Harris Real Time Express™

Background information on a new concept for real time control, plus descriptions of the RTX™ software development system, development board, toolbox, and the FORTH environment.

Table of Contents

Harris Real Time Express A New Concept For Real-Time Control

WHAT ARE THE REQUIREMENTS FOR "REAL TIME?"

"How fast is fast enough for real time?" is a complicated question because of the application specific nature of real time computing. For example, an acceptable real time response for a transaction processing system might be a half-second response to a query. For an avionic control system, however, a half-second response to an external event is likely to be too slow. For the former, conventional data processing type processors are clearly an effective solution to real time requirements, but in many applications -- data acquisition, process control, robotics, local area network controllers, digital signal processing -- the response within a critical time period is extremely important. As a result, real time can only be defined within the context of the end application. It is driven by the processing requirements to meet time critical external events.

Many computers need fast instruction execution speeds. However, for real time applications interrupt latency and context switch times are important specifications in addition to the raw instruction execution speed of a processor. Recent advances in hardware and software have reduced response times for interrupts and context switches down an order of magnitude for many processors, to the many tens of microseconds or many hundreds of microseconds, respectively. The RTX 2000 brings these response times down another order magnitude to 400 nanoseconds for interrupt response time,and two microseconds for context switch time, while achieving an instruction execution rate over 10 MIPS.

A key consideration for real-time processing is predictability. Most general purpose computers have features to improve average instruction execution times. Features such as pipelines, caches, on-chip registers, and optimizing compilers all contribute to an improvement in average execution rate. But, they also contribute to more uncertainty with respect to critical timing predictability. As a result, external logic, such as DMA controllers or I/O processors, must be provided to support all but the most routine interface requirements with the external world. The RTX 2000 significantly contributes to moving more hardware into software as a result of high instruction execution rate, rapid responses to external events and predictable timing of instruction execution.

HIGH SPEED MICROCONTROLLERS - A NEW ALTERNATIVE TO CONVENTIONAL MICROPROCESSORS FOR REAL TIME

Most conventional microprocessors are optimized for an office automation computer or computer-aided workstation computing environment. Significant complexity is added to these machines in order to support the memory management and general purpose data processing requirements of these applications. Microcontrollers, on the other hand, take advantage of the increased circuit density made possible by advances in silicon processing to enhance the device for controller or other specific applications. This has produced a large variety of devices that are useful for specific applications as well as microcontrollers for more general purpose use. However, microcontrollers have traditionally been much slower than microprocessors and have not been optimized to support real time applications.

Microcontrollers can be broadly classified as either application generic or application specific. The generic products, such as the RTX 2000, provide solutions for a broad variety of applications. The RTX 2000, however, with its unique ASIC Bus, provides the capability of optimizing the solution by simplifying the partitioning between hardware and software through the use of external ASIC peripherals. The obvious extension to this philosophy is to incorporate those peripherals on the ASIC Bus within the IC. Therefore, application specific devices (both standard product or customer specific) can be developed as a natural extension to the core processor.

The emergence of high speed 16-bit microcontrollers provides a new opportunity to replace hardware with software. A major limitation to the ability to replace hardware with software has been the performance constraints of software solutions. The RTX 2000's substantial increase in performance provides new opportunities to replace random logic with software providing significant improvements in flexibility and time to market. For example, because of the very high speed of the RTX 2000, a full duplex UART can be emulated in software and requires less than 1% of the processor's bandwidth to perform the functionality of a 1500 gate UART.

RTX 2000 - PERFORMANCE THROUGH SIMPLICITY/ PARALLELISM

The RTX 2000 achieves performance through simplicity. The chip is designed for simplicity. It has no pipeline; no microcode sequencer; and no microcode. All instructions except memory access instructions execute in one cycle (memory access instructions execute in two). The RTX 2000 minimizes address calculation delays by incorporating a simplified memory paging mechanism, and eliminates the complexity of multiple addressing modes and memory management. The RTX 2000 is a stack machine. Stacks facilitate the evaluation of expressions and minimize the control overhead needed to organize data. The stack uses only a single pointer register to keep track of and access its data. A stack machine not only uses a stack for temporary data storage, but executes all operations on data from the stack. The ALU thus finds all of its data in a pre-defined location, and can get that data without an address specification. In addition, no addresses need to be compiled for stack access. The RTX 2000 also has a hardware return stack which handles subroutine return addresses. This stack can also be used for temporary data storage as well.

The RTX 2000 instruction set is sub-divided into six instruction classes, with each section controlling a hardware operation. Like horizontally micro coded bit slice architecture instructions, multiple operations can be compacted and coded within a single OP code to execute in parallel. Four separate buses for the data stack, return stack, memory and ASIC Bus and operate in parallel, significantly increasing instruction execution efficiency. For example, the OP code BE68 (8 G@ ;) simultaneously references all four address spaces. It fetches a 16-bit data value from the ASIC Bus, pushes it onto the data stack, forces a subroutine return, which pops a 20-bit return address from the return stack and fetches the next instruction all within a single clock cycle.

In a conventional processor, high-level structured programs are converted from groups of procedures with stack-oriented local variables to machine code. A considerable change in the look and feel of the program takes place as high-level language operations are transformed into groups of primitive operations. While the complex machine instructions that may support such stack operations (such as frame pushes and pops) and even fetch a variable (given a frame pointer and an offset) the paradigm switches from variables and frames (in a high-level language) to registers and memory pointers in machine code. The means of passing information between many high-level language procedures is the stack. The way of passing information between conventional machine language instructions is through registers or discrete memory locations. The fundamental mechanisms are completely different. Furthermore, conventional machines do not support efficient subroutine calls. Many clock cycles are required for managing the internal operations.

The RTX 2000 uses simple, and fast hardware to execute high semantic content instructions that closely reflect the structure of the program. Performance is not penalized for organizing programs into small, compact, understandable procedures. This results in compact program structures that are composed of hierarchically arranged solutions to sub-problems. Thus programs can be simultaneously optimized for small memory space, fast execution speed, and low development costs. This allows the hardware/software environment to deliver cost effective solutions to the users problems.

SUBSTITUTING PARALLELISM FOR PIPELINING IMPROVES SPEED WHILE IMPROVING REAL TIME RESPONSIVENESS

Pipelining is a common architectural strategy to increase the speed for conventional processors. For example, portions of a processor concentrate on fetching instructions, fetching operands, computing values, computing next addresses, and storing results. This method is a very efficient mechanism to increase speed for sequentially executing programs at a relatively small cost of added hardware complexity. However, pipelining impacts the timing of instruction response to subroutine calls, interrupts, and context switching, and the speed increases achieved by pipelining can be lost for highly structured programs.

The RTX 2000 abandons the use of pipelining as a means to increase circuit speed, and substitutes parallelism. The parallelism comes from exploiting two fundamental characteristics of the RTX architecture. The dual stack Quad Bus architecture provides an efficient mechanism to increase simultaneous operations. Harvard architectures have become increasingly popular in many applications because they effectively double the bandwidth of a microcomputer bus system. However, this is at the expense of increased interface requirements to separately address and interface both the data and program memory spaces.

In the case of the RTX 2000, parallelism is achieved by having on-chip stack memory for both parameters and return addresses, as well as an interface to main memory and the ASIC Bus for hardware acceleration. Since the stack buses are on-chip, I/O restrictions are eliminated. Also, since stacks are implicitly addressable without requiring address fields in the instruction, the number of functions that can be included in each word is increased. This leads to a significant improvement in performance due to the increased amount of work that is done within every instruction execution.

Another important difference between a conventional RISC machine and a CISC machine is the large semantic gap between high-level language source code and its corresponding machine code. This results in creating a large number of machine instructions for every high-level language instruction. The RTX 2000 differs dramatically from that mode of operation by having a single instruction correspondence to most FORTH instructions and, in fact, can pack up to three FORTH instructions in a single word. This high semantic content in each instruction greatly improves the effective operating speed of the processor.

Another limitation of pipelining in a heavily structured or asynchronous real time environment is the inefficiency and uncertainty of emptying and refilling the pipeline when branches or interrupts occur. The RTX 2000, by eliminating the pipeline, significantly improves the utilization of all available memory cycles and reduces the timing uncertainty of instruction execution.

STACK BASED PROCESSORS - OPTIMUM FOR REAL TIME STRUCTURED PROGRAMS

Conventional computers are optimized for executing programs made up of streams of serial instructions. Conversely, modern programming practices stress the importance of non-sequential control flow, and small procedures. The result of this hardware/software mismatch in today's general purpose computers is a costly sub-optimal compromise. In fact, the very philosophy of conventional RISC architectures is to implement only the most used instructions by studying software programs which have been based on existing architectures. This is a self-perpetuation of the programming style dictated by register based Von Neumann machines. The RTX 2000 takes a radically different approach by optimizing the instructions set to the requirement of a particular high level language (FORTH) which is well suited to real time control. However the architecture is also well suited to other high level languages. The objective is to promote the use of a highly structured programming style for real time, without the usual performance penalties. The RTX 2000 provides an efficient procedure for subroutine call through the use of the stack to store the parameters and a highly efficient subroutine call itself. A subroutine call takes only one clock cycle and the return can take zero clock cycles (since it can be implemented within the last instruction of the subroutine sequence).

RTX - PROVIDING FLEXIBLE PARTITIONING BETWEEN HARDWARE AND SOFTWARE

While semicustom ICs provide an attractive performance and integration alternative to standard microprocessors, they are not amenable to dealing with change. Off the shelf programmable ICs, with their support hardware and software tools, offer far greater flexibility. Consequently, when designers face requirements for both high performance and adaptability, some are finding that a mixture of application specific ICs and standard products is the optimum approach. The ability to partition the task between hardware and software is an important element in achieving total system performance. The RTX 2000 significantly increases the speed of solving hardware problems in software. In some applications, however, only hardware is fast enough. The ability to make an efficient partitioning between what is implemented in hardware and what is implemented in software is a key element in achieving system cost and performance objectives. The RTX ASIC Bus is a unique approach to assist developers in partitioning hardware and software efficiently. Because the memory bus and ASIC Bus run concurrently and the ASIC Bus can be operated on directly by RTX instructions, hardware accelerators can easily be added internally or externally to speed up processing functions.

