NASA Technical Memorandum 83805

Hardware for a Real-Time Multiprocessor Simulator

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

Prepared for the
1985 SCS Multiconference "Distributed Simulation"
sponsored by the Society for Computer Simulation
San Diego, California, January 24–26, 1985

NASA

HARDWARE FOR A REAL-TIME MULTIPROCESSOR SIMULATOR

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

ABSTRACT

This paper describes the hardware for a real-time multiprocessor simulator (RTMPS) developed at the NASA Lewis Research Center. The RTMPS is a multiple-microprocessor system used to investigate the application of parallel-processing concepts to real-time simulation. It is designed to provide flexible data exchange paths between processors by using off-the-shelf microcomputer boards and minimal customized interfacing. A dedicated operator interface allows easy setup of the simulator and quick interpreting of simulation data.

Simulations for the RTMPS are coded in a NASA-designed real-time multiprocessor language (RTMPL). This language is high level and geared to the multiprocessor environment. A real-time multiprocessor operating system (RTMPOS) has also been developed that provides a user-friendly operator interface.

The RTMPS and supporting software are currently operational and are being evaluated at Lewis. The results of this evaluation will be used to specify the design of an optimized parallel-processing system for real-time simulation of dynamic systems.

INTRODUCTION

Hardware-in-the-loop and man-in-the-loop simulations, which require real-time performance, provide many cost- and time-saving benefits. This is particularly true for jet aircraft systems, where ground and flight testing are costly. For example, piloted simulators are used as a convenient, low-cost method to evaluate system design changes and their effect on pilot workload (1). Real-time simulations of aircraft propulsion systems are extensively used to evaluate new control system designs (2). More recently, there have been proposals of fault-tolerant propulsion controls, where an airborne engine simulation would be used to detect sensor failures and to provide a simulated signal for the faulty sensor (3).

These applications of real-time simulations require simulations that are portable and cost effective. Currently available mainframe computers and hybrid (analog/digital) computers can achieve real-time speeds but are not portable or cost effective for these applications. Microcomputers are small and inexpensive but lack sufficient computational power. However, by operating several microcomputers in parallel, and through proper partitioning of the simulation problem, computational throughput can be increased while cost and portability benefits are maintained.

The real-time multiprocessor simulator (RTMPS) project at the NASA Lewis Research Center is aimed at developing multiprocessor hardware and software that satisfy the aforementioned design goals. To accomplish this task, an experimental multiprocessor system was constructed. The design is intended to provide as many interprocessor communications paths as possible by using off-the-shelf microcomputers and minimal customized interfacing. System firmware is used to coordinate data transfers, to provide extensive diagnostic capabilities, and to interface the multiple processors to a simulation-oriented operating system. This approach allows simulation software development tools to be tested and verified in the multiprocessor environment. It also provides a valuable facility for investigating various methods of partitioning simulations to run on multiple microcomputers.

This paper describes the multiprocessor research hardware that is currently operational. The RTMPS is being used as a test vehicle to aid in specifying hardware and software for a more optimized multiprocessor simulation system. These specifications can then be applied by using the most current microprocessor technology for special-purpose simulators, such as fault-tolerant control systems.

SYSTEM ARCHITECTURE

Several board-level computer systems are currently available to support multiprocessing in a single-bus environment. Intel's Multibus and Motorola's Versabus are examples of this (4,5). In these systems, there is typically a bus controller that arbitrates requests from potential bus users. The bus controller grants access to the bus according to some priority scheme. Only one processor or device can use the bus at any time. In such a multiprocessor system the bus can quickly become congested when frequent interprocessor communication is required. This is especially true in a highly interactive environment, where an operator may be continually accessing data generated on one or more of the processors in the system. One method of improving the bus throughput is by adding one or more additional communication paths to the system. This is the approach taken for the RTMPS.

The RTMPS uses a dual-bus architecture (fig. 1). The lower bus, called the interactive information bus (IBUS), provides a data path primarily for user input and output. The upper, or real-time, information bus (RBUS) provides a data path for transferring data between the simulation processors. A special device on the IBUS, called the front-end processor (FEP), handles all input and output between the operator and the RTMPS. The other processors on the IBUS (called IBP processors) perform tasks related to data transfer to and from the FEP. The IBP processors can also handle part of the real-time simulation calculations. Processors on the RBUS (called RBP's) normally perform the bulk of the simulation calculations. An IBP processor communicates with a RBP processor through a dual-port interface memory. The combination of a IBP processor, an RBP processor, and interface memory forms a channel. Interprocessor communications can

be viewed as occurring locally within a channel (via dual-port memory) or globally between channels (over the IBUS or RBUS). The number of channels can range from zero to the maximum allowed by the physical bus limitations. The first channel (channel 0) plays a special role. The RBP processor in channel 0, called the real-time controller, is responsible for maintaining synchronization between all RTMPS channels. It sets up and begins program execution and monitors the timing to ensure that all channels complete their calculations in a specified time. The real-time controller also handles analog input and output, interfacing the RTMPS to external hardware such as actuators or controllers. The IBP processor in channel 0 controls the operating mode (RUN, HOLD, or STOP) and provides real-time analysis functions as commanded by the FEP. Real-time analysis functions include data collection, event triggering, rate-of-change monitoring, and peak detection.

