer Computer Simulation Conference
cosponsored by the Instrument Society of America
and the Society for Computer Simulation
Washington, D.C., July 15-17, 1981

NASA

AN APPROACH TO REAL-TIME SIMULATION

USING PARALLEL PROCESSING

Richard A. Blech and Dale J. Arpasi
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

ABSTRACT

Current applications of real-time simulations to the development of complex aircraft propulsion system controls have demonstrated the need for accurate, portable, and low-cost simulators. This paper presents a preliminary simulator design that uses a parallel computer organization to provide these features. The hardware and software for this prototype simulator are discussed. A detailed discussion of the inter-computer data transfer mechanism is also presented.

INTRODUCTION

The development of complex digital electronic controls for aircraft propulsion systems requires engine simulations that run in real-time and provide a high degree of accuracy and user interaction. In addition, the use of propulsion system simulations in many hardware-in-the-loop applications adds the further requirement that these simulations be implemented on dedicated, portable, and reliable hardware. Hybrid (analog-digital) computers have been used successfully in the past (refs. 1 and 2) to achieve real-time results. However, these computers are often difficult to program, lack portability, and are expensive. Many digital simulations performed on large mainframe computers such as an IBM 370/3033 can achieve real-time operation, but at the cost of portability. More recently, the advent of microcomputer technology has made compact, low cost, and portable computing power readily available. However, currently available, off-the-shelf microcomputers do not of themselves possess the necessary computational speeds to perform accurate real-time simulations of complex dynamic systems, such as aircraft propulsion systems. One approach to this problem is the use of microcomputers in a parallel arrangement. By using parallel processing, it is possible to retain the cost, size, and portability advantages of microcomputers and achieve the accuracy necessary for real-time simulation by increasing the number of computations per unit time.

In general, many schemes for parallel processing exit (refs. 3 and 4). One method of classifying these schemes (ref. 5) uses three categories:

1. SIMD (Single Instruction Multiple Data) - This category describes systems in which all processors execute the same instruction but with different data. Included in this category are array processors, such as ILLIAC IV, and associative processors.
2. MISD (Multiple Instruction Single Data) - In this computer organization, a single data stream is operated on by several instruction streams. A pipelined machine would be one example of this structure.
3. MIMD (Multiple Instruction Multiple Data) - Here, several data streams and instruction streams exist concurrently. Data flow machines fall into this category.

Of the three parallel processor organization described, the MIMD machine seems best suited for this simulation problem. No restrictions are placed on the problems' structure, as if often the case with array processors and other SIMD organizations. The MIMD organization also offers the most potential for exploitation of inherent parallelism in simulations.

MIMD architectures may be categorized as multiprocessor systems and distributed systems (ref. 6). The multiprocessor system uses an operating system to dynamically allocate tasks as they are received. One example is given in reference 7. These systems are advantageous when the computations contain much inherent parallelism. In the distributed system, the computers perform dedicated functions. Reference 8 presents a conceptual study of such a system. The advantage of this approach is that it permits simplified program development and debugging of individual program segments.

The design approach selected at the NASA Lewis Research Center is similar to that described in reference 8. This selection was made because transfer between processors is limited to data transfer only, thereby simplifying the hardware design and support software development. The potential value of the multiprocessor approach is also recognized and a grant has been awarded to UCLA to persue application of these techniques to real-time propulsion system simulation.

This paper presents the design of prototype simulator hardware being built at the NASA Lewis Research Center. The transfer mechanism between computers, as well as the approach to developing simulator software and operational software requirements are also discussed. It should be noted that, due to the exploratory nature of this work, it is expected that some of the design details presented will eventually be modified.

PROTOTYPE SIMULATOR DESIGN

The basic structure of the simulator is shown in figure 1. The core of the system consists of a transfer controller which synchronizes N (up to 10) 16-bit processing elements (PE's) on a high-speed data transfer bus. All but two (N-2) of the PE's perform simulation computations. One of the remaining PE's is of the same architecture but dedicated to input/output functions. This processor is called the I/O Processor (IOP). The last PE is a special purpose processor to link low-speed, operator-type functions with the high-speed simulator core. This PE is termed the RTX or Real-Time Extension of the Front-End Processor (FEP). The FEP provides an operators' interface as well as handling of peripheral communications and other simulator overhead, such as downloading of programs to the simulator PE's.

The simulator operation is separated into two basic cycles - a compute cycle and a transfer cycle. During the compute cycle, each PE performs the numerical computations for a predefined part of the simulation task. Upon completion of these computations, the PE sets a transfer flag to indicate that it is ready to enter a transfer cycle. The transfer controller initiates a transfer cycle when all PE's have set their transfer flags.

Transfer Controller - A functional diagram of the transfer controller is shown in figure 2. The transfer controller issues commands to all PE's from 8000 words of 32-bit Random Access Memory called a transfer map. The transfer map is cycled through by an address register/counter. When the start condition (all PE transfer flags set) is detected, the control logic automatically increments the address register every 100 nanoseconds. At the same time, the pipeline register is clocked, outputting the command word fetched at the previous address. This process continues until a stop condition is reached. The stop condition is programmed in the transfer map and detected by the control logic. When a stop is reached, the control logic resets the address register to its initial value. The transfer controller logic is also flexible enough to allow different start conditions and cycling through several transfer map loops, rather than the single loop operation previously described.