RTX vs. CONVENTIONAL RISC PROCESSORS

Traditionally, most embedded control processing functions have been performed by general purpose microprocessors such as the 8086, 80286 or 68000 family. With the emergence of RISC computers, many manufacturers are claiming that they are well suited for embedded control applications. RISC processors are generally optimized for the requirements of 32-bit workstations and super microcomputers. While they are clearly capable of providing computing power for embedded applications, they have many specialized features that relate to supporting the UNIX environment and the specific requirements of computer aided engineering. While these processors have a small simple instruction set to achieve speed, they are hardly simple devices, most having in excess of 200,000 transistors, extensive memory management, and extensive pipelining. In addition, they require complex compilers to create efficient optimized code. As a result, the programmer loses visibility into the actual operations of the machine, thus creating a difficult to design environment for time critical functions. Since the machine is dependent on the operation of this optimizing compiler, programming in assembly language for those time critical applications is prohibitive.

While on-board memory management is useful for computer aided engineering functions in a multi-user environment, most realtime applications run logical to physical, not using virtual addressing at all. There's usually no need for virtual addressing in realtime applications. While the protection features of a memory management unit can be handy during program development, they are usually not required once the finished code is running.

Most RISC machines make extensive use of registers. As an example, the AMD 29000 has 192 general purpose registers on board. While each task may not use all registers, swapping out an extensive set of registers during a context switch creates an excessive latency problem which is often unacceptable. Allocation of a fixed number of registers to each task becomes a confining condition for the compiler.

Several characteristics of FORTH facilitate a simple scheme for context switching. Conventional architectures are not fast at context switch because they use a large number of registers. Saving or restoring a FORTH task in an RTX context switch takes little time because FORTH uses as few as three registers. The RTX 2000 core contains only eight registers so that a complete context switch to store all of the registers can be done very quickly.

RTX - EFFICIENT OPERATION FOR MULTI TASKING REENTRANT PROGRAMS

Although all real time operating systems are multi-tasking, not all multi-tasking operating systems are real time. Unix, for example, takes far too long to answer interrupts and make a context switch to suit real time applications. Its file structure suits program development, but not online control. Unix does not use reentrant code. If 16 users invoke an editor, then Unix loads 16 copies of the editor, thereby consuming large amounts of memory. Further, it has only rudimentary facilities for inter-task communication synchronization.

Another advantage of a stack oriented machine is the capability to efficiently support reentrant code. Reentrant code is useful in real time systems for two reasons: first, it saves space, because many tasks can use the same reentrant code simultaneously. The fastest real time systems must keep all code in memory -- a practice that puts a premium on compact coding style. Second, reentrant code exactly suits multi-tasking since you can interrupt the process using reentrant code at any point in the code segment, and later resume the process with no adverse effects. FORTH produces code that is inherently suitable for reentrant programs. Other languages require discipline on the part of the programmer. Making a routine reentrant simply means that all variables must reside in an area private to the task using the code, not in the code itself. The penalty for using reentrant code can be increased overhead or more CPU cycles for conventional processors, because read and write operations are indirect rather than immediate. Because FORTH promotes an object oriented programming style, reentrant programs are more manageable and comprehensible, especially in a multi-tasking environment. Of all the languages commonly used for real time control, only FORTH offers a straight forward programming facility for building classes of objects.

FORTH - WELL SUITED FOR REAL TIME OPERATING SYSTEMS

FORTH was originally developed for real time applications, and from its inception it has included features designed to simplify handling multiple concurrent tasks. Harris plans to offer several real-time operating system solutions for the broad variety of application requirements that the RTX 2000 will serve. FORTH, Inc.'s polyFORTH, for example, which will be offered for the RTX 2000, includes a multitasking, multiuser real time OS which has been widely used in industrial and aerospace applications. polyFORTH uses a proprietary multitasking algorithm which has been benchmarked at between four and twenty times faster than other real time OSs on CPUs of the 68000 and 8086 families (I&CS, October, 1987).

The secret to polyFORTH's speed is a "non-preemptive" multitasking algorithm. This means that a task relinquishes control of the CPU under well-defined, predictable circumstances, instead of having control taken away unpredictably when a time interval is up or an external event requires handling by a higher priority task.

In systems using preemptive multitasking, a task may be suspended when it is partially through updating a variable, for example. To avoid this, mailboxes or shared variables may only be accessed through system calls, which resolve potential conflicts at some cost in overhead. In polyFORTH, however, such a situation cannot arise, and so a shared memory region may be used to contain data of interest to two or more tasks with no OS overhead involved.

The polyFORTH multitasker is a simple round robin, and normally tasks don't have priorities. The non-preemptive round robin algorithm ensures optimum performance to all tasks. A task relinquishes the CPU whenever it performs any kind of I/O operation (including "virtual I/O" such as writing into screen memory). When the operation is complete, the task will wake up on its next turn. In most real-time applications there is so much I/O being performed that this is sufficient to guarantee rapid turnaround. There are commands available to "tune" CPU-intensive operations, if necessary. A real-time clock helps you to monitor how long tasks have to wait to wake up, and do whatever tuning is appropriate.

Another way in which polyFORTH ensures that all functions are performed as fast as necessary is by having no OS overhead whatever on interrupt servicing. Event response in polyFORTH is a two-layer process: interrupts are serviced instantly, with the hardware vector going straight into the application service routine. This service routine handles the most time-critical operations (reading a value and storing it in memory, for example), then notifies the task responsible

for the interrupting device that the event has occurred. The task will "wake up" on its next pass through the round robin and handle the more complex aspects of processing.

The combination of low-overhead task management with instantaneous interrupt servicing provides the ability to handle extremely high data rates and complex real time applications with ease.

RTX COMPILERS FOR OTHER LANGUAGES

While the RTX 2000 directly executes FORTH, efficient compilers for other languages can also be developed. The architecture for the RTX 2000 is well suited for providing facilities for the efficient implementation of other languages such as "C", PROLOG and ADA. "C" compilers also use stacks to create local variables and to pass run time parameters among functions. "C" breaks tasks up into functions. A "C" program consists essentially of a series of functions with one beginning function specified as main ( ). It is straight forward to implement a "C" language run time allocation stack using the RTX 2000's fast access user memory locations as pseudo-registers. Data can be accessed by allocating one of the RTX 2000 pseudo-registers (first 32 words of memory) to create a pointer into the stack. This stack can be used to file clusters of information called frames. Offsets into the stack are addressed to fetch as 2-part addresses. A routine to find data first asks which frame and then which element in the fr e to fetch. A stack frame is a miniature segmented memory with a 2-part address.

Stack frames store information on entry to functions. They permit temporary storage of variables and parameters so that subsequent routines can run without interfering with other functions, variables and parameters. The RTX instruction set pseudo-register operations support the capability to develop an efficient "C" compiler. Since instructions execute at a high clock speed, the use of pseudo register enables micro code-like performance of custom "C" run time stack instruction sequences.

RTX 2000 - WELL SUITED FOR REAL TIME ARTIFICIAL INTELLIGENCE

The recent flurry of activity in commercial expert system development has all but bypassed the real time computing community. Although it is desirable to incorporate expert systems in some real time computing applications, the amount of computing over symbols (reasoning) required by an expert system is difficult to implement in real time. General real time symbolic processing has remained an AI research problem, however a number of applications have been implemented which use FORTH. FORTH provides an integrated environment for both conventional data processing and AI with a FORTH Prolog compiler. This technique has been applied in a diesel electric locomotive repair expert system, an orbiting spacecraft command and control system, a spacecraft trajectory processing data error detection system, and a real time polysomographer sleep disorder diagnosis system. Other applications include utilization of expert system capabilities for such applications as radar and sonar processing, image compression and analysis, etc. A Prolog compiler is being developed which provides a set of high-level artificial intelligence programming tools (i.e., inference engine, language parser, etc.) built from FORTH primitives to take advantage of the high run time execution speed offered by FORTH. In this implementation, the Prolog interpreter is imbedded in a FORTH environment. Real time algorithms stored in the knowledge base through this mechanism can subsequently be executed on a logic driven basis by the expert system.

RTX ALGORITHMIC COPROCESSORS IMPROVE TOTAL SYSTEM PERFORMANCE

Designers of both microprocessors and peripheral interfaces are struggling to minimize the traffic jam their own success has created. New techniques in bus management, heavy use of specialized memories like caches and FIFO buffers, and a new reliance on distributed I/O processing are all strategies being used to resolve microcomputer I/O bottlenecks. As these techniques are more widely used they are changing the architecture of silicon components, and they are equally affecting the practice of microcomputer system design. The RTX 2000 provides a significant capability to attack the I/O processor and coprocessor requirements to improve overall system performance by distributing the processing. Of significance is the development of a shared memory to provide an efficient interface between an RTX coprocessor and host. The ability to have a stored program controller as a coprocessor provides the capability to compute complete algorithms rather than just a single operation at a time. This provides a major improvement in the system performance due to elimination of heavy bus overhead and the inefficiency of tying up the host processor to service the coprocessor. Such applications as Local Area Network controllers and virtual disks, demand that memory pages be swapped over the network or disks for remote file service, but the increased frequency with which packets arrive and depart can cause a workstation to approach its memory bandwidth limitations. Another example is when a microcomputer forms the backend of the signal processing system. Data may be required for the digital signal processing algorithms at very high rates. Maintaining data flow through the system may demand that the system microprocessor continue running at these high speeds until all data is processed.

The push to make individual components run faster soon runs into complications. The utilization of coprocessing controllers within the microcomputer environment is a

technique which can significantly increase performance of the overall system by reducing the amount of bus interface time for the host processor. A powerful strategy for breaking up bottlenecks is to move the I/O driver code from the CPU to the peripheral controllers. An obvious example is in serial concentrators where the processor monitors the number of serial lines, accumulating data until the entire block has been received. In a multi-user UNIX system for example, this function usually requires a serial concentrator to perform basic UNIX line editing functions since these operations must occur before the end of line character comes in from the terminal.