The RTMPS architecture provides a highly user-interactive environment. However, the architecture can provide higher performance at the cost of user interaction. In this case, both the IBUS and the RBUS would be used for interprocessor transfers related to simulation calculations. The FEP would serve only as a loader and initializer for the simulation code. This mode of operation would be useful for applications where a proven, dedicated real-time simulation is required and only minimal operator interaction is needed.

SYSTEM COMPONENTS

The various components of the RTMPS system (fig. 2) are mounted in the top halves of two relay racks. Two terminals are shown. One terminal is normally used to communicate with the multiprocessor system through the FEP. The other terminal is available for software development. The printer is used for listing programs and results. The various architectural components introduced previously are described in detail here.

Microcomputer Boards

The IBP and RBP processors are implemented by using Motorola VM02 microcomputer boards (6). These boards consist of an 8-MHz 68000 microprocessor, 128K of random-access memory, three programmable timers, a system bus interface, and an interrupt controller. The VM02 RAM is dual ported to the board's local bus and the external system bus. Therefore all of the 128K RAM is accessible by the local 68000 microprocessor or by any other microcomputer board on the system bus. Each board also has an input-output channel to allow communications with devices external to the board and the system bus. This input-output channel is used to interface the microcomputer board to a dual-port interface memory.

Interface Memory

The dual-port interface memory is the communications link between processors on the IBUS and those on the RBUS. The memory consists of a small amount of customized circuitry contained in a rack-mountable chassis (fig. 3). It is interfaced to an IBP processor and an RBP processor through each processor's input-output channel. The input-output channel is composed of address, data, and control lines that define a memory-mapped segment of the processor's address space. The input-output channel is specified (7) to be an asynchronous communications path. Thus, whenever a processor begins an input-output channel access cycle, it must wait for an acknowledgment from the channel before completing the cycle. The control and arbitration logic decodes memory accesses from both of the processors. Either processor can read from or write to the memory. However, only one processor can access memory at any one time. If simultaneous accesses (contention) occur, one processor is delayed while the other completes its cycle. Typical access times without contention a 1.3 μs for a read cycle and 1.5 μs for a write cycle.

There are three 1K blocks of memory. The address spaces occupied by each of the first two blocks are switch selectable by software (fig. 4). If the switch is off, memory block 1 occupies the address space 0 to 1023, and block 2 resides in locations 1024 to 2047. When the switch is on, memory block 1 is now located at addresses 1024 to 2047 and block 2 is at 0 to 1023. This feature is especially useful for a simulation problem where past values, such as the derivatives for an integration algorithm, must be retained. For example, assume a variable is assigned to location 1024 and its past value to location 0. By toggling the switch and then updating the variable at location 1024, the past value is automatically retained at location 0.

The firmware on each channel's RBP, which is described in detail later, controls the setting of the memory switch. It toggles the switch on each channel at every update interval. The update interval defines what is "real time" for the system. The simulation's dynamic equations, which are functions of time, are recalculated at every update interval by using a new value for time. Time is advanced by the increment defined by the update interval, and the memory switch is toggled.

The third and final block of memory is used to store flags, control parameters, and other information that synchronizes communication between the two processors sharing the memory. Two 8-bit latches within the interface memory are used to generate interrupts. One latch is assigned to each of the processors that share the memory. One bit from the RBP processor's latch is also used for the memory switch as previously described. This bit is controlled by the RBP processor firmware. Both processors have four interrupts, each of which is used to begin interprocessor communications within an RTMPS channel.

Real-Time Information Bus

The RBP processors physically reside in Versabus card cages. Thus data transfers that occur over the RBUS follow the Versabus specification. The specification allows for multiple bus masters and five levels of bus priority. Because all data transfers are asynchronous, a sending device requires an acknowledgment from a receiving device before the transfer is completed. In this arrangement, the bus controller must reside in the first slot of the card cage. The bus controller monitors requests for bus access and grants access according to priority level.