The transfer map command word consists of 32 bits and is segmented into four fields. The first is a field of 10 transfer destination (TDES) bits. There is one TDES bit for each of the PE's. When this bit is set, it signals the corresponding PE that it will be the destination for the next data transfer. The next field contains 10 transfer enable (TE) bits. Again, there is one TE bit for each of the PE's. If the TE bit is set, it enables the transfer mode. When the bit is reset, the transfer mode is disabled, and the PE resumes computation. When all TE bits are reset, the control logic terminates the transfer cycle. Thus, when a transfer cycle begins, all PE's that will be involved in the transfer are signalled via TE. During the cycle, any PE can be disabled from the transfer mode and resume computing while data transfer is still occurring between the remaining PE's. Four "this unit" (TU) bits contain the code which identifies the source PE during a data transfer. The 8-bit address following the TU bits is the address of the data

to be transferred from the source PE. Each PE, then, can be the source for up to 256 data words, and the destination for as many words as its memory will allow

The transfer controller also monitors several simulator timing parameters. A group of 16-bit programmable timers is used to accomplish this task. Ten of the timers measure each PE's compute time during an update interval. An update interval consists of one compute cycle and one transfer cycle. An update interval timer sets a limit on the length of an update interval. If this timer should time-out before an update interval is completed, an interrupt is issued to the FEP. Another timer, which sets a time limit on the computer cycle, sends an interrupt to the RTX if a time-out should occur. Finally, timers are used to generate interrupts for the IOP to inform it that data input or output should now occur. The specific details on the handling of these interrupts are given later in the FEP, RTX, and IOP discussions as well as the simulator software discussion.

The transfer map, address register/counter and all timers are programmable through the FEP. The timer contents, as well as the transfer map, can be read by the FEP. The transfer controller hardware meets the high-speed data transfer requirement through the use of pipelining, 45 nanosecond memory and Schottky TTL logic.

Each PE has 32k bytes (or 16k words) of Random Access Memory for program and data storage. During a compute cycle, all of this memory is accessible by the PE. During a transfer cycle, control of the memory is passed to the transfer controller. The memory configuration during a transfer cycle is shown in figure 3. The first 256 words of memory are available for use as source data. The remaining memory is available as storage for destination data. The source memory receives the 8-bit source address from the transfer controller via the transfer address bus. The source data is then latched at the source memory output. The input to the destination memory is latched from the 16-bit high-speed data-transfer bus. The local destination address register determines where received data will be stored. This register is initialized by the PE before a transfer cycle occurs, and automatically incremented after receiving each data word. Pipelining is used to maximize speed. One cycle is needed to initialize the pipeline and allow the PE's to clock in the first command word from the transfer controller. If the PE has been designated a source by the TU code, it will gate the source data onto the transfer bus during the next cycle. At the same time, all PE's designated as destinations prepare to accept data by updating their local address registers. The destinations then latch the incoming data, increment the local address, and prepare for the next cycle by clocking in a new command word. This process repeats for as many data transfers as programmed in the transfer map. Upon completion of the transfer cycle (all TE's reset), 200 nanoseconds is required to clear the pipeline. Thus, after 100 nanoseconds initialization, a data transfer from a source processor to as many as nine destination processors can occur every 100 nanoseconds.

Processing Element - Several factors must be considered when selecting a design approach to be used for a simulator PE. Speed is one of the most obvious requirements. Also, the PE instruction set has to be powerful enough to allow efficient programming of simulation algorithms. Cost, size and software support are other factors to be considered. Two design approaches are being investigated to provide maximum flexibility in optimizing these factors. The first is a custom SSI/MSI logic design tailored to the requirements of real-time simulation. The second is a design based on the 8MHZ Motorola MC68000 16-bit microprocessor.

The custom SSI/MSI design is microprogrammable, allowing for experimentation with different instruction sets. The projected cycle time is 133 nanoseconds, with many instructions taking only one cycle. The hardware is optimized for interprocessor transfer of data. This higher-risk, higher-payoff design approach is being undertaken in parallel with the lower-risk, off-the-shelf technology approach. Although the MC68000 instruction set is fixed, with most instructions taking several clock cycles, significant advantages exist in software support, size, and cost. Another advantage lies in the possibility of future upgrades in compatible hardware and software. Both PE designs will be hardware-compatible with the other simulator components, and each PE design will have the same memory configuration as outlined in the transfer controller description.

Operator Interface - Operator control over the simulator is accomplished via the FEP and RTX (see fig. 1). Such functions as simulator programming, mode control, operator advisories, and commands are provided. Both the FEP and the RTX designs are based on the Motorola MC68000 microprocessor.