The strategy of removing the responsibility for device and housekeeping off the CPU and on to the peripheral controller dictates that the device controllers become programmable to pick up many of the tasks formerly executed by the device driver software in the host. Moving these tasks to the controller not only removes slow dumb tasks from the central resource, but also spares the central processor the hail of interrupts, context switches, commands and status bytes that are a necessary side effect of centralized device drivers. This not only maximizes I/O speeds, but it also simplifies the user's programming requirements. The I/O controller concept provides a system design that emphasizes distributed processing with a high degree of concurrency and parallelism. In addition, this architecture provides an environment that reduces the data movement within the system.

Currently, I/O processors of this class are based on bit slice architectures in order to provide the performance necessary to manipulate data on the fly. A standard product microprocessor such as the RTX 2000, with a high level programming language, helps users interface standard I/O ports to the host processor. The result is an architecture optimized for high performance, but with a high degree of modularity and user programmability, saving developers both time and expense. The utilization of I/O processors also helps avoid arbitration between devices producing input and output data streams. Since the RTX 2000 has all of the provisions to support direct memory access management, the processor has the capability to incorporate smart DMA, which relieves the host of this burden.

REAL TIME SOFTWARE DEVELOPMENT IS MORE COMPLEX THAN CONVENTIONAL PROCESSING

The characteristics of real time software systems that set it apart from conventional data processing applications are that real time systems must:

- Respond to real world stimuli.
- Within a finite period of time.
- By directly manipulating hardware resources.
- As a set of concurrent asynchronous processes.
- With a high degree of reliability.

The RTX family contains both software development tools as well as fast hardware to operate in this environment. It promotes an interactive programming environment which has four primary attributes:

- A set of highly integrated tools.
- Which are low cost and simple to use.
- Which do not burden the target system with unnecessary complexity.
- Which promote use of the underlying structure of the program as an organizing tool.

STRUCTURED PROGRAMMING SUPPORT REDUCES DEVELOPMENT/DEBUG COSTS FOR REAL TIME

Hardware that is fundamentally based on the concept of modularity and programmer interactiveness will lead to changes in programming style that will better support efficient software development. The expense of using a software programmable microcontroller to solve a problem consists of not only the money for hardware, but also the development costs of creating and debugging the code. Previously the cost of solving problems with computers was dominated by hardware costs, but as hardware costs have plunged, software costs have grown by leaps and bounds. This is nowhere more evident than real time control. Real time control applications tend to be significantly more complex than conventional programming and, ironically, offer the least amount of high-level language support for those time critical functions. The developer is caught in the dilemma of trying to write most of the program in a high-level language support such as C or ADA, only to have to merge in assembly language programming for the most time-critical applications. Worse, during the debug process there is a poor correlation between what is written in the high-level language and what appears in the machine language, because the compiler has dramatically altered the programming style of the program. The compiler modifies content by providing optimization including the unrolling of loops into in-line code, and expanding the lowest level procedures as macros within the calling routines.

RTX - MAINTAINING KNOWLEDGE OF HARDWARE OPERATION WHILE EXECUTING A HIGH LEVEL LANGUAGE

Since the RTX creates a virtual FORTH machine, the correlation between the high-level FORTH language and the operations executing in the FORTH engine are closely coupled. Therefore, the uncertainty with regard to what the compiler is doing to the source language of the real time system is significantly reduced. Also, by providing symbolic capabilities within the development system and operating completely within the high-level language, software and hardware debug and integration is significantly simplified.

Optimizing compilers obscure the correspondence between source code and compiled object code. Programs written in a high-level language that need to meet specific response time specifications require the programmer to switch on the compilers optimizer and then debug optimized code. Among the many techniques optimizing compilers employ is register allocation by coloring. Coloring keeps the most commonly used values and registers at all times. The compiler examines the entire subroutine to determine which local variables and parameters are used most often in a routine. It allocates them to registers. Further, the register allocator can use data flow analysis to find the lifetime of each variable. Using this information it can increase the number of variables that get stored in registers by using the same register for several variables in the same subroutine.

A software developer must be very familiar with the operation of the compiler in order to understand the implications of what is happening in real time software. The designer cannot be assured of just where program variables are as when programming in assembler language and making the variable assignments explicit. A key feature of writing in FORTH and executing in FORTH on the RTX 2000 is the close correlation between the high-level language and the actual execution of the machine. Real time programs which must respond in a critical time period can therefore be more easily designed since the complete operation of the machine is understood.

RTX IMPROVES SOFTWARE PRODUCTIVITY FOR REAL TIME

According to the Defense Science Board Task Force on military software, most software productivity gain has been brought about by three factors. First is utilization of a high level language. The removal of awkwardness of machine language has been shown to increase software productivity by a factor of 10. While the benefits of a high level language have been available for low performance functions, many real time applications heretofore have often been programmed in assembly language. Even with high level languages, slow turnaround, edit, compile, link, load, and debug cycles contribute to a loss of mental continuity in the development and debugging of software. An integrated, interactive development environment has been shown to result in improvements in productivity from two to five. Finally, compatibility of files, formats and interfaces among the various tools has been demonstrated to increase productivity by a factor of two. These are precisely the benefits of FORTH that have been incorporated into the RTX family development tools.

REAL TIME DEBUG DEMANDS SPECIAL CONSIDERATIONS TO OPERATE AT FULL SPEED

In a typical software development environment (the host and target method), programmers use a general purpose host such as a DEC VAX, a CAE workstation, or an IBM PC AT to generate application code. After they write the code, programmers must transport it to the target for final testing. Unfortunately, for most embedded microprocessor development the target system does not offer enough programming and test resources to support the total development process. In order to debug the hardware, an In Circuit Emulator (ICE) is the most traditional vehicle for testing programs written for embedded processors. While an ICE is a powerful tool for software and hardware integration, its high cost may be difficult to justify for an application specific device. Of more importance, the cabling requirements and timing constraints for a 10-MIP processor provide a significant performance limitation in the feasibility of debugging the system at full speed for realtime applications. The Harris approach provides a real time debugging monitor in the target system and is an excellent alternative to ICE. Because the RTX is fully static and eliminates pipelining, the problem of getting the hardware to work initially can be solved through single step operations using low cost logic analyzers and the host development system. Then the full capabilities of the integrated hardware/software development system can be used to provide a cost effective and high performance approach to achieving hardware/software integration.

RTX - OPTIMIZED FOR RAPID TIME TO MARKET

Time to market is becoming an increasingly important criteria for the makers of electronic systems. The problem has been intensifying as product lifetimes have progressively shortened. Makers and users of semicustom chips feel the pressure even more because of the need to generate prototypes before a system can be debugged and demonstrated, much less marketed. The RTX 2000 provides a cost effective means of implementing a core based processor in a semicustom environment. Because it is a standard product with an ASIC Bus designed to accommodate application specific peripherals, prototypes can be rapidly developed with a standard product. This significantly lowers the risk of development for a complex microcontroller based on a core processor. Because the ASIC Bus can be integrated on-chip, future versions can incorporate not only the core processor, but the external peripheral logic and memories all on one chip.

In almost every project -- military or commercial -- the faster the prototype is up and running the more likely the system will meet the market window. Shorter prototyping time means getting to market faster, with less risk. Realtime systems tend to have complex relationships with external hardware. A hardware prototyping vehicle is a useful mechanism to validate real time performance and is complimentary to the use of simulation.

RTX Questions and Answers

Q. What is Real Time Express (RTX)?

A. The Real Time Express is the new name for our FORTH-based RISC computer technology we previously called FORCE. RTX is an acronym for Real Time Express.

Q. What is RTX 2000?

A. RTX 2000 is the first standard product based on RTX technology. It is a high performance 16-bit microcontroller with on-chip timers, interrupt controllers, and 16x16 multiplier. A unique feature of this processor is the high performance ASIC bus which provides for architectural extension using off-chip hardware acceleration logic and application specific I/O devices. The RTX 2000 has been designed and fabricated using the Harris standard cell library and compilers and can be incorporated into customer ASIC designs.

Q. What is the RTX Family?

A. The RTX Family will include a broad variety of general purpose microcontrollers, standard product application specific products (such as LAN controllers) and customer specific ASIC's.

Q. Why did you change the name from FORCE to Real Time Express or RTX?

A. FORCE (FORTH Optimized RISC Computing Engine) was an accurate description of the product, but it did not focus on the primary strength of the product - real time, and fast time to market. While the architecture is FORTH optimized, it supports C well and we expect many other languages to be supported in the future. Also, we wanted to avoid confusion with FORCE Computers - a board computer company.

Q. What type of applications are best suited for RTX?

A. The RTX 2000 is best suited for applications requiring:

- extremely high speed instruction execution rates
- very high speed fixed point arithmetic requiring 16 bits or less precision (including 100ns 16x16 multiply)
- moderate speed floating point (speed similar to 8087)
- fast response to interrupts (400ns)
- rapid context (task) switch (2-5μsec)
- rapid decision times (100ns)
- high integration with peripherals on chip
- form factor/minimum chip count
- extremely low power dissipation at high speed (300 MW typ. @ 10 MIPS)
- full static CMOS with near zero power at idle
- memory requirements <1Mbyte
- path to semicustom
- embedded expert systems (artificial intelligence)
- multi-processor configurations

Q. What type of equipment requires this type of processing capability?

A. Computer I/O controllers (LAN, graphics, disk, DMA, etc.), computer peripherals (laser printers, line printers, FAX, copiers, disk drives, optical storage), communications (PBX, T-1 multiplexers, statistical multiplexers, fiber optic links), robots, machine vision, medical, avionics (1553 communications, display drivers, closed loop control systems, countermeasures, radar, sonar, etc. etc.) missiles (similar to avionics), process control, instrumentation and test equipment, machine controllers, commercial aviation.