Referring again to figure 1, the first processor (located in channel 0) on the RBUS is designated as the real-time controller. This processor has the responsibility of synchronizing all of the RBP processors during RTMPS operation and also performs the analog input and output. Because board-level analog peripherals are not available for the Versabus, a Versabus-to-Multibus converter is used. The converter allows a Multibus card cage to act as an extension of the Versabus system. Thus the many

Multibus-compatible analog peripherals can be used with the Versabus. Another benefit of the Multibus extension is the ability to communicate with other Multibus board-level products, including microcomputer boards. This is a useful feature in the research being performed at Lewis. Future applications of RTMPS include evaluating experimental control systems. These control systems are typically implemented at Lewis by using Intel 8086-based Multibus microcomputers. With the Multibus extension of the Versabus the experimental control computer can be easily integrated into the RTMPS for testing. The RTMPS processors would perform engine simulation calculations, obtaining the control values they need from the control computer's dual-port memory (via Versabus-Multibus). In a similar manner the control computer would sample the various engine parameters it needs.

Interactive Bus

The IBUS is contained within a Motorola Exormacs development system. The chassis is a 15-slot Versabus, with several slots containing the development system processor board, disk controller, RAM, and communications controller. The development system's Versabus forms the IBUS, with the previously described board set constituting the FEP. The remaining card slots are used for the IBP processors in the RTMPS channels.

Although the Exormacs development system was designed for software development, it has many useful features that make it ideal for an FEP. The system communications controller and a terminal are used as the RTMPS operator's console. The disk controller, hard disk, and floppy disk systems provide storage for programs and data. The resident disk operating system, Versados, contains utilities and system routines that can be called by the RTMPOS operating system. Since the disk operating system is multiuser and multitasking, the RTMPS can be used simultaneously for software development and running real-time simulations.

FIRMWARE

Hardware/Software Interface

The heart of the RTMPS system is the firmware. Each IBP and RBP has its own distinct firmware. The firmware contains the code (in read-only memory) that initializes and synchronizes the system and coordinates the transfer of data between processors. It provides an interface between the RTMPS hardware and the system software. It also controls the flow of information to and from the analog hardware (i.e., DAC's and ADC's), as illustrated in figure 5. One of the software/firmware interfaces shown is to the real-time multiprocessor programming language (RTMPL) (8). This is a NASA-designed language that is used to program multiprocessor systems. Another interface is to the real-time multiprocessor operating system (RTMPOS) (9), which is a simulation-oriented operating system used on the RTMPS. RTMPOS was also developed by NASA. Detailed descriptions of these software packages are provided in the references, but a brief overview is given here.

The RTMPL language allows a user to describe simulation equations that have been partitioned to run on multiple processors in a high-level, structured manner. RTMPL acts as an assembly language programmer, translating the high-level simulation description into time-efficient, assembly language code for the processors. All required interprocessor communications (i.e., data transfers) are automatically established by the RTMPL translator. RTMPL simulations are self-documenting since the translator utility produces listings, error messages, warnings, and database files that can aid in the debugging and running of a simulation. The RTMPL utility is written in Pascal and runs on a Motorola Exormacs development system (which also serves as the RTMPS FEP). The RTMPL interface firmware implements data transfer between processors and issues simulation-generated advisories to RTMPOS.

The real-time multiprocessor operating system provides the RTMPS user with engineering-level, run-time operations such as loading and modifying of programs, simulator mode control, data handling, and run-time monitoring. The RTMPOS acts in conjunction with the FEP's manufacturer-supplied disk operating system, Versados. Versados supplies typical utilities such as file handling and input-output services. It also provides software development tools such as a text editor, an assembler, a linker, and a Pascal compiler. The RTMPOS software is programmed mainly in Pascal with some assembly language routines.

The RTMPOS interface firmware provides the primitive operations necessary for (1) information transfer between the FEP and the RTMPS processors, (2) simulation mode control, (3) sequencing and execution of simulation code segments, (4) execution of functions for data analysis, and (5) issuing of diagnostic advisories and status information.

The information transfer functions allow program loading, initialization, data display, and execution control by RTMPOS to be done by using "handshake" mechanisms. Flags are used to synchronize firmware and RTMPOS execution. RTMPOS sets a flag to begin a function and monitors it until it is reset by the firmware, an indication that the function has been performed.

Data Transfer

The transfer of data between processors during simulation execution is coordinated by firmware. The transfer requirements are established by RTMPL during translation of the simulation source code. For example, the source code for one processor may reference a variable that is calculated on another processor. RTMPL recognizes this situation and generates the necessary code to begin the transfer of the variable from the source processor to the destination processor. The firmware provides the services to perform the physical data movement and to maintain data currency within a simulation update interval. Data currency is maintained through the use of a currency flag. All external variables have a currency flag that alternates in value every update interval. A destination processor requiring an external variable must first verify that the variable is current by testing this flag. A source processor transferring a variable to a destination processor must send the currency flag with the variable. The use of the currency flag allows data to be transferred asynchronously (i.e., at any time) during an update interval. The ability to transfer data asynchronously provides the maximum flexibility for partitioning of simulation code to run on multiple processors. A more detailed discussion of asynchronous and synchronous (where all transfers occur at a fixed interval) data transfer is given in McLaughlin (10).