The FEP handles the peripheral communications for the simulator (CRT, keyboard, floppy disk, etc.). There is also a host computer interface which allows uplinking and downlinking of data to/from the host. In our case, the host is an IBM 370/3033. The simulator operating system resides in the FEP. This is described in more detail in the simulation software discussion.

To allow the operator to dynamically access and process simulator data without slowing down the simulator core system, the RTX is attached to both the high-speed data-transfer bus, and the FEP I/O bus. Thus, the RTX has access to data within the high-speed core of the simulator, and can manipulate this data as commanded by the FEP. The RTX is alerted to an upcoming transfer cycle by an interrupt from the transfer controller. The FEP is informed by the transfer controller of any violations of the update interval (computer time + transfer time) by another interrupt. The operator can structure the software system to take appropriate action in handling both interrupting conditions.

Communication between the FEP and RTX is maintained by an interrupt-driven structure. FEP requests for actions involving the simulator high-speed core are initiated via an interrupt to the RTX. Likewise, advisories from the RTX to the FEP are accomplished by interrupts. In both cases, vectored interrupts are used. Up to 192 user-defined vectors are available in both the FEP and RTX. The FEP can also be interrupted by other conditions, such as overflow error, compute time violation, etc. Any of these interrupts can be used to halt the simulator and allow the FEP to examine the simulator status.

The operator interface, therefore, provides a users' link to the high-speed simulator to allow programming, debugging, and interactive execution of simulator software.

Input/Output Interface - The I/O interface for the simulator consists of the IOP, D/A convertors and A/D convertors. Typically, analog signals generated by the simulator would be monitored by a control system while analog signals, representing control system responses would be input to the simulator.

The IOP has the task of programming the DAC's and ADC's, and of transferring data to or from these devices. The IOP is tied to the high-speed transfer bus in the same manner as all other PE's. Synchronization of I/O operations is maintained through the use of interrupts. These interrupts are generated by progarammable timers in the transfer controller. Thus, the time interval between interrupts is selectable by the operator. The IOP inputs data during a pre-specified interval before a transfer cycle occurs, and outputs data at the beginning of a compute cycle and at a selectable sub-intervals during a computer cycle.

PROTOTYPE SIMULATOR SOFTWARE

The software development effort for the simulator consists of two major tasks: (1) the development of real-time operating tools to provide on-line diagnostics, control, execution, and analysis and (2) the development of user support tools for off-line simulation software development and verification. A goal of this effort is to minimize the simulator specific knowledge required for development and application of real-time simulations. This will be accomplished by using high order languages (HOL), user-friendly software development tools, and general engineering level commands. In order to avoid software obsolescence and to ease transportability between host computers, all tools will be written in PASCAL. Re-hosting will require recompilation on a resident PASCAL compiler and minor modifications to the tools to allow interfacing to the host operating system.

A similar software development effort is underway at the NASA Langley Research Center. The LaRC Multipurpose User-Oriented Software Technology (MUST) program (ref. 9) consists of a number of closely related software development efforts to: (1) decrease the cost and increase the reliability of software for space and aircraft digital flight computers and (2) provide an integrated software support system for aerospace embedded computer systems. Many of the MUST objectives are common to the simulator effort in that both are concerned with the development of real-time software. Because of the significant accomplishments at Langley, technology transfer is expected to reduce simulator software development effort and cost.

Operating System - The simulator operating system must combine standard management functions with unique simulation oriented operations. Simulator and simulation management functions should stress user convenience and are implemented on the FEP. Real-time data processing functions, while compiled on the FEP, are loaded into the RTX for real-time execution. RTX function execution is controlled and monitored via priority interrupts. Operator/FEP-initiated interrupts enable and disable real-time functions. RTX-initiated interrupts provide operator advisories, simulator control, and bookkeeping services.

A simplified diagram of the FEP program structure is shown in figure 4. An off-the-shelf disk operating system provides the basic I/O and file management functions. These are supplemented with simulator-specific I/O and interrupt handling rou-

tines to provide operating system interface to the peripherals and the simulator. The simulation data base provides descriptive information necessary to permit high-level operator specification of variables, programs, and IOP functions as command arguments. The interrupt data base is used to schedule the response to priority interrupts according to operational requirements. The remaining elements include: (1) the command processor, for invocation of operator commands, (2) four categories of operator command routines, (3) the interrupt processor, to provide pre-specified interrupt responses, and (4) four categories of interrupt responses.

The four categories of command routines are: data base management, RTX real-time file management, simulator/simulation control, and interrupt programming. Data base management consists of: (1) editors for all elements of the data base, (2) a compiler for formulating the RTX instruction file, and (3) read and write routines to effect program transfers between the data base, the simulator, and a dynamic data storage device. RTX real-time file management provides a means for enabling and disabling real-time command execution on the RTX. The simulator/simulation control provides standard computer control functions (RUN,HALT,SINGLE-CYCLE, etc.) in addition to simulation control functions (RESET,HOLD,OPERATE).