Q. Can RTX serve digital signal processing (DSP) applications?

A. Definitely. While there are some competitive products which are optimized for specific DSP operations, the RTX offers excellent general purpose DSP capability due to its high instruction execution rate, on chip 16 x 16 multiplier and stack architecture. A significant advantage is the ability to achieve high speed while programming in a high level language. In addition, RTX can implement the data acquisition and control algorithms offering very high integration. DSP peripherals can be attached to the memory or ASIC bus to further accelerate DSP functions.

Q. Why 16 bits, when others are introducing 32 bits?

A. The RTX packs more work in its' 16 bit instructions than 32 bit RISC processors due to its' innovative architecture. Most real time controller systems interface with 8 - 10 bits (video) or 12 - 16 bit (sensor) analog/digital converters which don't require 32 bits of precision. Also, data handling for LAN, graphic and other computer peripherals is done with 8 or 16 bit words. 32 bits creates larger, more expensive die, requires more external memory and bus drivers, and dissipates more power. 32 bit processors also restrict the amount of on chip peripheral functions which can be integrated at the same cost and create added complexity for interfacing to 8 and 16 bit peripherals common in real time control.

In summary - for most real time applications, 16 bits does the job, at the same or superior performance as 32 bits, with lower price, with lower power, with more on chip functions.

Q. Does that mean RTX can't do 32 bit arithmetic?

A. No. The RTX performs 32 bit arithmetic and shifts. However, in applications requiring extensive 32 bit operations, a 32 bit processor may outperform us.

Q. What about those applications where 32 bits seem necessary?

A. Do you need to address a much larger memory space? Do you have extensive 32 bit data precision operations or require the highest performance possible in floating point? Do you really want to pay more for equivalent performance? Have you considered the added cost of memory for 32 bit high speed systems and the uncertain timing of cached, pipelined systems? Most 32 bit processors are >10X power and require 4X the peripheral circuits. However, if you really need 32 bits – we are developing an RTX 32 which will be announced later this year.

Q. Can RTX 2000 do floating point?

A. Yes. The RTX 2000 performs 32 bit floating point in software at about the same speed as an 8087 and in some cases as an 80287. Our compiler will have a full set of floating point functions. If faster speeds are required, floating point accelerators or ASICs can be attached to the ASIC bus to speed up floating point operations. A future product containing internal floating point is planned. However, some applications requiring optimization for highest speed floating point may be better served by other processors.

Q. Are there evaluation boards?

A. Yes, we have a development board which contains the RTX 2000, 16K bytes of EPROM, 32K bytes of RAM, three parallel input ports, three parallel output ports, a UART and user breadboard space and memory expansion space. This is available for $1,500. A technical backgrounder (Harris RTXDB Real Time Express Development Board) describes the capability of the board. Ask your sales rep.

Q. Are there software and hardware development tools available?

A. Yes, a complete software and hardware development system (RTXDS) can be purchased for $3,000. This includes the development board above plus software development programs and a hardware de-bugger. A description of this capability is contained in a Technical Backgrounder (Harris RTXDS Software Development System). Contact your rep.

Q. Why is RTX well suited for real time control?

A. Fast instruction execution, fast interrupt response time, fast context switch, fast branching, fast subroutine call/returns, predictable timing, FORTH language. For more information see Technical Backgrounder: Harris Real Time Express (RTX): A New Concept for Real Time Control.

Q. How is this microcontroller different from all the other newly announced RISC processors?

A. The AMD 29000, Motorola 88000, MIPS and SPARC RISC computers are 32 bit processors which have been designed to be the central processor for computer aided engineering workstations and large general purpose computers. They are optimized for supporting multiple users solving large computational problems requiring large memories. They require significant support logic to implement a complete computer.

The RTX is optimized for embedded real time control. RTX eliminates the complexity required for large multi-user general purpose processors and increases the functionality and performance for real time applications. A minimum system based on the RTX requires far less support products than other RISC processors.

Q. Why can't other RISC processors meet the time critical requirements for real time control?

A. Other RISC processors (and even complex instruction set {CISC} processors) have timing that is difficult to predict due to the use of caches, pipelines, and optimizing compilers which make actual performance extremely variable. Also, interrupts, branches, and context (task) switches can take up to an order of magnitude more time under worst case conditions than the RTX because pipelines and caches may need to be emptied and refilled.

Q. Why did Harris build a processor which directly executes FORTH?

A. We licensed (and subsequently have purchased) the core processor technology from Novix. Our first reason for selecting the processor was not FORTH. We recognized Novix achieved a major breakthrough – a processor which achieved performance through simplicity. A processor which could meet the requirements of real time control. A processor small enough to integrate into a CAD system and have enough space to integrate memory and I/O without creating large chip sizes and expensive die.

It turns out also that FORTH is a language well suited for real time embedded systems. FORTH creates compact code. FORTH is fast. FORTH simplifies real time development. FORTH is capable of primitive operations yet it is extensible (actually FORTH is an application specific language – we'll have more on that later). Because the RTX directly executes FORTH primitives, the combination of hardware and software is extremely fast. But of even more importance for time critical applications – the programmer can program in a high level language and know what the processor is doing. No surprises from an optimizing compiler. And best of all – no assembly language. The RTX 2000 runs FORTH as fast as assembly language (i.e. FORTH is the assembly language).

For more information on FORTH – see Technical Backgrounder: FORTH: A Software Development Environment.

RTX QUESTIONS AND ANSWERS

Q. Why is RTX well suited for embedded artificial intelligence (AI)?

A. FORTH is a "threaded" language. That means it can implement decision trees -- which are the heart of AI processing -- efficiently. A Prolog compiler is under development which will allow customers to incorporate rule-based expert systems into their applications easily.

Q. Who needs that?

A. There are significant opportunities in pattern recognition, threat identification, operator assistance, and error recovery. This is an emerging technology, but becoming very important. We will have extremely high performance and low cost. Our RTX 32 will be even better.

Q. Is it practical to put more functionaliy on chip?

A. Yes. The RTX 2000 is extremely small. 107K $MILS^2$ for the processor, stacks, multipliers and peripherals. Most RISC processors are two to three times as big with less peripheral functions. So, we can easily add more functionality including on chip ROM and RAM. When we migrate to our 1.2 micron process next year, we will be able to put even more on chip.

Q. Is there an In Circuit Emulator (ICE) available?

A. One of the beauties of the RTX is that it is static and has no pipeline and no cache. That means that it's easy to get the system up and running with single step operations and low cost logic analyzers. The hardware debugger contained in our development system contains the capabilities normally found in an ICE, but it allows operation at full speed and it is much lower cost than an ICE.

Q. Don't you require high speed memories to achieve 10 MIP performance?

A. All processors operating at >5 MIPS require high speed memory. However, RTX requires far less fast memory than other processors due to the extremely compact code which results from having an efficient subroutine call and a FORTH programming style. This is important for small systems which only use fast storage. For those applications requiring larger (and slower) memory, the fast subroutine call allows the user to put all frequently used subroutines and instructions in fast RAM, thus eliminating the complexity of cache memories required by other high speed processors.

Q. What about memory requirements >1 Mbyte?

A. RTX creates extremely compact code, as much as an order of magnitude, less than other processors. If additional memory is required, external logic can be used to extend the address space beyond 1 Mbyte at a small impact on system performance.

Q. How does the RTX compare to other microcontrollers?

A. The RTX has 10X - 20X (sometimes 100X) the performance of other microcontrollers (such as Intel's 8051, 8096, 80196, National's HPC, NEC, etc.). These are 16 bit controllers. We can substantially upgrade performance or use our speed to eliminate external logic. Conventional controllers are lower priced but Harris offers more performance/buck. If your application can benefit from increased speed or reduction of peripherals, then you should consider RTX.

Q. Where can you substitute RTX for CISC microprocessors?

A. Many CISC microprocessors such as 68020, 68030, 80186, 80286, 80386 are used as embedded controllers, because microcontrollers are too slow. For example, many LAN's are controlled by 68020. We offer higher integration, fewer peripherals, faster speed, lower systems cost, lower power, high level language programming, and superior overall performance.

Also, we are a great coprocessor for other CISC microprocessors. We can offload the time critical functions from a CISC, leaving it to do more central processing functions.

Q. Can you substitute RTX for bit slice?

A. Yes. For those applications where bit slices are not being used for instruction set compatibility (such as minicomputers) or highly specialized architectures (such as systolic signal processors) we can offer a major improvement in speed, power, integration, cost, software development tools, etc. etc.

Q. Is RTX 2000 an alternative to semicustom?

A. Semicustom is used for many applications which require speed and integration. However, semicustom creates hidden costs in time to market, lack of flexibility for change, non-recurring expense to correct system errors, and cost of lost inventory if a system bug is discovered after parts are ordered. In some cases, RTX can meet performance, cost and integration requirements. RTX provides a significant opportunity when possible to displace semicustom because its speed is 10X conventional microcontrollers - allowing logic to be implemented in software. Also, a combination of RTX and simple semicustom ASICs may provide a favorable alternative to one complex semicustom ASIC.

Furthermore, RTX provides a superior approach to developing semicustom ASICs. Since it is based on proven macrocells, has robust hardware and software development tools, and is fully integrated into a CAD system, it can reduce risk and add flexibility to full semicustom designs.

RTX QUESTIONS AND ANSWERS

Q. Will Harris offer a military version of the RTX?

A. The RTX is designed to be fully 883C compliant. We will introduce an 883C compliant product during Q4 of 1988.

Q. What about rad hard?

A. Since the RTX is developed in our standard cell library, migration to a rad hard process is straightforward. A rad hard product will be announced in 1989.

Q. How do I learn to use the development tools and learn the details of the RTX 2000?

A. A 3 - 4 day training program will be available starting in September which will provide customers with in-depth training of RTX development tools and the RTX chip set.

Q. Do you have a second source?

A. We have not selected a second source. However, we have several interested second sources. Harris plans to announce a second source by the first quarter of 1989.

Q. Is Harris committed to this product family?

A. Definitely. We expect to use it broadly as a standard product, semicustom, and custom product. In both divisions. Note - RTX (FORCE) was featured on the cover of the Harris Corporation FY'87 Annual Report for the whole $2.1B corporation. This is a major initiative.