Mode Control

Three major simulation modes are supported by the firmware: STOP, RUN, and HOLD. In the STOP mode all simulation processors are available for program loading and initialization. In the RUN mode the various segments of the simulation code are executed repetitively on the basis of the cyclic timeouts of a programmable timer. The timeout generates an interrupt in each of the RTMPS processors. The period between interrupts defines the simulation update interval. It is the responsibility of the real-time controller to verify that all channels have completed their calculations before the end of the update interval. During the computation interval each processor performs calculations for its one segment of the simulation. The RUN mode is illustrated in figure 6 for an example simulation involving three variables (VAR1, VAR2, and VAR3).

In the example, variable VAR1 is computed on IBP 1 and VAR2 and VAR3 on RBP 2. Variable VAR1 must be transferred to RBP 1 in order for that processor to complete its calculations. Similarly, VAR2 must be transferred to IBP 1 and VAR3 to IBP 2. The wait cycles shown in figure 6 represent testing of a variable's currency. When each RBP has completed its calculations, it waits until the associated IBP processor is finished. The RBP then signals the real-time controller, via interrupt, of that channel's completion. Before this interrupt the real-time controller has been performing simulation calculations, supporting analysis functions, and outputting information to the DAC's. When all chanrels have completed their calculations, the real-time controller inputs ADC information for the next update interval. If all channels have not completed their calculations before the next update interval, an interrupt is issued to the operating system. RTMPOS then advises the user of a failure. In addition, the firmware maintains a watchdog timer for each processor that, if not reset periodically, interrupts RTMPOS to issue an advisory.

The HOLD mode is similar to the RUN mode in that the simulation is repetitively executed. In HOLD, however, user-specified variables are held constant and the computation is recycled upon completion and is not timer driven. The watchdog timer does operate in HOLD. Before each computation is executed, latches (associated with each variable whose value is to be held) are set by the firmware. During computation in this mode the latches are tested when a variable value has been computed and, if set, the new value is replaced by the old one.

Analysis and Advisories

The firmware serves the following RTMPOS-selectable analytical functions: data sampling, peak detection, and rate-of-change detection. These services are set up in STOP and may be activated in either RUN or HOLD. The real-time analysis processor in channel 0 executes the analysis firmware (if an analytical task is selected). This allows the other processors to perform simulation calculations uninterrupted.

The firmware provides for advisories to the user through RTMPOS. These services are of two types: system advisories indicating RTMPS status, and user advisories indicating simulation status. System advisories are used for timeouts, hardware errors, and diagnostic information. User advisories are programmed in RTMPL and allow the user to obtain run-time simulation information. The firmware issues advisories by interrupting the FEP to activate service tasks that issue the advisory.

The firmware also performs power-up and initialization functions for each processor. At power-up the processor interrupt vector table is set up. Program memory is initially cleared and a channel interrupt test is performed. This test checks the interrupt mechanism in the dual-port interface memory, which is critical to RBP-IBP processor communications. Once the interrupt test is completed, each channel executes a test of the interface memory. This is a dual-processor version of the sliding-ones-and-sliding-zeros test described by Milner (11). If a channel fails to pass these tests, RTMOS alerts the operator. After the power-up tests each channel then enters the STOP mode to await program loading.

To accomplish these functions, the firmware requires about 4K bytes of ROM on each processor. As development proceeds in RTMPS, more features will be added to enhance the interactive nature of the operating system.

CONCLUDING REMARKS

The overall performance of the RTMPS is tied directly to processor speed and the level of parallelism in the simulation code. In the existing RTMPS configuration an increase in processor speed will decrease the minimum update interval achievable (i.e., more calculations can be done during the update interval). The update interval can also be decreased by more efficient partitioning of the simulation code. There is, however, a "critical path," or a limit to how far the code can be segmented.

Advances in microelectronics are expected to affect the RTMPS hardware. For example, the next generation of microcomputer boards, which can be "plugged" directly into the RTMPS system, is expected to be three to four times faster than existing boards. Hardware performance can also be improved by using bit-slice computer technology for the processors, such as the AMD 2900/29000 family. This approach must be evaluated, however, from the standpoint of size, cost, and software development, since a penalty is paid in each of these areas for the increase in speed.

The RTMPS hardware described herein is currently being used to investigate the application of parallel processing to real-time simulation of dynamics systems. The RTMPL language and the RTMPOS operating system are operational and in the process of optimization. Partitioning algorithms are also being evaluated on RTMPS. A benchmark simulation of a small turboshaft helicopter engine has been partitioned and is operational on the system. A future application is the real-time simulation of a wind tunnel facility. The simulation will be used to evaluate control system hardware and for operator training. As the evaluation process proceeds, it is anticipated that improvements to the RTMPS hardware and software will result.