Interrupt programming routines allow the operator to specify the FEP response to priority interrupts initiated by the RTX through modification of the interrupt data base. The interrupt data base governs the execution of the interrupt processor. As shown in figure 4, the basic responses to RTX interrupts are combinations of operator advisories, simulator control functions, RTX command control functions, and priority data transfer functions. combined with a compatible RTX executive program, this interrupt structure provides an effective means for the operating system to meet simulation-specific needs.

Combinations of real-time instructions and proper uitilization of the interrupt structure should provide sufficient capability for general purpose real-time simulator operation. All program preparation could be done off-line using support software on a host computer to formulate the data base. Modification and extension of the data could then be accomplished on the FEP. This preliminary design forms the basis for a baseline simulator software package that can be revised as appropriate to support particular simulation needs.

Simulation Development - Figure 5 illustrates a general approach to simulation software development. The simulation formulator accepts an engineering statement of the system to be simulated and provides scaled equations, a definition of the computational split, variable transfer requirements, and other pertinent data to describe the simulation in terms of operator-selected nomenclature. The formulator will necessarily be specific to the selected computational approach. Initial development of this tool for the simulation of turbofan engines will be based on previous LeRC experience in developing of generalized turbofan simulations (refs. 10 and 11).

High order language (HOL) specification of the program for each PE follows the formulation. Primary prerequisites to the selection of a simulator HOL are: (1) easy representation of non-linear ordinary differential equations, (2) support of optimized functions such as integration and function generation, and (3) self documenting features. The NASA standard language HAL/S (ref. 12) is being evaluated as a simulation HOL. It contains the necessary features and a subset version is heavily supported by LaRC in MUST.

Intermediate code, generated by the HOL translator, may be used for static and dynamic verification and testing of the HOL program.

Up to this point, simulation development is independent of the target simulator. Dependence is introduced by the target translator. Replacement of the target translator allows transportation of simulations between different processing element designs. The translator should provide time-optimized source modules. It is therefore, a critical element in the simulation development process and must be developed for efficient code translation.

A meta-assembler (ref. 13) can be targeted for the PE's to provide relocatable object modules for inclusion into the PE object library. These modules would then be linked and loaded into PE program files for inclusion in the FEP data base.

The transfer control programmer develops a transfer map source module to govern real-time data transfer between the PE's. The meta-assembler, targeted for the transfer control, provides an absolute transfer program file for inclusion into the FEP data base.

The PE and transfer control modules form one element of the data base for the simulator operating system. The second element of the data base consists of descriptors which define: (1) program and subprogram names, locations, and entry points, (2) variable names, locations, and scale factors with reserved storage for measured values, (3) array names and names of the array elements, and (4) input/output specifications relating variable names to real-time I/O channels. The data base is then downlinked to the simulator's front-end processor.

SIMULATOR STATUS

The simulator hardware and software previously discussed are undergoing exploratory development at the NASA Lewis Research Center. Prototype hardware such as the SSI/MSI processing elements and the transfer controller are currently being fabricated and tested. Integration and testing of all hardware componments for the simulator system is expected to be completed by late 1982. At that time, it is planned that all user support software will be complete. This will allow running of applications programs, including a non-linear simulation of a turbofan jet propulsion system.

CONCLUDING REMARKS

An approach to real-time simulation using parallel processors has been presented. The prototype hardware is currently being fabricated and tested at the NASA Lewis Research Center. User support software being developed at NASA Langley will be used to aid in the development of simulator operational and applications software. Initially, the prototype simulator will be used as a research tool for investigating various parallel processing hardware and software configurations. Knowledge obtained from this research will be applied toward the development of airbreathing propulsion system simulations.

Currently-available computer and digital electronic technologies have made simulations implemented on dedicated, low-cost, and portable hardware possible. It is expected that future improvments in this technology will have a significant impact on the development of simulator hardware and software.

REFERENCES

1. Szuch, J. R. and Bruton, W. M.; 1974, "Real Time Simulation of the TF30-P-3 Turbofan Engine Using a Hybrid Computer", NASA TM X-3106.
2. Szuch, J. R., Seldner, K., and Cwynar, D. S.; 1977, "Development and Verification of Real-Time Hybrid Computer Simulation of F100-PW-100(3) Turbofan Engine", NASA TP 1034.
3. Enslow, P. H.; March 1977, "Multiprocessor Organization - A Survey", Computing Surveys, Vol. 9, No. 1, pp. 103-129.
4. Thurber, K. J. and Wald, L. D.; December 1975, "Associative and Parallel Processors", Computing Surveys, Vol. 7, No. 4, pp. 215-255.
5. Flynn, M. J.; September 1972, "Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers, Vol. C-21, No. 9, pp. 948-960.
6. Weissberger, A. J.; June 1977, "Analysis of Multiple-Microprocessor System Architectures", Computer Design, Vol. 16, No. 6, pp. 151-163.
7. Halin, H. J., Buhrer, R., Hälg, W., Benz, H., Bron, B., Brundïers, H. J., Isacson, A., and Tadian, M.; October 1980, "The ETH Multiprocessor Project: Parallel Simulation of Continuous Systems", Simulation, p. 109.
8. O'Grady, E. P.; February 1980, "A Communication Mechanism for Multiprocessor Simulation Systems", Simulation, p. 39.
9. Straeter, T. A., Fopudriat, E. C., and Will, R. W.; 1977, "MUST - An Integrated System of Support Tools for Research Flight Software Engineering", AIAA Computers in Aerospace Conference, American Institute of Aeronautics and Astronautics, Inc., New York, pp. 442-446. AIAA Paper 77-1459.
10. Szuch, J. R.; 1974, "HYDES, A Generalized Hybrid Computer Program for Studying Turbojet or Turbofan Engine Dynamics", NASA TM X-3014.
11. Szuch, J. R., Krosel, S. M., and Bruton, W. M.; 1981, "An Automated Procedure for Developing Hybrid Computer Simulations of Turbofan Engines", NASA TM-81605.
12. Ryer, M J.; 1979, "Programming in HAL/S", Intermetrics, Inc., Long Beach, CA, 2nd ed., NASA CR-162501.
13. McDonnell Douglas Astronautics Company, West Huntington Beach, CA, 1979, "Meta Assembler User's Manual", MDC G5876, Rev.