Q. What are your next products?

A. Harris is developing a pin for pin compatible version of the RTX 2000 but without multipliers and smaller stacks. This product is well suited for LAN controllers and other computer I/O. It will sell for less than $100 in a PLCC. A 1553 LAN processor incorporating a Mil Std 1553 interface and an RTX core processor is in development. Also planned is a unit with internal floating point acceleration. Other products will be announced later.

Q. How do I learn more about RTX?

A. Contact your local sales office and ask to be placed on the RTX mailing list. If you have specific technical questions, they can refer you to an RTX applications engineer.

Harris RTXDS Software Development System

RTXDS, the Harris Real Time Express Development System is a set of integrated tools for use in the development of software for the Harris RTX processor. RTXDS is part of the Harris RTX Toolbox, a complete hardware and software system for breadboarding and debugging RTX-based products and semi-custom integrated circuits.

The RTXDS system consists of two subsystems. The "Host" portion runs on an IBM-PC/XT/AT family computer and provides an interactive software development environment. The "Target" portion resides on the user's RTX-based application board and provides run-time debug support for applications code.

The RTXDS system gives designers the power and advantages of using a high-level language - FORTH - in an interactive environment, without the need for developing a full FORTH implementation for each new application system.

The static nature of the RTX architecture and its lack of instruction pipelining eliminate the need for a hardware In-Circuit-Emulator (ICE) and its associated speed limitations. This means that an application program can be run at full speed for testing time-critical code sections, or single-stepped under hardware or software control for debugging problem areas. The RTXDS system supports both modes of operation, allowing operator interaction at the single instruction level, or letting a program run undisturbed until the operator decides to intervene.

RTXDS provides the following tools:

- a Host/Target Monitor for debugging applications on an RTX-based target system
- A PC-based FORTH development system, including editor and 8086/286 FORTH compiler
- an RTX FORTH cross-compiler
- Disassembler for displaying compiled RTX machine code as individual RTX instructions
- a DOS File Interface utility for saving/loading RTX code

RTXDS allows a user to develop an application in FORTH on a PC, using a full-screen editor and file and printer utilities. Non-hardware specific functions and algorithms can be tested and debugged in the PC environment, with no target hardware present. When the target system becomes available, the application can be cross-compiled with the TFORTH compiler to generate RTX machine code.

The compiled code may be debugged using the Interactive Host/Target Monitor system. This system provides an interface between the Host Monitor on the PC and the Target Monitor running on the RTX target system. Operator commands at the PC are transmitted through a serial link to the RTX target system. The target system's responses are formatted and displayed at the Host PC. The Monitor provides commands for downloading RTX code from the PC to the target system, examining and changing target memory locations, registers and I/O ports, executing code, setting breakpoints, and monitoring memory accesses. Debugging through the Host Monitor is done at the symbolic level, using actual FORTH names from an application, rather than referring to numeric memory locations.

The Target monitor provides a FORTH-like terminal interface for applications programs to print debugging and status information while running. Unlike traditional debuggers, which typically only report breakpoints and possibly a register dump, the RTXDS system allows a programmer to perform any FORTH operation as part of an embedded debugging command. For example, an application program might, as part of a debugging statement, read an input port, test a bit, then perform different actions based on the state of the bit. In addition, these debugging commands may be left in the source code, but not compiled into the final application, so that they may be used when modifying the code at a later date.

Highlights

Debug Monitor

- Dual window system - Host window running PC-based FORTH for operator interaction and Target window for displaying data generated by target system.
- Fully interactive - the operator may define new commands at any time
- Real time debugging - the application system can run at full speed
- Symbolic Debugging - application subroutines and variable names may be referred to by name or address
- Source code and compiled code available for viewing at all times
- User-programmable diagnostics - application program may display status information and perform any FORTH functions
- Debugging features
  - ▶ Download program code
  - ▶ Examine/change memory, registers, I/O ports
  - ▶ Execute code
  - ▶ Set breakpoints
  - ▶ Monitor memory accesses
- Application programs may run unattended, freeing the PC for editing, listing, etc.

TFORTH compiler

- FORTH-83 compatible
- Generates ROMmable code
- Optimizes code to take advantage of RTX architecture
- Extensions for RTX features
  - ▶ Memory mapping
  - ▶ On-chip controllers - timers, interrupt controller, stack
  - ▶ Hardware multiplier
- Compiled code may be formatted for programming into ROMs

Hardware Requirements

The RTXDS system will run on any IBM-PC/XT/AT compatible with the following:

- 256K of memory
- Two floppy disks or a hard disk (preferred)
- Serial port addressable as COM1
- DOS 2.1 or later

Harris Real Time Express Development Board

The Harris Real Time Express Development Board (RTXDB) is a ready-made RTX 2000™ Microcontroller system which is configured for immediate operation when used with the RTX Development System (RTXDS™) software. The RTXDB provides a development environment for real-time evaluation of the RTX 2000 and eliminates therequirement for user designed prototype boards.

Real-Time Control Automation: The Determining Factors

In real-time control automation, the quality of a control system is dependent upon many factors. Fast and predictable interrupt response, context switching, and instruction execution rates are primary factors in the ability of a microcontroller to meet critical timing requirements in a real-time environment.

The RTX 2000 Advantage

As a microcontroller designed for real-time applications, the RTX 2000 offers a combination of features and advantages which make it unique. Where many modern RISC architectures rely on pipelining and caching, the RTX 2000 does not. The unpredictability introduced by pipeline flushing or cache-miss related reloads causes these features to become a hazard in speed intensive tasks. Instead, the RTX uses a dual stack, highly parallel architecture with the Harris ASIC Bus™, which allows the RTX to achieve a very high level of predictability and speed.

By integrating these hardware features with software which uses an optimized instruction set, the RTX 2000 can produce fast and predictable interrupt responses of 400 nanoseconds, context switching times of less than 20 microseconds, and an instruction execution rate of over 10 MIPS. With these characteristics, the RTX 2000 exceeds the real-time response capabilities of conventional general purpose processors by a full order of magnitude or more. This exceptional performance can provide the critical edge needed for success in meeting severe real-time system requirements. In fact, the RTX 2000 is the first microcontroller which can be programmed in a high level language while still retaining this capability.

The RTXDB: A Convenient Development Environment

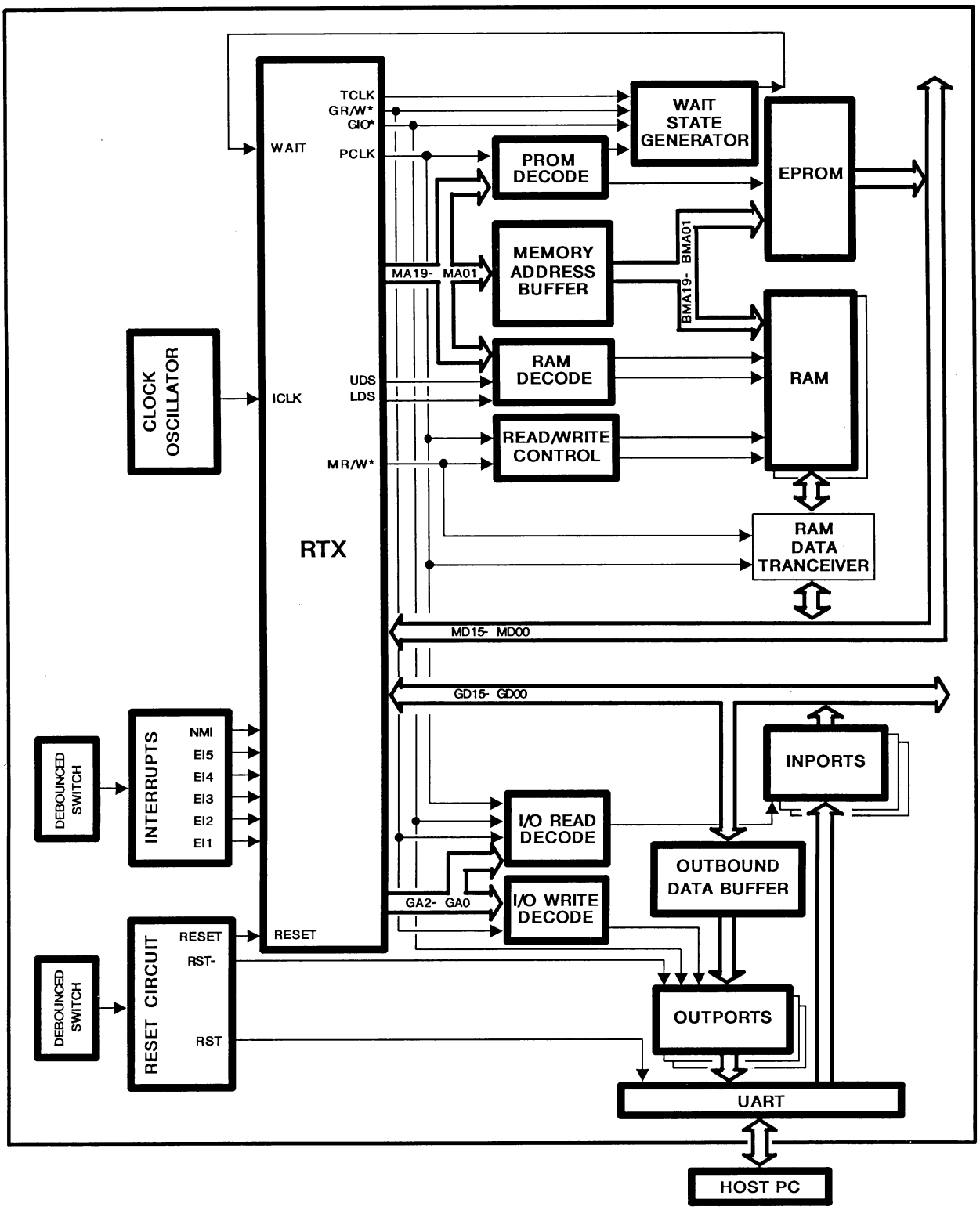
The primary function of the RTX 2000 Development Board is as a design and development tool through which users can define and prototype the RTX 2000-based control system which best suits their needs. As a test vehicle, the RTXDB provides the means to validate the performance capabilities of the RTX 2000 Microcontroller. In addition, the RTXDB can be utilized as a demonstration device and as a training/learning tool with which the user can learn and evaluate the RTX 2000 Microcontroller in a real-time application environment.