REFERENCES

1. Nieuwenhuijse, A.W.; and Franklin, J.A.: "A Simulator Investigation of Engine Failure Compensation for Powered Lift STOL Aircraft." NASA TM X-62363, 1974.

2. Szuch, J.R.; Skira, C.; and Soeder, J.F.: "Evaluation of an F100 Multivariable Control Using a Real-Time Engine Simulation." NASA TM X-734648, 1977.

3. Merrill, W.C.: "Sensor Failure Detection for Jet Engines Using Analytical Redundancy." NASA TM-83695, 1984.

4. Intel Corp: "Intel Multibus Specification." Intel Manual 9800683, 1978.

5. Motorola, Inc.: "Versabus Specification Manual." Motorola Manual M68KVBS(D4), 1981.

6. Motorola, Inc.: "VERSA Module Monoboard Microcomputer User's Guide." Motorola Manual M68KVM02(D1), 1982.

7. Motorola, Inc.: "Input/Output Channel Specification Manual." Motorola Manual M68RIOCS(D2), 1982.

8. Arpasi, D.J.: "RTMPL - A Structured Programming and Documentation Utility for Real-Time Multiprocessor Simulations." NASA TM-83606, 1984.

9. Cole, G.L.: "Operating System for a Real-Time Multiprocessor Propulsion System Simulator." NASA TM-83605, 1984.

10. McLaughlin, P.W.: "Parallel Processor Engine Model Final Report." NASA CR-174641, 1984.

11. Milner, E.J.: "A Generalized Memory Test Algorithm." NASA TM-82874, 1982.

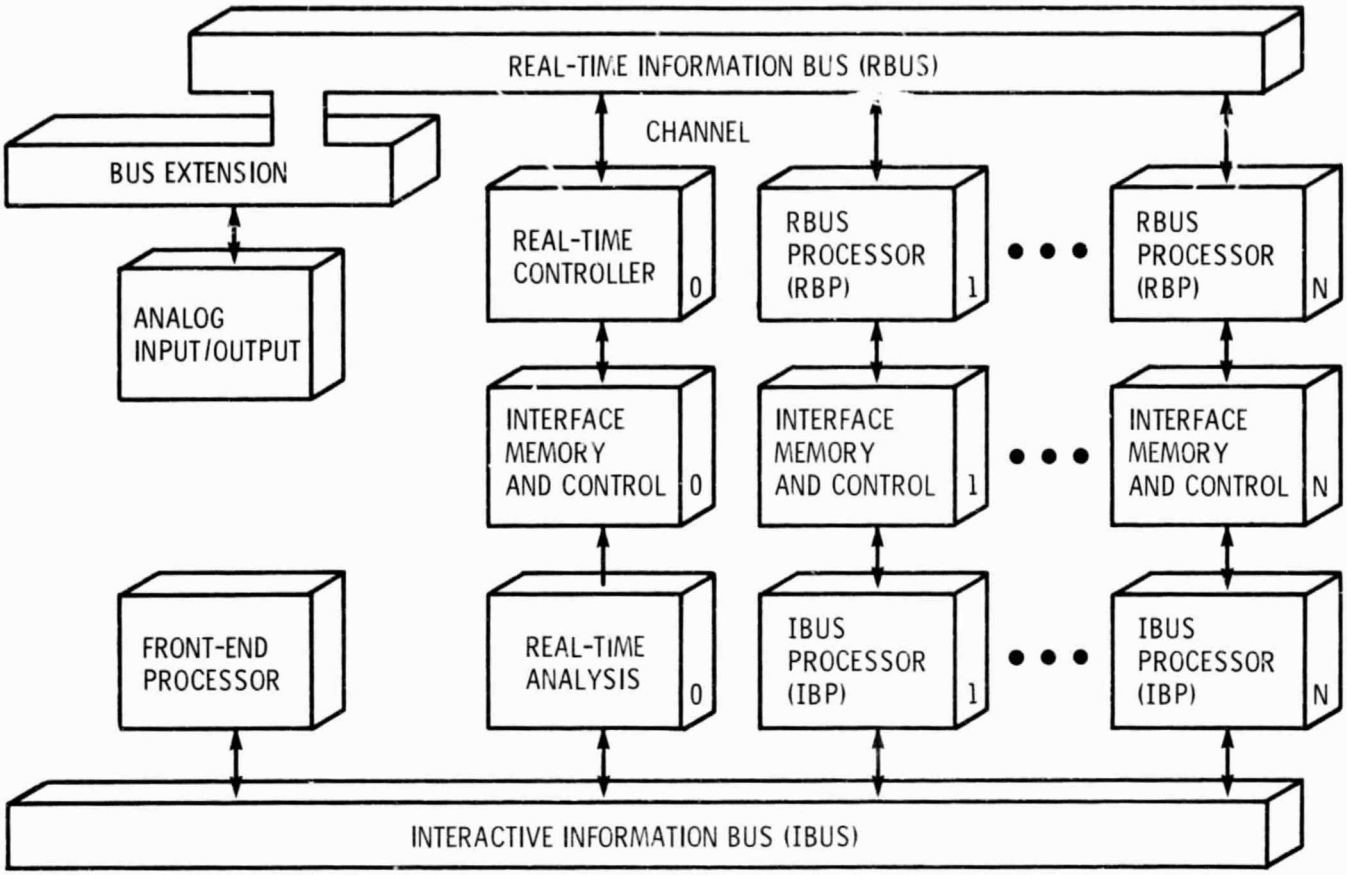

Figure 1. - General simulator configuration.

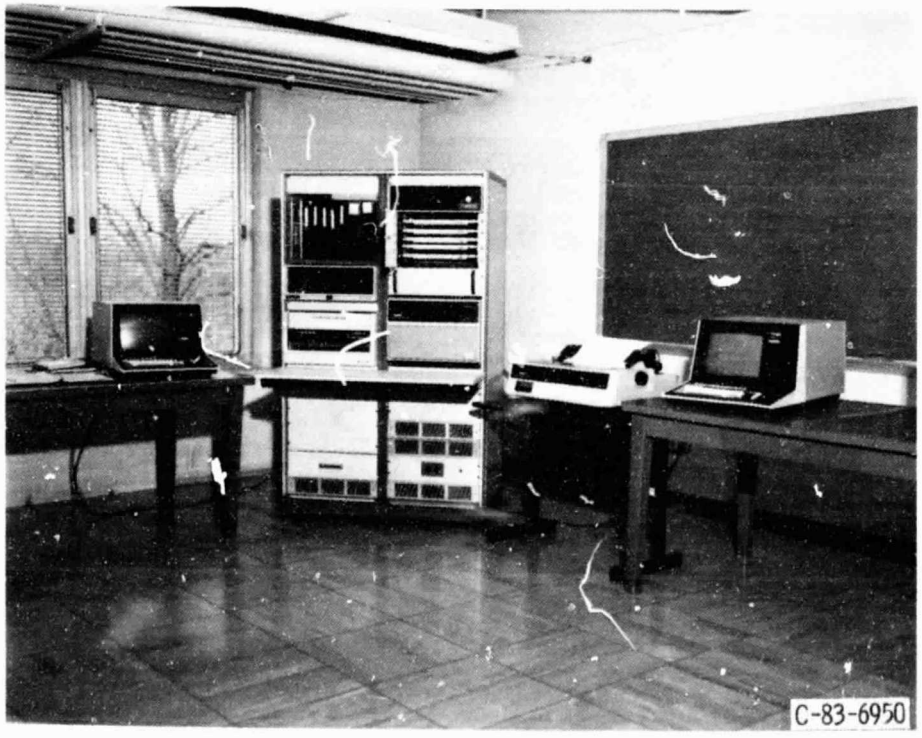

Figure 2. - Real-time multiprocessor simulator.