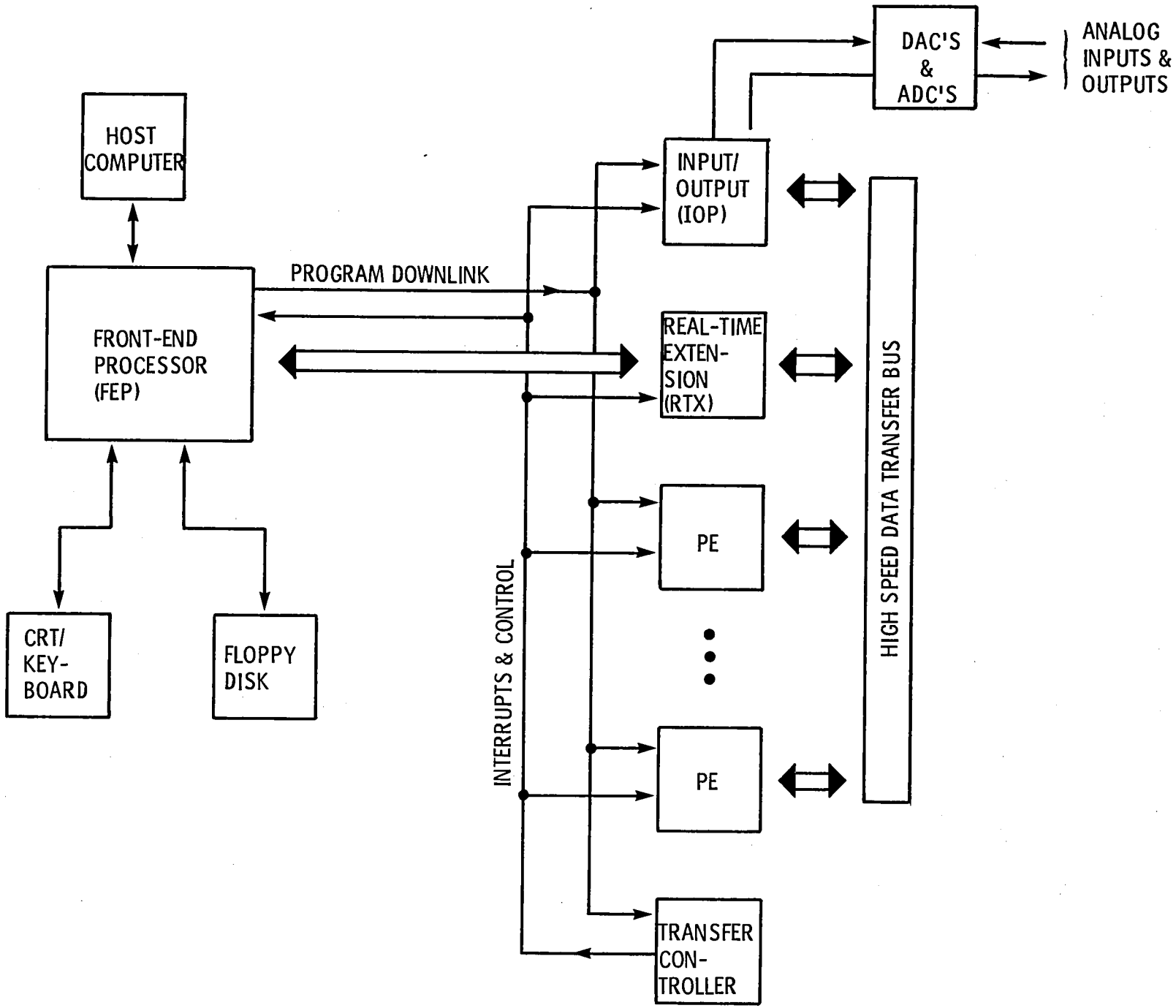

Figure 1. - Basic simulator structure.