The RTXDB offers its user a complete development environment with extensive breadboarding and debugging capabilities. This flexibility allows application specific hardware and software to be fully integrated, developed, and tested before fabrication. The net result of utilizing such an environment for system development is a significant reduction in time and costs.

The RTX Development Board

The RTX Development Board utilizes the Harris RTX 2000 Microcontroller as its processor and a Crystal Clock Oscillator for its time base. The on-board memory provided with the RTXDB is composed of 32K bytes of high speed static RAM, 16K bytes of System EPROM, and sockets for up to 16K bytes of User EPROM. The System EPROMs contain the Host communications protocol and debug utilities. The two User EPROM sockets can accommodate 2K by 8, 4K by 8, or 8K by 8 user provided EPROMs to hold application software. The RTXDB includes an RS-232 serial port and the buffering and decoding circuitry necessary to support memory and I/O operations. Two input ports and two output ports can also be added to the defined circuitry by installing the necessary ICs into the locations provided.

In addition, a breadboarding area and a memory expansion area are provided on the RTXDB to allow expansion or modification to the board configuration. Using these areas, physical memory mapping, external interrupt configurations, Wait State generation, I/O functions, and memory capacity can all be user defined to meet the needs of a specific development application.



–, * Represent Active Low Signals.

RTX 2000 DEVELOPMENT BOARD FUNCTIONAL BLOCK DIAGRAM

Harris RTX Toolbox

The Problem Being Addressed

Designers of real-time systems have until now faced three choices: building around a standard microprocessor, a microcoded engine or recently, Reduced Instruction Set Computers (RISC).

The appeal of a standard microprocessor has been related mostly to the proliferation of hardware and software development systems and the comfort of writing the application software in either a high-level or native assembly language. Application software written in high-level languages assures the ease of portability of the code onto a newer design, preventing obsolescence. These advantages are often outweighed by the relatively low timing performance of the design, which is closely related to the average performance of the general purpose processor.

Microcoded bit slice designs usually offer considerably better speed performance than standard microprocessors. Unfortunately the amount of time involved in writing the applications code often by far exceeds the hardware design cycle and may make the product designer miss the market window entirely. The code written for a particular engine is totally non-portable, making the design relatively short-lived unless an upgraded version is periodically developed.

The introduction of RISC processors in the past few years has addressed many problems associated with general purpose and bit slice machines but may not provide an optimal solution for real-time control systems.

Real-time can mean two things. First, the processor is fast enough to process data and provide control signals to the target system "on the fly" or in "real-time". Second, and just as important, is "Time-to-Market" or the concept of a real-time product. This means that the best solution to any problem is a timely solution.

Most general purpose RISC processors use a reduced set of low-level instructions that help the chip optimize throughput. Often programs are big and development time is long. In addition many RISC engines require large areas of silicon and burn prodigious amounts of power, which precludes inclusion of application specific hardware on the same chip. Integrating large amounts of application specific logic at the board-level can be expensive and more important, time consuming.

The Harris Solution

Harris has developed a comprehensive solution to the problem of developing an ultra-high-performance, real-time control system called the Harris Real-Time Express (RTX) Toolbox. It is comprised of a FORTH Optimized RISC Engine, a family of support macrocells, an integrated computer-aided design (CAD) environment, a comprehensive set of Harris and third-party developed software and hardware development tools and Harris' advanced CMOS technology.

'Harris' RTX is the first microcontroller to directly execute a high-level language, the first processor to eliminate assembly language programming for real time applications, the first processor with an ASIC Bus, the first algorithmic co-processor architecture, the first RISC core for the ASIC market and the first dual stack-four bus architecture with such performance capabilities.

The Engine

The Harris RTX processor is the heart of the toolbox. It executes in hardware a FORTH-language virtual machine (see FORTH: A Software Development Environment). This machine directly executes FORTH, which is a high-level language optimized for real-time control applications.

Because the Harris RTX processor executes a high-level language it is capable of providing several key benefits to the user of this processor. First: software development time is greatly reduced. Second: software documentation and maintenance requirements become easier. Third: processor throughput as measured in equivalent MIPS (Millions of Instructions Per Second) can actually exceed the clock rate of the processor.

The third point requires further explanation. The Harris RTX processor is based on a RISC architecture. Adding to this is a massively parallel design philosophy which enables the processor to execute multiple operations within a single clock cycle. This compares to general purpose or microcoded bit slice designs which may take five or more clock cycles to accomplish the same task. The Harris RTX processor achieves this with a low gate count and without resorting to extensive pipelining or instruction queues, as in other high performance machines.

The absence of an instruction queue enables the Harris RTX processor to handle interrupt routines and return to its main program flow with only three clock cycles of overhead. This feature is critical to real-time control systems. The performance obtained by this feature can exceed that of many general purpose processors by an order of magnitude or greater.

Low-interrupt overhead and efficient subroutine calls encourage modular/structured programming techniques, recognized as contributing to rapid, efficient code development which is highly testable and maintainable.

Because this processor is fabricated in CMOS technology, power consumption is directly related to the clock frequency used. With the ability to achieve the same performance as other architectures operating up to 50 MHz, power consumption of RTX based processors can be as much as five times less than other architectures fabricated in an equivalent CMOS technology. Static circuit design in the Harris RTX processor means the clock can be stopped or slowed whenever the processor is idle, resulting in near zero power consumption levels. This feature will be important in military and industrial applications where power consumption can be a major concern.

Just how fast is the Harris RTX processor? Simulations and characterization of a pinned-out bare-bones processor indicate that performance levels in the 10 to 15 MHz range are possible. Equivalent MIP performance will be application-dependent, but a realistic estimate is in the 10-20 MIPS range, with peak execution for simple repetitive operations up to 30 MIPS.

Support Macrocells

A major feature of the Harris RTX toolbox is Harris' ability not only to put the Harris RTX processor core on a single piece of silicon but also to customize standard application-specific integrated circuit (ASIC) products by adding other logic functions to the processor core.

Harris currently has an industry-standard library of 7400 series logic functions as well as macrocells developed as part of its 80C88/80C86 family. These functions include timers, interrupt controllers, clock generators, parallel I/O controllers, UART's, LAN controller support cells (MIL-STD-1553) and many others. In addition Harris has developed Harris RTX specific macrocells designed for optimum performance in real-time control system applications. The first two of these are a 16-input interrupt controller and an integrated stack controller which includes variable depth RAM capability on-chip or off-chip. Stand-alone silicon has been produced on these two devices and pinned-out packaged units are currently available in a development board environment. It is anticipated that these pinned-out macrocells will be used for internal Harris development activities as well as made available to third party hardware/software development system suppliers and end users of Harris' RTX Toolbox.

Other macrocells which have been developed include a family of industry-standard and proprietary 16 X 16 bit multipliers. They will accelerate math intensive operations considerably over the already fast capability of the Harris RTX processor core - due to their ability to perform 16 X 16 multiply operations in one clock cycle of the processor.

How many macrocells can be integrated onto a single piece of silicon with the Harris RTX processor? It will depend on several factors, including the size of the particular macrocells used, the number of pin-outs, process and yield considerations, maximum permissible power consumption/frequency considerations and testability issues.

As an example, it is well within the capabilities of the technology to include the processor core, a 16 X 16 bit multiplier, an interrupt controller, a stack controller with

on-chip RAM, as well as several timer/counters, on a single piece of silicon. Harris is developing such a real-time control processor (RTX 2000) with these and other macrocells, for introduction this year.

Integrated Design Environment

The Harris RTX Toolbox is a synergistic set of macrocells implemented in standard cell technology, as well as an integrated Harris-SDA computer-aided design (CAD) hardware-software system which permits extremely efficient design, simulation, place and route, design rule check and pattern generation tape capability in an integrated design environment.

This means Harris can provide RTX-based standard products rapidly and efficiently once the architecture and capabilities of the product are determined. Initially Harris will utilize this capability internally for standard product generation. It is anticipated that strategic business partners with expertise in real-time control and DSP applications will be invited to use this ultimate system design capability to generate products specific to their own businesses. Finally, a semicustom capability will be offered to Harris' customer base in 1988 for generation of customer specific integrated circuits (CSIC).

System-Level Development Tools

Another important component in the Harris RTX Toolbox is Harris and third-party developed tools for both software generation and hardware evaluation and development.

Traditional microprocessor, bit slice and RISC architecture machines are heavily dependent on software compilers, operating systems, in-circuit emulators and the availability of high-level languages as the tools needed to get a working final product to the market. A Harris RTX-based product in many respects is no different in its need for software/firmware and a means of developing and validating an end-use product.

The Harris RTX core processor directly executes FORTH. A high-level language conceived expressly for real-time control applications, FORTH is an integrated software development environment (see: FORTH, A Software Development Environment). The Harris RTX processor executes the FORTH language directly, with no intervening generation of assembly code or microcode and their inherent generate, debug, validate and document problems.

FORTH code generated on an industry-standard PC using the Harris TFORTH target compiler generates directly executable code in real-time within our customers' time-to-market window. The TFORTH compiler has already been used to generate test vectors for debug and test of the Harris RTX pinned-out core. The code generated can be downloaded to a target system, or a PROM/EPROM programmer or evaluation/development boards developed by LDI of Ft. Lauderdale, Florida and Silicon Composers of Palo Alto, California.

Used in conjunction with Laboratory Microsystems Inc. PC/FORTH this software with a PC and LDI's development board provides a powerful, low-cost software/hardware development system.

As development of the Harris RTX Toolbox proceeds in the next 12 months, it is anticipated that additional third party and Harris developed tools will be available. Compilers for other languages are being developed. "C", "PROLOG" and "ADA" have been targeted for compiler development this year.