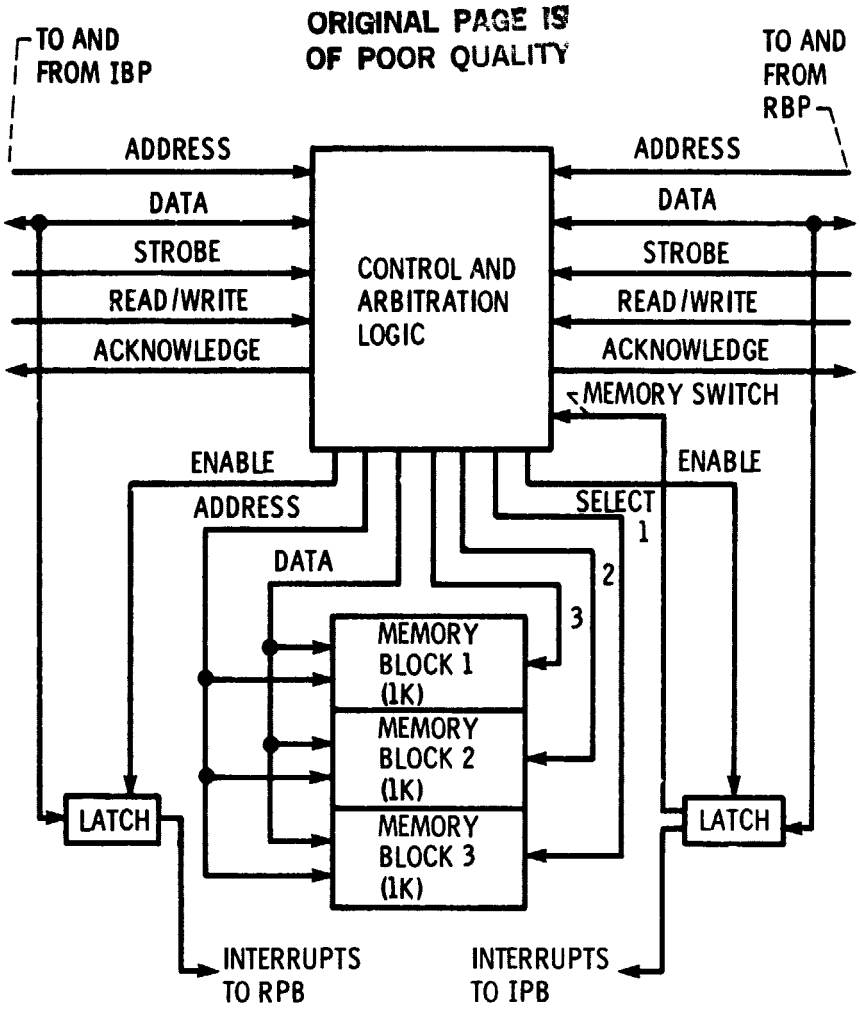

Figure 3. - Diagram of interface memory blocks.

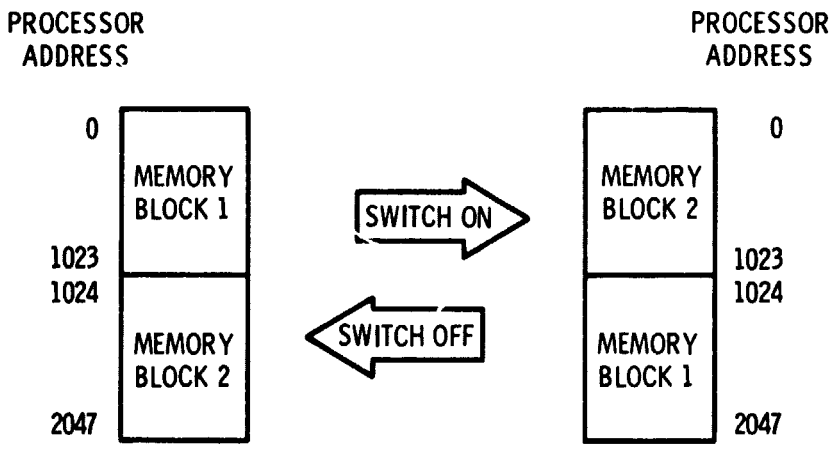

Figure 4. - Memory switching arrangement.