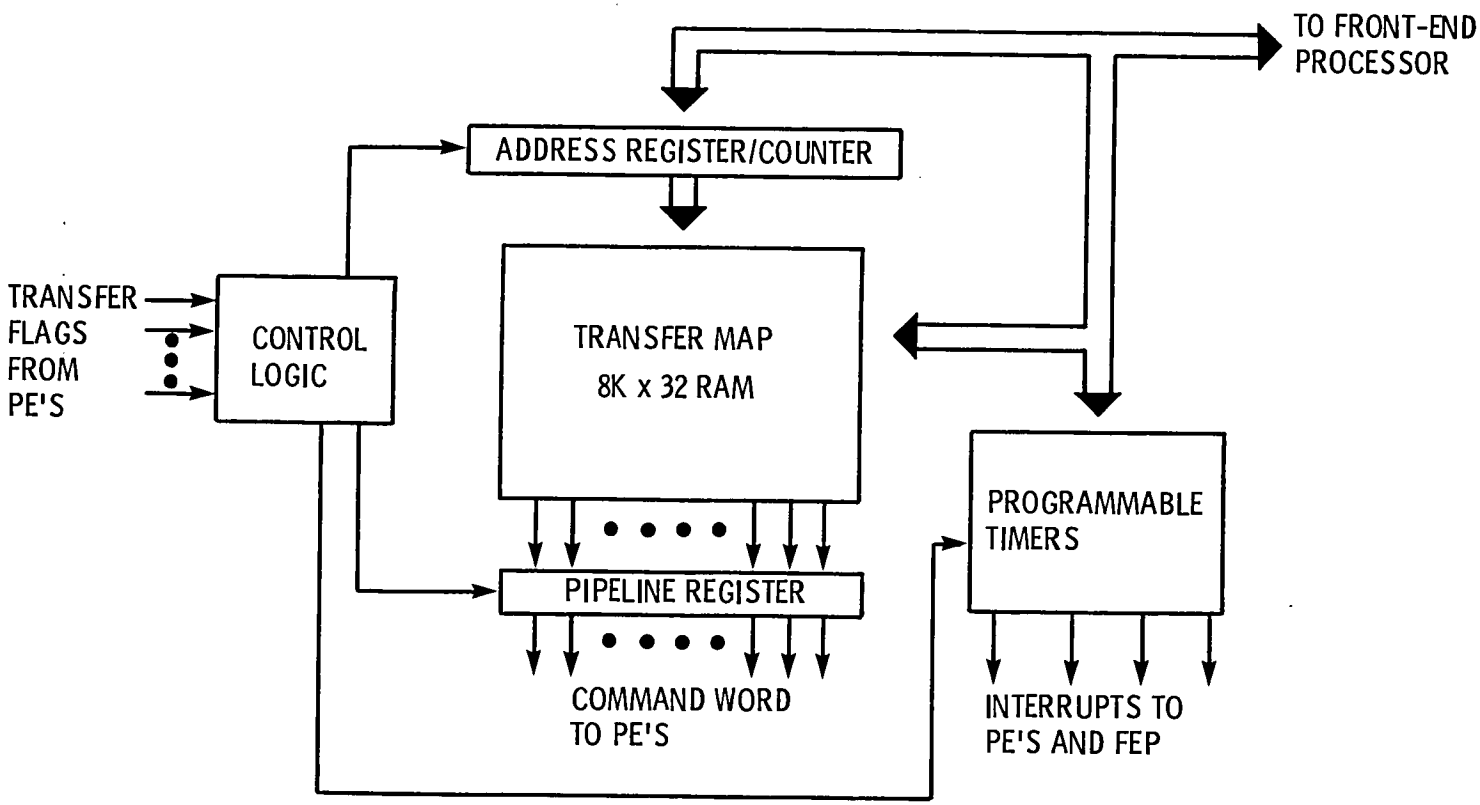

Figure 2. - Transfer controller block diagram.