Because of the FORTH language, the Harris RTX toolbox and the ready availability of standard PCs, Harris expects that development systems costing $50K - $100K will be associated only with general purpose and microcoded bit slice machines. The efficiency of Harris RTX and the FORTH language can make development costs small in terms of time and money.

Design Example

As a simplified example, we will consider using a Harris RTX processor core with seven macrocells to generate a general purpose real-time control processor (see: Figure 1).

The stack controller consists of four sections expanded out for clarity in the figure. The return stack control consists of a stack pointer controller and an associated RAM based return stack. Similarly the parameter stack controller provides control of the RAM-based parameter stack.

REAL TIME CONTROL PROCESSOR

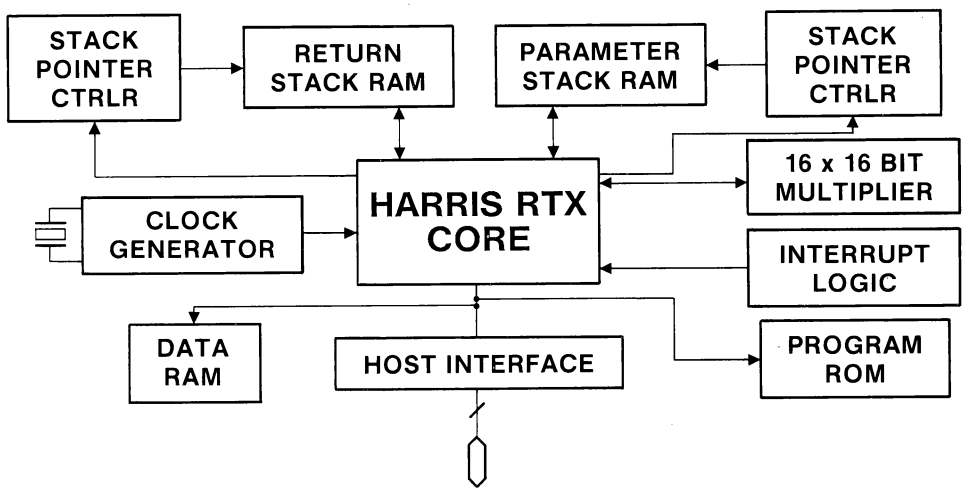


FIGURE 1.

The FORTH language and the Harris RTX processor are stack-based. The Harris RTX core will always require some type of stack controller whether on-chip or off-chip. In this example, 32-word deep stacks would be sufficient for many applications. It would be quite simple to scale the depth of the stack to 64, 128 or even 1K words on-chip. This capability is a result of the standard-cell nature of the Harris RTX Toolbox macrocells and the compilable RAM feature available as part of the toolbox.

Similarly, off-chip RAM could be used with appropriate stack expansion signals. These could be industry-standard static RAM.

The clock generator macrocell shown could be the crystal controlled Harris 82C84A macrocell or the designers' own clock generator circuit using available 7400 series cells.

The data RAM and program ROM can be on or off-chip with a 2K byte RAM and 8K byte ROM easily implementable on-chip using the toolbox RAM/ROM compiler. Larger sizes are feasible even on-chip, depending on die size/cost constraints. Off-chip memory could be configured up to 16M bytes using address extension capabilities already defined in the toolbox.

The host interface macrocell is application-dependent and might be custom designed for the particular application in mind. Harris is currently developing a universal host interface which permits the processor either to operate as a standalone processor or as a high performance co-processor loosely coupled to a general purpose processor such as an 80386 or 68020. This interface would permit either processor to access common memory and permit the host processor access to the Harris RTX co-processor on-chip data RAM.

Of course, the Harris RTX co-processor could include on-chip program RAM, in which case the HOST processor could program the Harris RTX co-processor to execute tasks it cannot handle due to speed requirements.

Next, almost all control system processors would require an interrupt controller. Harris has developed a 16-input controller which has undergone validation or, if the application requires, a custom interrupt controller.

Finally, no real-time control/DSP engine would be complete without a hardware multiplier. The 16 X 16 bit multiplier currently available operates in less than 40ns so that a multiplication operation can be executed in less than one processor clock cycle. If high speed math is not required, the multiplier could be left off. The core itself is capable of executing a fixed point 16 X 16 multiply in 20 clock cycles and a 32-bit square root with a 16-bit answer in less than 25 clock cycles. Both result from special on-chip hardware embedded in the core.

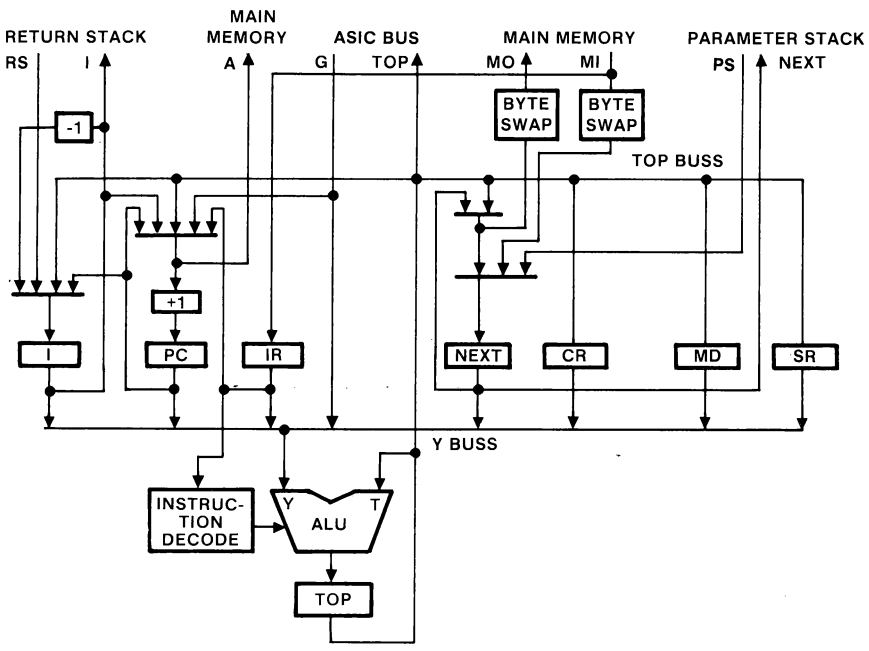
A chip similar to this example is being developed for introduction in 1988 as a standard product. Useful as a technology evaluation tool, it can also find use in such state-of-the-art applications as digital filtering, robotics, graphics and other real-time and DSP applications.

How Will Harris RTX Be Offered?

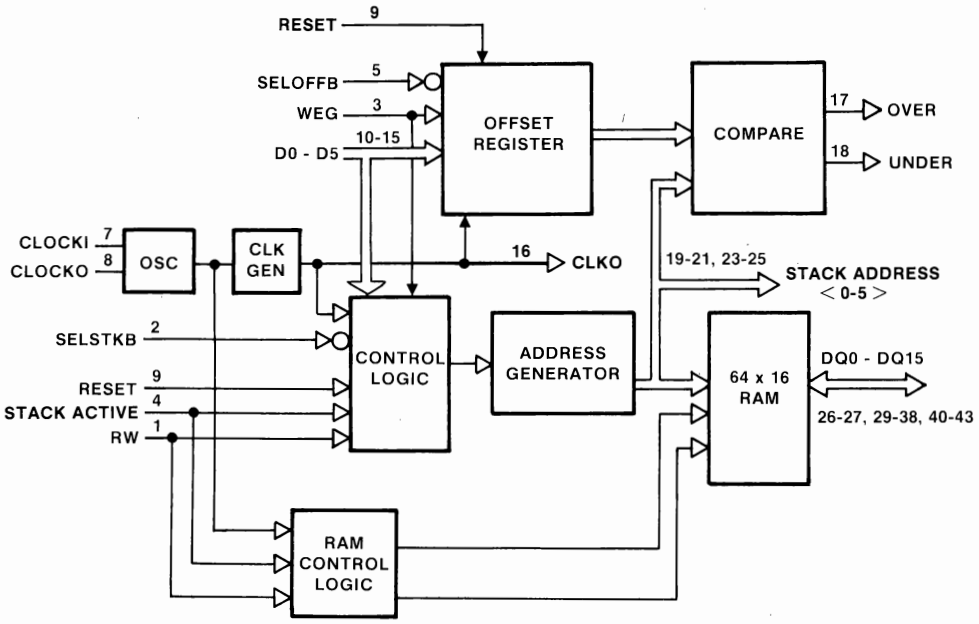
The Harris RTX core has completed development along with many of the macrocells. Other components of the Harris RTX Toolbox are similarly well along in development. Harris will be using the toolbox to generate Databook Standard Products during the coming months. Pinned-out versions of the macrocells will be used for validation, or as tools for development of new concepts and for third party software/ hardware development tool vendors.

Selected customers will also have access to these pinned-out macrocells. A complete Harris RTX Toolbox capability for the semicustom market is expected to be available by mid 1988.