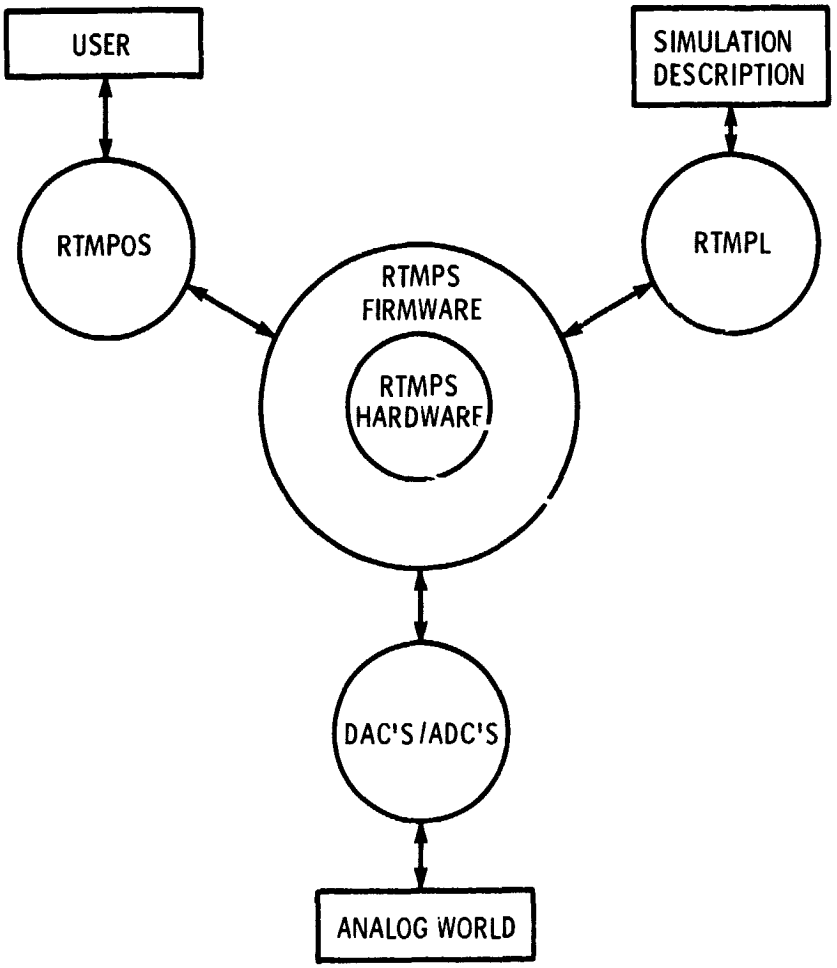

Figure 5. - RTMPS firmware interfaces.

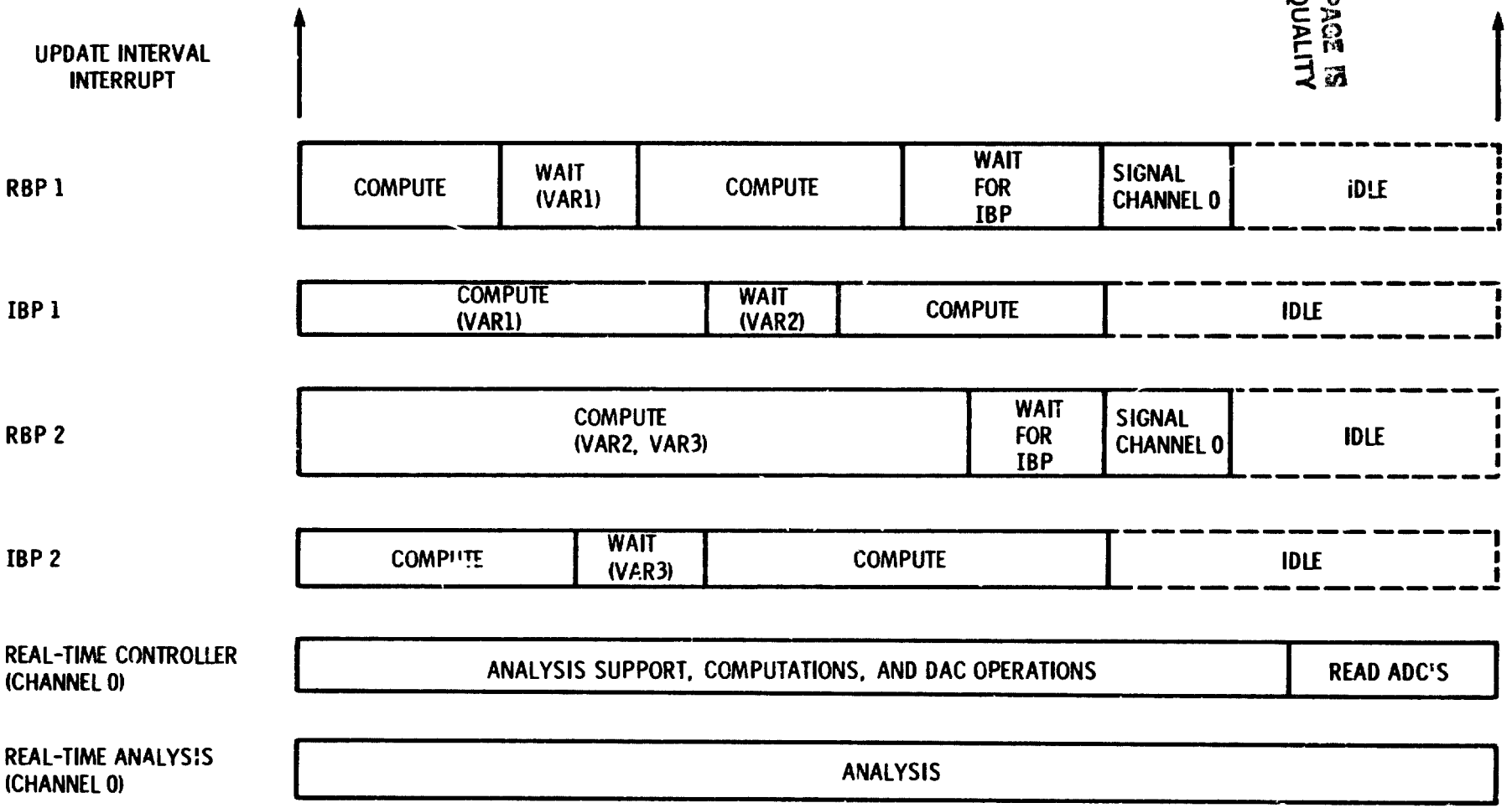

Figure 6. - RTMPS run cycle timing.