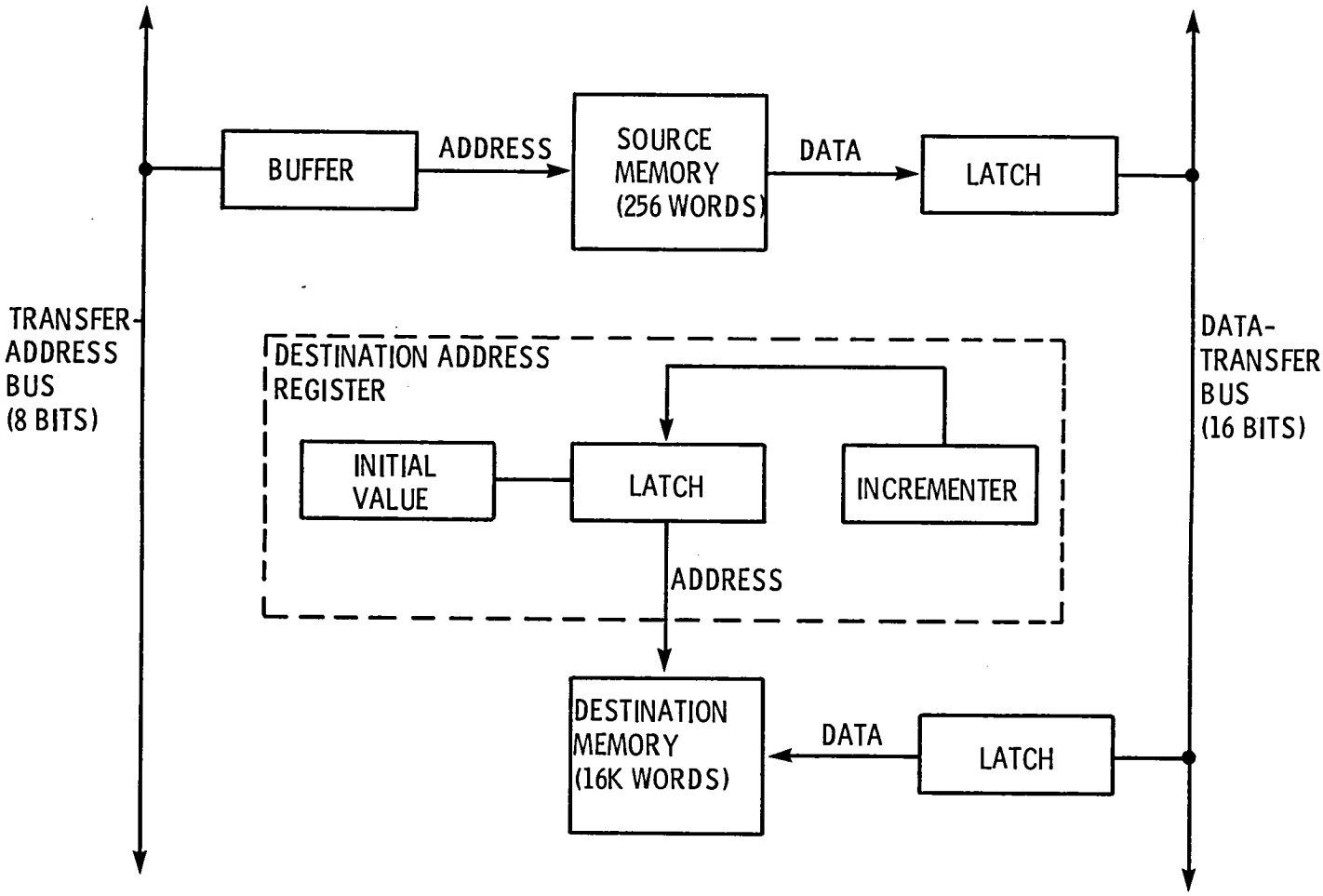

Figure 3. - Processing element memory during a transfer cycle.

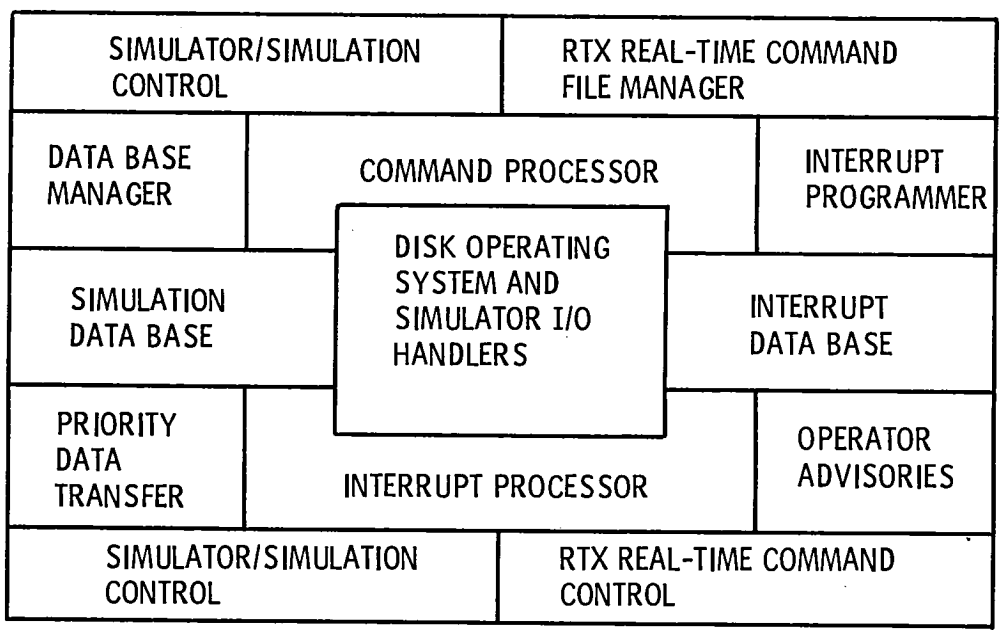

Figure 4. - Front-end processor program structure.