RTX PROCESSOR

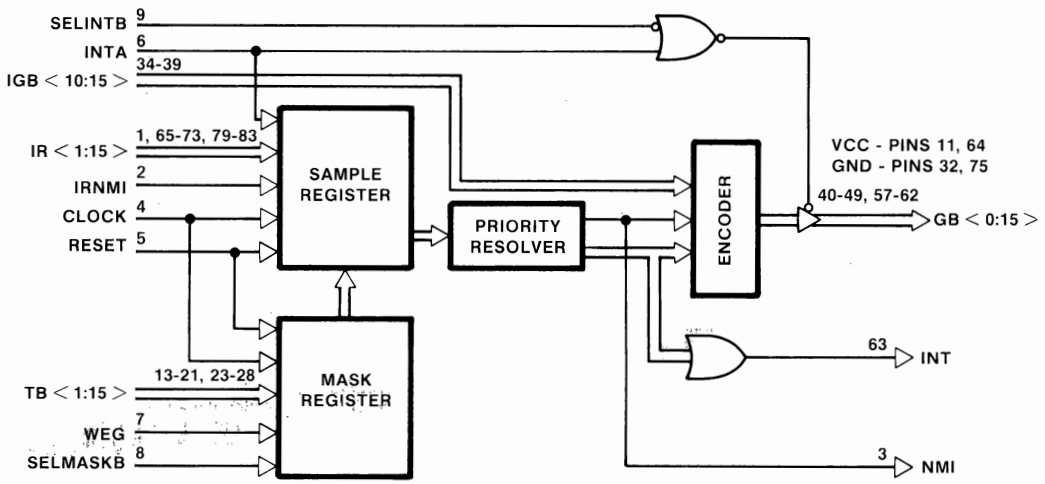


RTX SINGLE TASK STACK CONTROLLER WITH 64 X 16 RAM

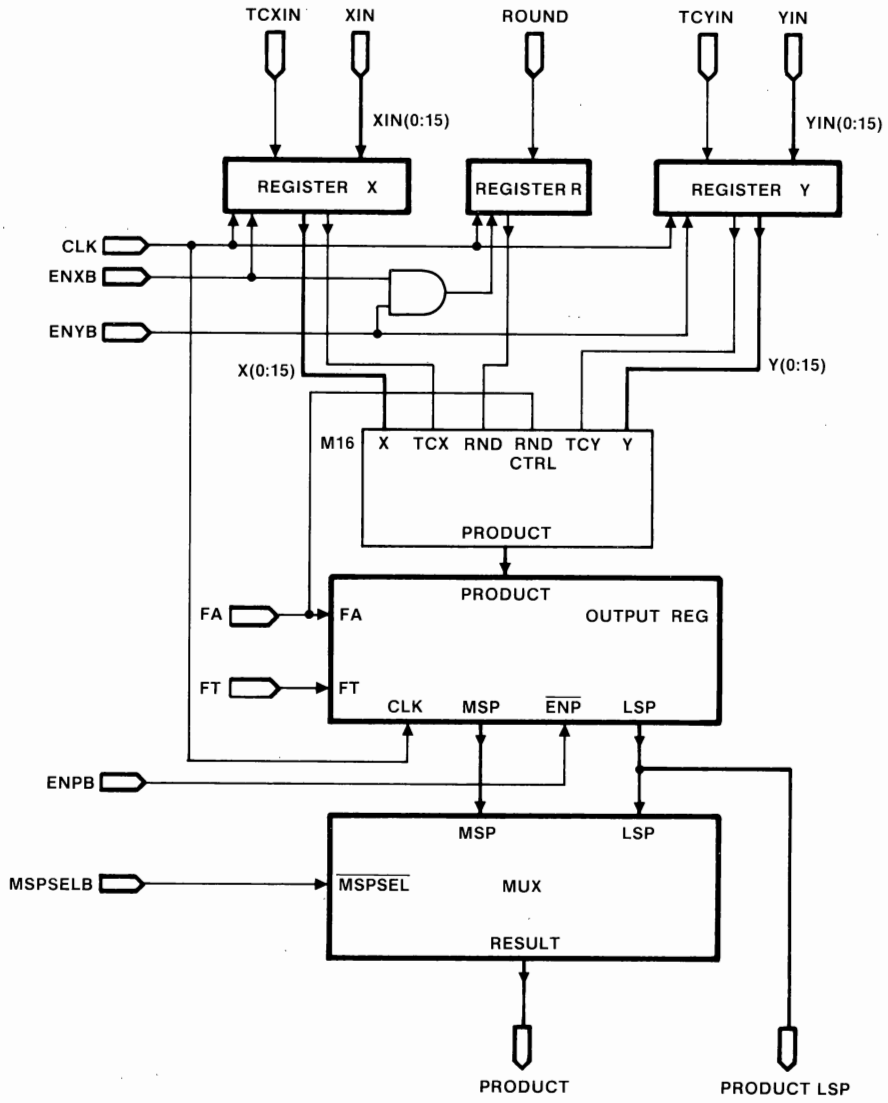


VCC - PIN 44
GND - PIN 22

RTX INTERRUPT CONTROLLER



RTX 16 X 16 MULTIPLIER



FORTH: A Software Development Environment

FORTH Introduction

FORTH is a fully integrated development environment, designed to speed software generation. Unlike most software development packages, FORTH is a complete development environment system: not simply a compiler or assembler.

Traditional computer systems consist of an operating system (such as VMS, MS-DOS), assembler (macro, ASM-80), linker/loader (LINK), high-level language (PASCAL, C, FORTRAN), a debugger and utilities. FORTH contains these in a single environment, including an operating system, compiler, assembler and programmer's toolbox.

This frees the programmer from having to learn a special language syntax for each step of development. FORTH can satisfy all programming needs, ranging from assembly to data base support.

The standard non-FORTH method for developing software consists of the following iterative cycle:

1) Edit a source code file with an editor program.
2) Assemble or compile the source code program with a compiler or assembler program.
3) Link the object code generated by the compiler program with a linkage program.
4) Invoke the Loader program to load the executable image generated by the Linker.
5) Run the program under the control of a debugger program, which provides some interaction to the programmer during the test phase.
6) Return to step 1 if the programmer finds an error.

The above sequence requires intimate familiarity with the syntax of each program needed to design and execute the program being developed - and the operating system, which controls each program. Many such programs have an exhaustively detailed set of instructions and options. Realistically, a programmer needs a considerable amount of time to learn and master them.

FORTH provides all of the separate development programs in a single, well-integrated package with a single syntax.

The average programmer using FORTRAN will create approximately 40 meaningful programs in his or her career. FORTH was designed to increase the programmer's productivity by as much as 10 times. Without the need to spend time manipulating a great many individual programs in the software creation cycle, the programmer has more time to actually solve problems.

FORTH is highly interactive. Unlike most other languages, FORTH lets the programmer interact directly with the execution of the software application - allowing fast prototyping and testing of new solutions to problems. This reduces the likelihood of incorrect solutions going undetected until the application is complete.

Interactivity also allows smaller sections of the program to be debugged. Fewer paths go untested, insuring accuracy of the solution. The interactive environment makes the program development cycle more intuitive. The programmer's concentration isn't interrupted between creation and execution. This prevents the programmer from losing a train of thought during long compilation and link times.

FORTH uses stacks to communicate parameters between routines and the outside world. It uses postfix (RPN) notation for math operations. Because FORTH is a stack-based virtual machine, the most natural syntax for communication is postfix; 2 X 3 becomes 2 3 X.

FORTH encourages modular programming by relying heavily on subroutine calls. In FORTH, these subroutines are called words. FORTH also consists of a dictionary divided into vocabularies. When a word (subroutine) is defined it is placed into a vocabulary in the dictionary, helping create application-oriented solutions.

FORTH can create readable code devoid of any implementation syntax. It communicates the problem's solution, not the implementation. This encourages easily read and maintained code.

FORTH produces compact reentrant code. Common code is actually reused, compacting the application into less space than normal machine language. Typically, the larger the application, the better the compaction.

FORTH also gives the programmer complete control over the architecture. Critical applications requiring machine program language routines can be coded directly in line. A programmer no longer needs elaborate schemes to link machine language routines to high-level code.

FORTH is fast. It executes at approximately 80 percent of machine code. This ability to produce high-speed compact applications makes FORTH ideal for real-time control applications. FORTH provides complete control of the machine.

FORTH, unlike other high-level languages, doesn't require an operating system. This allows complete software control of the hardware and eliminates difficult system-level calls and programming tricks designed to fool the resident operating system. By avoiding these pitfalls, applications perform more efficiently and predictably.

FORTH is available for every commercial CPU from the ILLIAC to the 68000. This means that no matter what the target system, the tools are available. This eliminates wasting time relearning tools before programmers can become productive.

FORTH provides many of the system primitives needed for the development of a multi-tasking system. Because FORTH is based on a software architecture model, it easily lends itself to system reconfiguration. This means that an application that is currently performed in a single task environment can be separated into functionally autonomous tasks. This process allows a programmer to independently think and code functions using a simpler, more modular approach. If the particular FORTH in use is not currently multi-tasking, a programmer can provide the super-structure needed.

This ability to modify FORTH's inherent structure is called extensibility. This is one of the most powerful features of FORTH, that few other languages provide. FORTH's extensibility feature is actually a description of machine architecture in software, called the "Virtual Machine" concept. The overall operation of FORTH code is controlled by this software description.

Since this is a software implementation of a machine, it may be altered by the programmer to perform differently, depending on the application. This modification may range from simply adding a new control structure to the language, to making FORTH a multi-tasking operating system. No longer does the fixed architecture of a traditional machine, or instruction set of a high level language determine the implementation of a solution. Now, the solution determines the implementation. This is a powerful tool in the hands of a software engineer.

FORTH History

FORTH was invented by Charles Moore, a programmer focusing on real-time applications, who created it via the traditional program development cycle. This vicious cycle of complex instruction sets, utility syntaxes and operating system knowledge prompted Moore to develop FORTH, supplanting the great array of program development tools and associated instructions.

FORTH first appeared on the IBM 1130 and was first used at the National Radio Astronomy Observatory in 1971 to perform radio-telescope data acquisition. Today's modern FORTH, with dictionary, compiler, assembler and multi-tasking capability first appeared on a PDP-11 at Kitt Peak Observatory. FORTH Inc. introduced its first FORTH development system in 1976. Today, FORTH is available for practically all CPUs from a multitude of software vendors.

Sales Offices

U.S. HEADQUARTERS
Harris Semiconductor
2401 Palm Bay Road
Palm Bay, Florida 32905
TEL: (407) 724-7418

EUROPEAN HEADQUARTERS
Harris Semiconductor Ltd.
Semiconductor Sector
Eskdale Road
Winnersh Triangle
Wokingham RG11 5TR
Berkshire
United Kingdom
TEL: 0734-698787

FAR EAST HEADQUARTERS
Harris K.K.
Shinjuku NS Bldg. Box 6153
2-4-1 Nishi-Shinjuku
Shinjuku-Ku, Tokyo 163 Japan
TEL: 81-3-345-8911