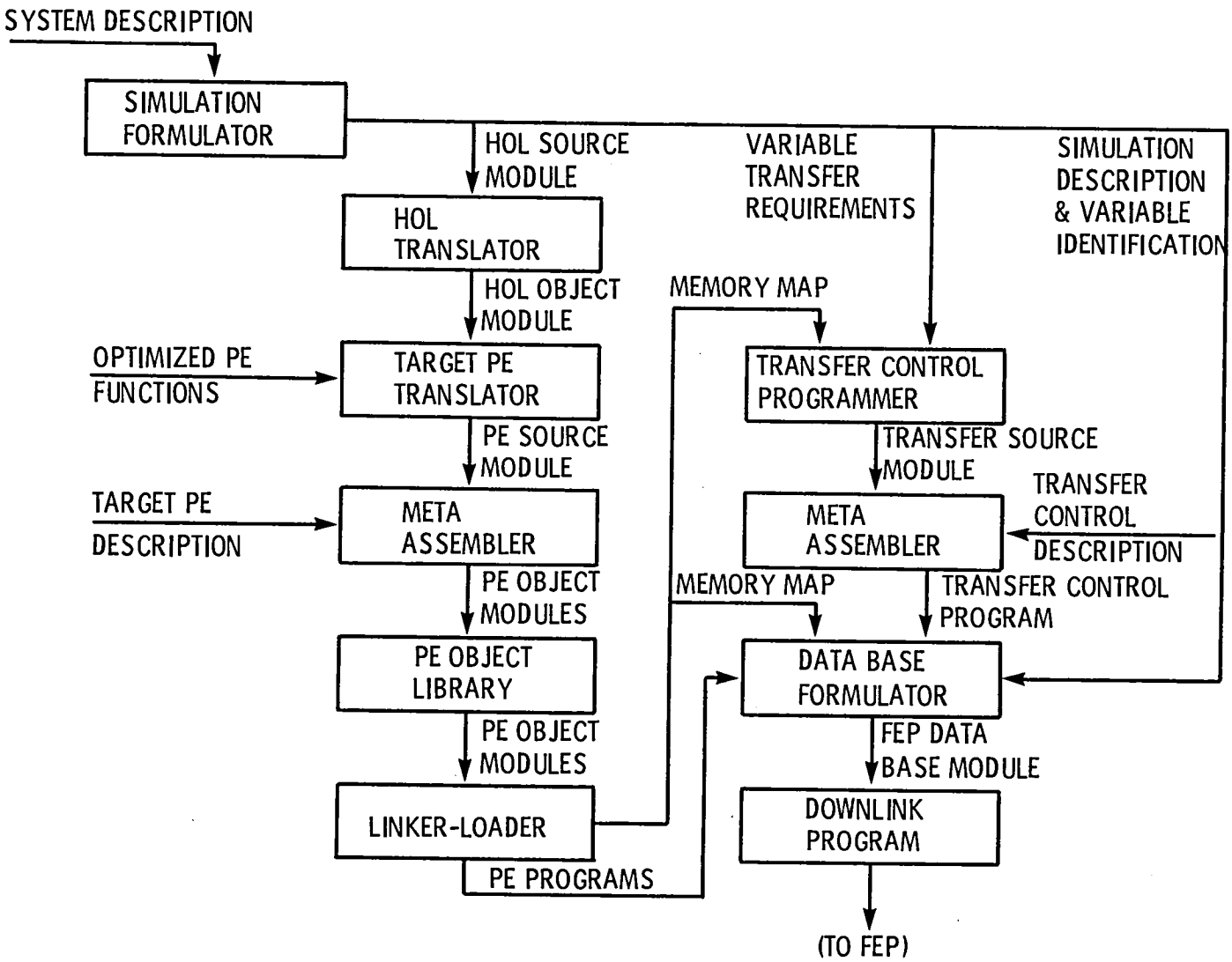

Figure 5. - Simulation software development.

1. Report No. NASA TM-81731	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle AN APPROACH TO REAL-TIME SIMULATION USING PARALLEL PROCESSING		5. Report Date
		6. Performing Organization Code 505-32-6B
7. Author(s) Richard A. Blech and Dale J. Arpasi		8. Performing Organization Report No. E-787
		10. Work Unit No.
9. Performing Organization Name and Address National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		14. Sponsoring Agency Code

15. Supplementary Notes
Prepared for the 1981 Summer Computer Simulation Conference cosponsored by the Instrument Society of America and the Society for Computer Simulation, Washington, D.C., July 15-17, 1981.

16. Abstract

Current applications of real-time simulations to the development of complex aircraft propulsion system controls have demonstrated the need for accurate, portable, and low-cost simulators. This paper presents a preliminary simulator design that uses a parallel computer organization to provide these features. The hardware and software for this prototype simulator are discussed. A detailed discussion of the inter-computer data transfer mechanism is also presented.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Real-time simulation Parallel processing Microcomputers	Unclassified - unlimited STAR Category 62

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

* For sale by the National Technical Information Service, Springfield, Virginia 22161

National Aeronautics and
Space Administration

Washington, D.C.
20546

Official Business
Penalty for Private Use, $300

SPECIAL FOURTH CLASS MAIL
BOOK

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

NASA

POSTMASTER: If Undeliverable (Section 158
Postal Manual) Do Not Return